2005

Scalable schemes against Distributed Denial of Service attacks

Kishori Nanduri
Louisiana State University and Agricultural and Mechanical College, knandu1@lsu.edu

SCALABLE SCHEMES AGAINST DISTRIBUTED DENIAL OF SERVICE ATTACKS

A Thesis

Submitted to the Graduate Faculty of
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering

in

The Department of Electrical and Computer Engineering

by
Kishori Nanduri
Bachelor of Technology (EEE)
Jawaharlal Nehru Technological University, 2002
August, 2005

To my parents and teachers for their love, encouragement & criticism.

ACKNOWLEDGEMENTS

I would like to sincerely express my gratitude to Dr. Durresi for his invaluable guidance and mentorship in my current research project. He was deeply instrumental in setting the direction of this research project. I am very grateful to Dr. Amawy for having given me the opportunity to work under him. He has been very encouraging and supportive of my work.

I thank my peers Vamshi and Ciby for their suggestions and help. I also wish to thank my parents, Mr. and Mrs.Sampath Kumar, for all their support and positive encouragement. I particularly wish to thank my father for making critical suggestions upon reviewing my thesis report.

I am also grateful to my friends and relatives for their moral support and encouragement.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Defense against Distributed Denial of Service (DDoS) attacks is one of the primary concerns on the Internet today. DDoS attacks are difficult to prevent because of the open, interconnected nature of the Internet and its underlying protocols, which can be used in several ways to deny service. Attackers hide their identity by using third parties such as private chat channels on IRC (Internet Relay Chat). They also insert false return IP address, spoofing, in a packet which makes it difficult for the victim to determine the packet's origin.

We propose three novel and realistic traceback mechanisms which offer many advantages over the existing schemes. All the three schemes take advantage of the Autonomous System topology and consider the fact that the attacker's packets may traverse through a number of domains under different administrative control. Most of the traceback mechanisms make wrong assumptions that the network details of a company under an administrative control are disclosed to the public. For security reasons, this is not the case most of the times.

The proposed schemes overcome this drawback by considering reconstruction at the inter and intra AS levels. Hierarchical Internet Traceback (HIT) and Simple Traceback Mechanism (STM) trace back to an attacker in two phases. In the first phase the attack originating Autonomous System is identified while in the second phase the attacker within an AS is identified. Both the schemes, HIT and STM, allow the victim to trace back to the attackers in a few seconds. Their computational overhead is very low and they scale to large distributed attacks with thousands of attackers. Fast Autonomous System Traceback allows complete attack path reconstruction with few packets.

We use traceroute maps of real Internet topologies CAIDA's skitter to simulate DDoS attacks and validate our design.

CHAPTER 1: INTRODUCTION

1.1 Background

Distributed Denial of Service attacks are one among the most malicious attacks in the Internet today. In a DDoS attack a myriad of compromised systems attack a target, causing it to crash or deny service to legitimate users. DDoS attacks overwhelm the target system with data such that the response time is slowed down or stopped altogether [1]. In order to create the necessary traffic an attacker installs DoS daemons on a large number of systems (agents). These agents either exploit vulnerabilities present at the victim or overload the victim with inordinate requests. The former attack causes the victim to reboot or crash while the latter causes the victim to utilize some of its critical resources to handle the attack traffic and deny service to legitimate users. In short, DoS effect is achieved when the service requests interfere with the victim's operation and make it hang, crash, reboot or perform unnecessary work.

There are two broad categories into which the defense against a DDoS attack falls: a) Prevention of the attack from happening b) Detecting the attack traffic and reacting to it. Proactive measures avert host based or network based attacks from compromising systems. Some of the preventive methods include keeping software patched up, ingress/egress filtering, source validation by using reverse turing tests, well organized networks with critical applications spread across several servers in different subnets, a firewall to prevent unwarranted intrusion and a demilitarized zone (DMZ) area for permission to outsiders.

Reactive measures constitute of two phases, detection and reaction. In the first phase, the attack is identified using signature detection schemes like Intrusion Detection System and Anomaly Detection. The second phase includes reactive methods such as adhering to the Disaster Recovery Plans of reinstalling OS and applications on compromised systems,

characterizing the attack traffic; filtering the malicious packets at the organization's firewall or informing the upper tiers of the malicious attack and blocking the packets; identifying the attack zone using trace back mechanism and informing the Autonomous System of the agents performing the attack.

The best mitigation strategy is to stop the attack from occurring at all. However it is impossible that proactive measures can impede every attacker. There will always exist operating systems & applications with glitches and servers with limited resources to handle requests. Reactive measures like traceback mechanisms play a crucial role when proactive approaches fail to block a malicious attack. Reactive measures may be the only means to identify and shutdown a zombie to prevent any further damage. A victim can identify an attacker solely based on the source IP address field in a packet and if this field is spoofed then measures to block attack traffic based on source IP address fail. Traceback mechanisms have routers mark partial information on packets traversing towards the victim, which then utilizes this information to reconstruct the path, the packets have taken through the internet to traverse towards it [2]. This combats the problem of IP spoofing as the path to the attacker is identified. Even finding partial path information would be useful because attacks could be throttled at far routers. Disregarding the problem of finding the person responsible for the attack, if a victim is able to determine the path of the attacking packets in near real-time, it would be much easier to quickly stop the attack [3].

This report, presents three novel and realistic traceback schemes namely Hierarchical IP Traceback (HIT), Fast Autonomous System Traceback (FAST) and Simple Traceback Mechanism (STM). All the approaches employ Autonomous System Border Routers (ASBRs) for marking as they are few in number, more powerful and have higher incentives to implement

traceback mechanisms when compared to normal routers. HIT reconstructs to the attacker in two phases, the first phase identifies the attack originating AS and second phase traces the attacker within an Autonomous System. In comparison with Authenticated Marking Scheme (an existing scheme), HIT involves lower overhead, reconstruction time, complexity and requires less number of packets to reconstruct to the attacker. FAST uses a node append mode of marking and requires a very few packets to reconstruct to the attack originating AS. This scheme is very efficient for attackers at large AS path lengths. Simple Traceback Mechanism can identify the attack originating AS in a single packet and the originating attacker in a few tens of packets. This scheme suffers from pollution attacks. As compared to HIT and FAST, STM has lower reconstruction time and computation overhead but suffers from pollution attacks.

1.2 Thesis Outline

This report begins with Chapter 1 on 'Literature Review' which includes sections on recruitment of agent networks, different types of DDoS attacks, the various toolkits used to launch these attacks over the years and the trend these attacks are likely to follow in future. Chapter 2, 'Defense Mechanisms - Preventive and Reactive' we discuss about the various defense mechanisms system administrators can adopt to protect their organizations against attacks. In Chapter 4 we give a contrast and comparison of various Tracing Technologies such as ICMP Traceback, IP Traceback, FMS, AMS & Pi. and discuss the general properties a traceback mechanism should have. In Chapters 5, 6 & 7 we give a description of the proposed schemes, Hierarchical Internet Traceback (HIT), Fast Internet Traceback(FAST) and Simple Traceback Scheme (STM) respectively. In these chapters we also present the packet marking algorithms and the reconstruction algorithms which can be deployed on the Internet routers and at the victim respectively. In these chapters we also detail on the inferences from the experimental results. In

Chapter 8 we describe the design of the software for the simulator. Appendices constitutes of simulations in Perl for each algorithm.

CHAPTER 2: LITERATURE REVIEW

The consequences of DoS attacks are catastrophic. Starting from the Morris worm that attacked the CERT Coordination Center in late 1980s to TCP/IP SYN flood attack in 1996, DoS programs have brought the Internet to its knees. Revenue, productivity, site performance are highly deteriorated and increased IT expenses only add to the mounting burden. Distributed computing and access to increased bandwidth have increased the level of sophistication causing DoS attack to be more powerful. The year 1999 witnessed the first large scale distributed attack (DDoS) against the IRC server at University of Minnesota [4]. The attack toolkits automate almost all the processes allowing the DDoS attacks to compromise nodes at an exponential rate and thus trigger large scale attacks. Trinoo, Tribal Flood Network and Stacheldraht were marked as the weapons of choice for DDoS attacks [4]. The motive behind DDoS attacks is only degrading to highest levels of unlawfulness. Attacks ranged from identifying loopholes and crashing a few systems for fun and bringing down the business of companies to stealing personal information of people. In March 2005, hackers hit California State University gaining personal information including names and social security numbers of 59,000 people affiliated with it. Earlier in the same year DSW shoe warehouse acknowledged stolen credit information from more than a hundred of its stores [5]. These attacks could get more lethal in conjunction with cyber terrorism, the use of computing resources to intimidate or coerce others. An example cyber terrorism could be hacking into a hospital computer system and changing someone's medicine prescription to a lethal dosage as an act of revenge [44]. Cyber terrorism could also be the next mode of revenge, terrorists can resort after the 9/11 attack. This only shows that attacks are getting very powerful and dangerous and that people have to take the necessary precautions.

2.1 Recruitment of Agent Network

Having understood the varied nature of DDoS attacks it would be useful to be aware of the whole process of a DDoS attack right from the recruitment of an agent network to the different tools used to wage an attack, the scale of an attack and the last but not the least defense mechanisms to fend off attackers. The following paragraph describes the process of recruitment of an agent network.

Depending on the scale and type of attack, the attacker intends to initiate, he searches for vulnerable hosts (hosts with high network bandwidth and poor administration), manually, semi-automatically or automatically, in a process known as scanning. Once the vulnerable nodes are identified DoS daemons are installed on these systems. Earlier it was the installation process that was automated, but now there are scripts to automate the search for vulnerable hosts and installation of attack code. Examples of automated scanning tools are *sscan*, *bots* and *worms*.

The various stages of the DDoS attack model as shown in Figure 2.1. The program (installed in attacking hosts) that performs the attack communicates with the communication channel.

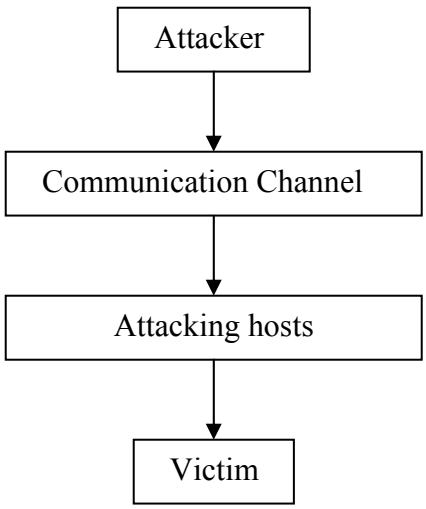


Figure 2.1 DDoS Attack Model

The communication channel can be a single host (IRC server) or a set of hosts (Handlers). The communication channel is a means for the infected hosts to declare themselves, to provide an indirect way of communication with the attacker [7]. The IRC model and Agent Handler model are examples of DDoS attack models with different communication channels.

In the IRC model, Figure 2.2, the communication channel is an IRC server or a set of IRC servers. This model uses bot as the automated scanning tools. A bot is a program on a compromised host, watching for encoded commands, like scanning a block of addresses or inviting a person to the IRC channel, to execute on the IRC channel. Once the bot obtains a set of vulnerable hosts they inform the attacker using botnet. The attacker creates a list of vulnerable hosts from the set addresses obtained from all the bots. The attacker then establishes communication channels between machines using IRC-based command to achieve an attack.

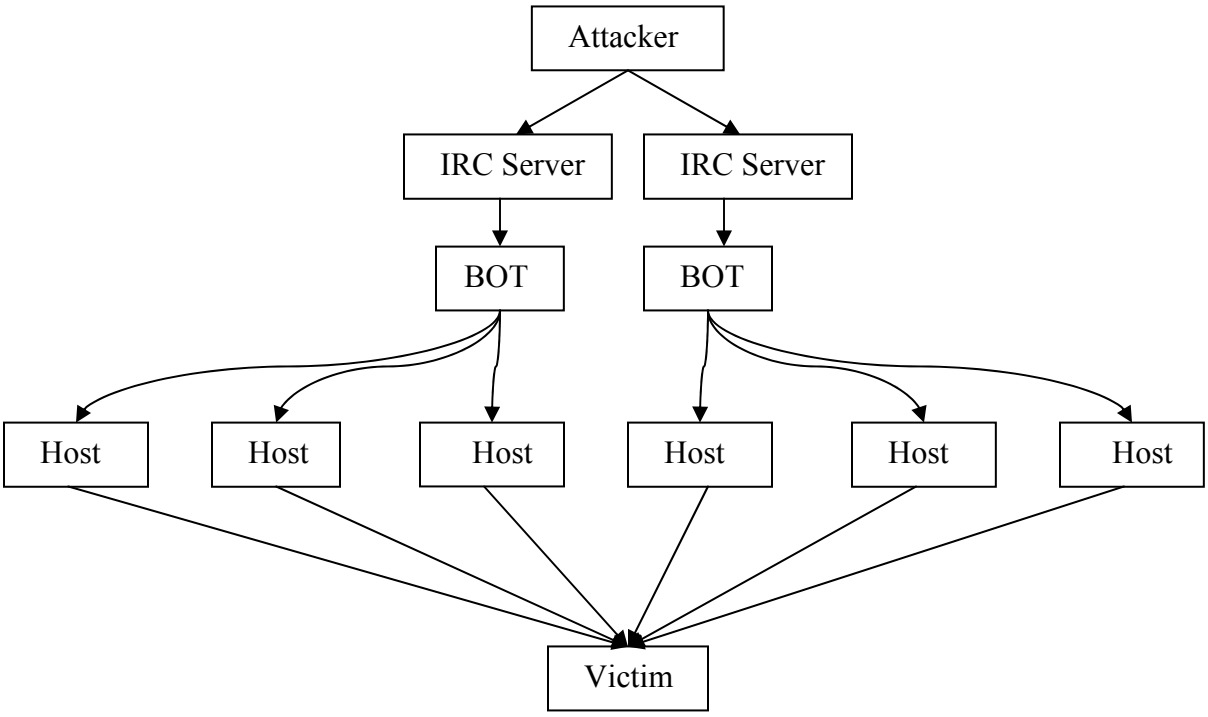


Figure 2.2 IRC Model

Thousands of people connect to an IRC server and form thousands of different chat channels. It is impossible to identify the attack channel in such a huge number of channels and therefore the identity of the attacker is left unknown [7].

In the Agent Handler, Figure 2.3, the communication channel consists of one or more hosts (handlers) that the attacker has compromised. The attacker issues commands to handler which inturn dispatches commands to the agent to launch an attack against the victim.

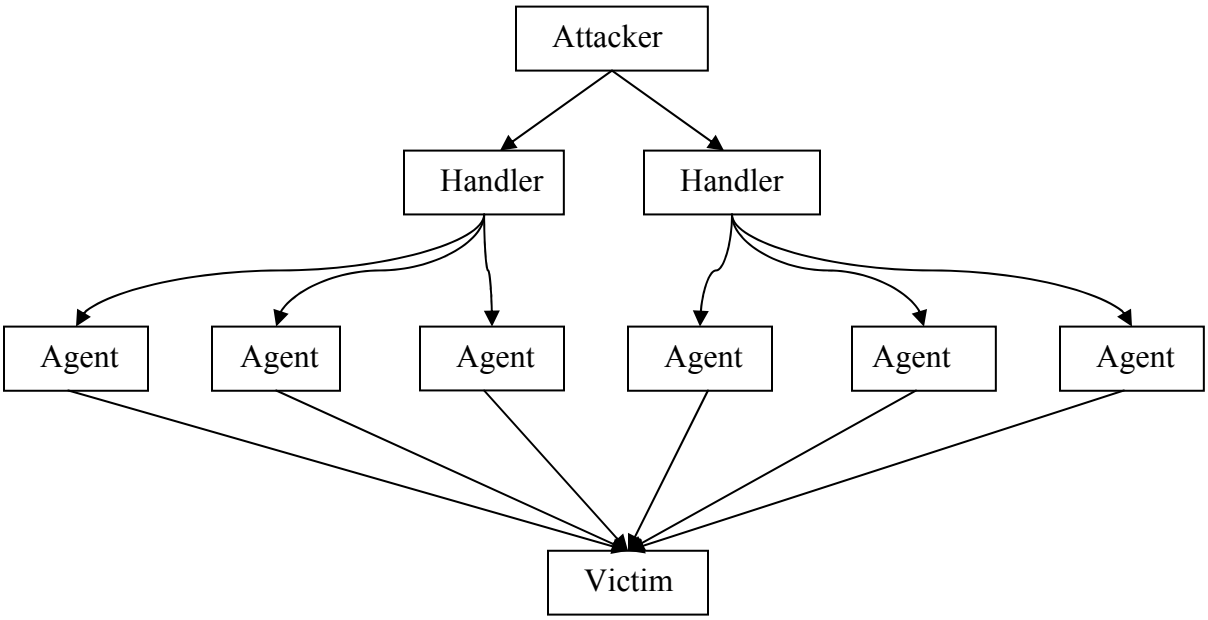


Figure 2.3 A Handler/Agent Control Structure

Like bots, worms are also automated client programs that propagate from one vulnerable host to another. Worms scan for vulnerable machines, exploit the compromised machines by installing the attack code and execute a code (called payload) to achieve an attack. Worms spread extremely fast because of parallel propagation pattern. The infected machines and the worm copies swarming the internet, grow exponentially [8].

2.2 Types of Attacks

The ultimate motive of a DDoS attack is to incapacitate a system's ability to process requests for its legitimate users. The various forms of attack are as follows:

2.2.1 Attack of Application

Every application can handle a certain number of requests per second. Attackers target a specific application and send packets such that this limit is reached. If the number of packets sent to the victim is more than its request acceptance rate the server can be overloaded causing it to hang or crash. This causes a denial of service to legitimate users.

Heavy provisioning, in the form of ample server and network capacity, can protect against application attack but can't guarantee immunity. Reacting to this kind of an attack by directing the traffic to a black hole or a sink hole might prevent the host from crashing but it doesn't rule out the possibility of denying service to legitimate users. There are not many defenses that can protect a victim against this attack. Reactive schemes such as traceback mechanisms can help identify the attack originating system and have it shut down.

2.2.2 Attack of Protocol

There are many vulnerabilities that exist in protocols. Attackers make use of these loopholes to launch a DDoS attack. One such vulnerability is with the transport layer protocol - Transmission Control Protocol (TCP). This protocol is responsible for the reliable transmission of packets between end systems. Any application that runs on a TCP protocol synchronizes session parameters between the server and client in 3 phases as shown in Figure 2.4 [9]. In the first phase the client sends a TCP/SYN packet requesting a service, to the server. In the SYN packet header the client provides a unique per connection number and a sequence number that keeps track of the amount and order of the data, sent by the client. Once the server receives the

SYN packet it allocates a TCB, transmission control block, storing information about the client. The server then sends a SYN ACK packet acknowledging the client of the service request it received and granting the service it requested. The packet header of the SYN ACK packet contains the server's initial sequence number. The client allocates a TCB and completes the opening of the connection by sending an ACK packet back to the server.

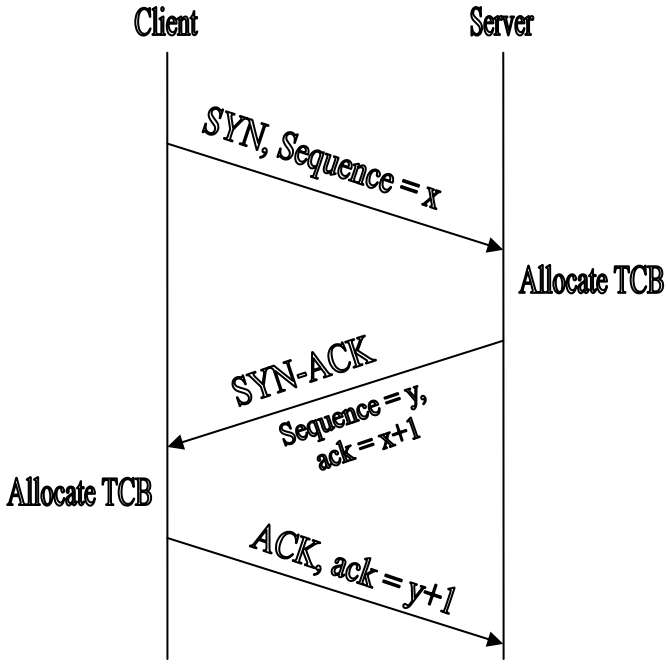


Figure 2.4 Opening of TCP connection: Three-way Handshake

The vulnerability an attacker exploits here is the allocation of TCB by the server before the client commits any of its resources. Figure 2.5 shows the handshake between the server and client, when the server is under an attack. If the server doesn't receive any ACK packet from the client it closes the connection after timeout expires. The resources of the server are tied up till it receives an ACK packet from the client. If the attacker identifies an open port and sends small number of requests with spoofed addresses before the timeout expires, the server will wait for

ACK from the spoofed clients and allocate memory for each of its spoofed clients [6]. In the process the victim exhausts its memory and crashes hence denying service to legitimate users. In another version of TCP SYN flooding the attacker can send enormous packets to random ports rendering the network resources useless rather than the buffer space.

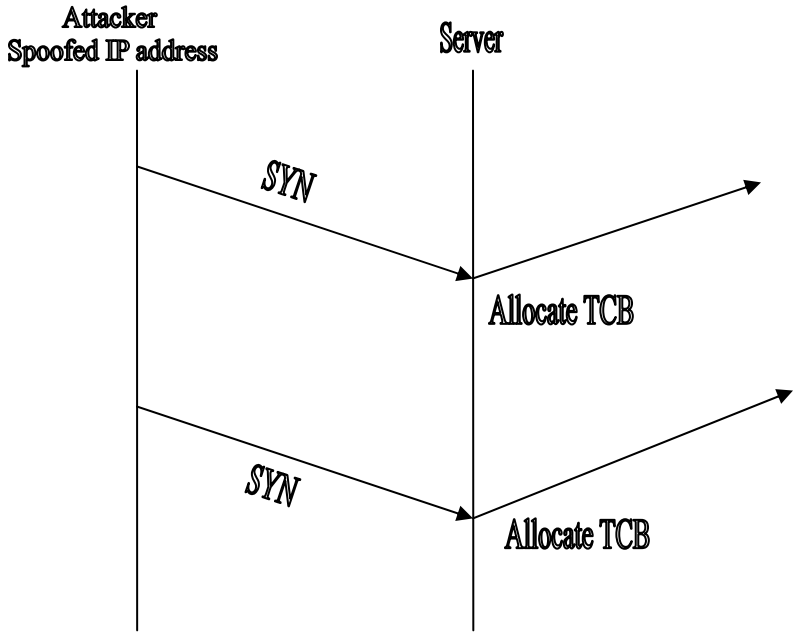


Figure 2.5 Handshake under an Attack

Protocol attacks target the asymmetry inherent in certain protocols by requiring the server to allocate its resources while sparing the client of its resources. Change in protocol specifications can be a solution but this will require all the systems to change their settings. Therefore, this doesn't seem like a feasible solution. The other solutions would be to create protocol patches that balance the asymmetry by requiring the clients to provide a proof of work or resource allocation through TCP SYN cookies and RST cookies, allocate micro blocks instead of the complete connection block for the client [9].

2.2.3 Attack of Resources

An attacker targets resources like the CPU cycles, computing power, router switching capacity & infrastructure to launch a DDoS attack. Requests, which involve huge mathematical calculations or vulnerable hash functions, require more CPU cycles as compared to the ordinary applications. A few requests can overwhelm the server rendering it incapable of processing legitimate requests. Many types of DDoS attacks are more or less overlapping. For example this attack can also fall in the attack of application category.

Router switching capacity is another resource attack, which targets at increasing the size of the routing tables with unimportant data and at times it leaves no room for critical entries. The router has to parse more information to forward packets to their destination. This causes delays in packet forwarding and sometimes packets get dropped before reaching their destination. The performance of routers is adversely affected due to such attacks.

Attack of infrastructure resources, exploits the network design of an organization. In a poorly designed network all the servers belong to the same sub network and an attack targeting routers, handling the subnet can easily cripple all the servers in that subnet. Proper design of the network, redundancy of information (backup servers & geographically distributed servers) and paths can be a solution.

2.2.4 Exploiting a Vulnerability

Attackers exploit vulnerabilities at the victim with well crafted packets. An improper handling of these packets disrupts the proper functioning of the victim. A well known vulnerability attack, Tear drop attack, exploits the markings in the fragmentation field of the IP header. When a router receives an IP packet that is too large for the network to which the packet is being forwarded, IP fragments the packet into smaller size packets to accommodate in the

network. At the receiving end the packets are reassembled into the original payload. The packet header has a fragmentation field, which keeps track of the order of the fragmented packets. If this field is manipulated for any two fragmented packets, reassembly of the packets at the victim causes a hang or crash of the system. Different versions of this attack exist with the fragments that indicate small overlap, a negative offset and different padding size or larger offset than the size of the IP header leading to the attacks Teardrop, bonk, boink and newtear[10].

For applications which run on TCP, a reliable protocol, which can discard the packet and request for the packet to be resent, will have a patch for this loophole but applications which run on unreliable protocols like UPD are still open to exploit.

In SSping attack, a variant of "Ping of Death" attack, a packet size greater than 65536 octets (Maximum size of a TCP/IP packet) sent to a victim will force it to halt. Land attack is another example of vulnerability attack exploiting the sources and destination address fields in an IP packet [10]. Unlike flooding attacks, vulnerability attacks become ineffective once a patch for these attacks has been developed.

2.2.5 Pure Flooding

In this attack, the intruder floods the victim with inordinate service requests. A multitude of illegitimate requests can consume many of the critical resources at the victim, rendering the victim incapable of processing legitimate requests.

As such there are no preventive measures against this attack. Reactive measures at the victim include detection and characterization of the attack traffic followed by contacting the ISP to filter, rate limit and direct the traffic to a black hole or sink hole. In the process of filtering packets which saves the ISP's and victim's resources the requests sent from legitimate users can also be denied and the motive of the flood attack is accomplished.

As most applications run on TCP traffic, UDP traffic can be filtered reducing the possibility of resources utilization by the UDP traffic. Smurf & Fraggle are two other flooding attacks, which use IP spoofing to swamp the network bandwidth. This type of an attack constitutes of three parties, the attacker, the intermediary (the broadcast address to which packets are sent) and the victim (the spoofed IP address). An attacker sends ICMP echo requests traffic to the intermediary address using a spoofed source address of a victim. This causes every machine on the broadcast network to receive the reply and respond back to the source address that was forged by the attacker [27]. The attacker can spoof a set of IPs and cause the intermediary systems to respond to the request causing the network to be swamped. Because the source address on the packets was forged, all the replies go back to the source address that was specified, which now becomes the victim's machine.

2.3 Attack Toolkits

An attack toolkit is a tool used to launch DDoS attacks using the above mentioned methods. It is a piece of code that is built into an easily usable package for launching attacks. Some attackers are sophisticated to originate their own attack code while most of them use existing codes. Moreover attackers write scripts to automate the installation process and this allows unsophisticated users to become DDoS perpetrators in a short time. Automation leads to widespread attacks thus increasing the intensity of destruction caused.

Early DoS attack technology involved simple tools that generated and sent packets from a single source aimed at a single destination. Over time, tools evolved to execute single source attacks against multiple targets, multiple source attacks against single targets, and multiple source attacks against multiple targets [10].

There exists a constant race between the attackers and the defenders. As soon as defenders come up with defense mechanisms, attackers find ways to bypass these defenses. Advanced toolkits are emerging to support the motivation behind every DDoS attack. Over the years attacks ranged from large scale UDP, TCP flooding with tools like Trinoo, Stacheldraht & Tribal Flood Network to financially motivated attacks with tools like *Agobot* and *Phabot.*

The trend of attack strategies is becoming more and more daunting as the sophistication of the tools being employed is scaling upwards. Table 2.1 gives a chronological order of the main tools and their attack methods.

Table 2.1 DDoS Tools and Flooding or Attack Methods

TOOLS	FLOODING OR ATTACK METHOD
Trin00	UDP
Tribal Flood Network	UDP, ICMP, SYN, Smurf
Stacheldracht and Varients	UDP, ICMP, SYN, Smurf
TFN 2K	UDP, ICMP, SYN, Smurf
Shaft	UDP, ICMP, SYN, Combo
MStream	Stream(ACK)
Trinity, Trinity V3	UDP,SYN,RST, Random Flag, ACK, Fragment
Phabot,Agobot	UPD, ICMP, Targa,

The following sections give a brief overview of the trends associated with the popular DDoS attacks.

2.3.1 Trin00

Trin00 was one of the prominent tools used to attack against the IRC server at University

of Minnesota in November, 1999. With IRC clients all over the globe to launch coordinated UDP flood denial of service attacks, all the resources got locked up and service was denied to all the legitimate users for days. The University counted around 2,500 attacking hosts, and that was an underestimate as the logs were not able to keep up with the attack [11].

Trinoo uses handler/agent architecture, with Trinoo residing in the agents and the handler commanding the agents to launch the attack. The following are the ports used for Trinoo attack.

Attacker - Handler destination port 27665/tcp

Handler - Agent destination port 27444/udp

Agent - Victims randomized destination ports

Agent - Handler destination port 31335/udp

All the communications with handler on port 27665/tcp require a password and all those on port 274444/udp require the UDP packets to contain the string "l44". The source address of trin00 packets is not spoofed; therefore finding the agent is not hard except that there will be many of them [12].

2.3.2 Tribe Flood Network (TFN)

TFN uses a similar handler/agent model as in trin00. TFN uses ICMP (Internet Control Message Protocol) echo reply packets with 16 bit binary values embedded in the ID field, and any arguments embedded in the data portion of the packet. The binary values, which are definable at compile time represent instructions between the handler and agent. The instructions when decoded can support several denials of service attacks like SYN floods, UDP floods, ICMP floods, and smurfing. For example the code 345 represents SYN flood attack. The TFN runs at the root therefore the source address can be spoofed making identification of the attackers even harder [12], [13].

2.3.3 Tribe Flood Network 2000 (TFN2K)

TFN2K appeared in December 1999 and was an improved version of the TFN attack tool. It includes a strong encryption (CAST-256) algorithm for control packets. TFN2K was designed to be compatible with almost all the operating systems including Linux, Solaris, Unix & Windows [14]. The communication between the handler and agent is done on randomly chosen protocols (TCP, UDP, ICMP) so that no recognizable pattern can be obtained. Therefore the packets pass through most of the filtering mechanisms. The communication is one way with no acknowledgments sent back to the handler by the TFK2N agent leaving no trace of the handler. The handler sends around 20 commands assuming atleast one command would have been received by the agent. TF2KN voids the concept on which the certain scan tools are built to identify the handler as the executables are stored as binaries in the handler. The scan tools detect certain matching string patterns to determine an attack agent. But with binaries there is string pattern matching tool is lacking. In the agents the binaries are extracted to executables and the pattern matching strings can be found.

In addition to flooding TFN2K can also attack by sending invalid packets, which includes invalid offset, protocol, packet size, header values, options, offsets, tcp segments, and routing flags [47]. This kind of an attack is called *Targa3* attack. Enough malformed packets can cause the system to crash.

2.3.4 Stacheldraht

This attack toolkit appeared during the same time as Trinoo and TFN. Stacheldraht combines the features of the two. Like TFN it can spoof the source address. It also uses encrypted communication between handler and agent and provides for automatic updates. With the update feature Stacheldraht can automatically replace agents with new versions and start

them. It uses TCP and ICMP on the following ports as default:

Attacker to handler: 16660/Tcp

Handler to and from agent: 65000/Tcp, ICMP ECHO_REPLY

The attacks performed by Stacheldraft are UDP flood, TCP SYN flood, ICMP Echo flood and smurf attacks. Many strings could be used in identifying the binaries in the file system. The NIPC tools would be effective in discovering these agents. However, if the binaries get compressed, this technique will not be successful [14]. Stacheldraht is far more difficult to prevent as it uses ICMP protocol for its communication. Through years Stacheldraht agent came up with varied version known as t0rnkit toolkit and Ramen worm.

2.3.5 Shaft

Shaft is another attack tool that uses a combination of features similar to Trinoo, TFN, Stacheldraht. The shaft network is made up of one or more handler programs and a large set of agents, which gives the ability to switch handlers and agent ports on the fly making it difficult to filter packets. The following are the ports it uses:

Attacker to handler: 20432/tcp

Handler to agent: 18753/udp

Daemon to handler: 20433/udp [15].

Shaft uses 'ticket' mechanism for keeping track of agents. Both password and ticket numbers have to match for the agent to execute a request. Agents can launch UPD flood, TCP SYN flood & ICMP flood attack. The flood occurs in bursts of 100 packets/host (This number is hard coded) with the source port and address randomized. The attacker can choose the duration, size of the packets and type of flooding attack towards the victim. Handlers have special command to obtain statistics on the attack traffic generated by each agent. This allows the attacker to keep track of the yield generated by the DDoS attack [16].

2.3.6 Mstream

Mstream is another attack tool which is less sophisticated when compared with other tools. The source code of this attack tool was recovered in its developmental stages with numerous bugs. Mstream agent was discovered in April 2000 on a compromised Linux machine in a major university. This tool slows down the machine by using up CPU cycles and eats up the network bandwidth. It generates a flood of TCP packets with ACK bit set. Handlers can be controlled remotely by one or more attackers using password protected interactive login. The communication between all the components in the attack can be changed at compile time. The handler does not require root privileges, but can login as a regular user on a Unix system to command the agents. The agent crafts the headers for forged packets and thus requires administrative privileges to perform the attack. The default ports are

Attacker to handler: 6723/tcp

Handler to agent: 9325/udp

Daemon to handler: 7983/udp

The handler expects commands to be contained entirely in the payload of the packet. Therefore the attacker can't use remote login like telnet to control the handlers. Like Trinoo, communication between handler and agent is accomplished through UDP datagrams [17].

2.3.7 Trinity

Trinity is the first DDoS tool that is controlled via IRC (Internet Relay Chat). After a machine is compromised by a Trinity agent it joins a specified IRC channel and waits for commands from the handler. This facilitates for a massive flood attack against a target machine. Trinity is capable of launching several types of attacks including UDP, IP fragment, TCP SYN, TCP ACK and other floods. Presently there are eight variants of Trinity attack found on Internet Relay Channels [45].

2.3.8 Agobot

Agobo*t* a descendent of Phatbot was widely used in 2003 and 2004. It is a blended threat which can perform a varied set of attacks for which it is called "Swiss army knife". It implements two types of SYN floods UPD flood and ICMP flood, the Targa flood (which includes random IP protocol, fragmentation and fragment offset values and spoofed source address), the wonk flood (one SYN packet, followed by 1023 ACK packets) and a recursive HTTP GET flood. This HTTP flood is difficult to obviate by filtering because the attack traffic pattern resembles the normal traffic pattern [18].

2.4 Attack Worms

2.4.1 Code Red

It is a self replicating malicious code that exploits a known vulnerability, buffer overflow in Microsoft Internet Information Server (IIS) Indexing Service DLL. The worm had a world wide impact in summer 2001, with 359,000 machines infected in 14 hours [19]. The *Code Red* worm attempts to connect to TCP port 80 on a randomly chosen host assuming that a web server will be found. Upon a successful connection to port 80, the attacking host sends a crafted HTTP GET request to the victim, attempting to exploit a buffer overflow in the Indexing Service. Systems not running IIS, but with a HTTP (HyperText Transfer Protocol) server listening on TCP port 80 will probably accept the HTTP request, return with an "HTTP 400 Bad Request" message, and potentially log this request in an access log. A packet-flooding DoS attack will be launched against a specific IP address embedded in the code [20]. If the exploit is successful, the worm begins executing on the victim host.

Code red 2, a disruptive version of code red, preferentially probed for machines on the same subnet and nearby subnets. As a result, once a machine within a corporate firewall is

infected, it would quickly probe every machine within the firewall and infect virtually the entire network [21].

2.4.2 Nimda Worm

The Nimda worm has the potential to affect both user workstations (clients) running Windows 95, 98, ME, NT, or 2000 and servers running Windows NT or 2000. End-users can get infected by either opening an e-mail attachment called README.EXE or by surfing on an infected web site, which might offer the user to download README.EXE. After the end-user has executed the file, the worm will continue to spread in two different ways. First it will send itself out via e-mails directed to addresses found from users e-mail inbox. Secondly it will start to scan random Internet addresses trying to locate vulnerable IIS web servers [21]. The actual spreading of Nimda can be split to four parts: Infecting files, Mass mailing, Web worm and LAN propagation as seen in Figure 2.6

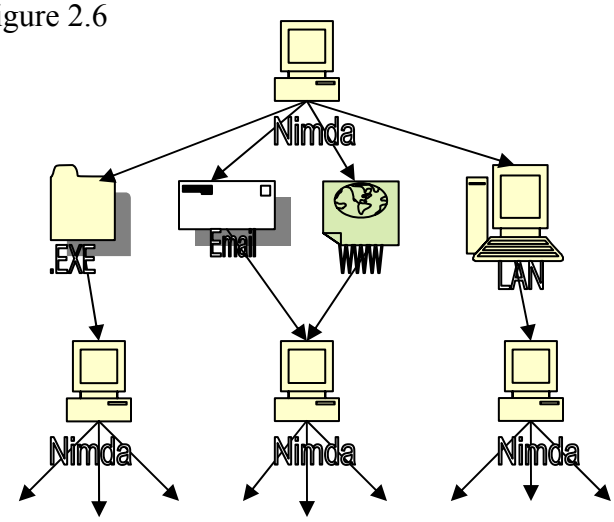


Figure 2.6 Spreading of Nimda Worm

Nimda is the first worm to modify existing web sites to start offering infected files for download. Also it is the first worm to use normal end user machines to scan for vulnerable web sites. Other worms which spread at such a rapid rate were vbs.loveletter worm and the w32.bugbear worm. These worms took the major anti-virus manufacturers off-guard.

2.5 Reason Why DDoS Attacks Are so Easy to Implement

- **Dependence on Internet**

  Internet has become a vital part of human life. Businesses like Ebay, Amazon and Netflix rely heavily on online transactions. Any infliction on the business servers can cause a huge revenue loss to such companies. Frequent attacks can damage a site's reputation and ward off customers and investors. Public network is the easiest & most accessible platform for the invaders to launch attacks to bring down any victim to its knees.

- **Availability & Simplicity**

  DDoS tools are available at large and can be easily downloaded. Automation of processes like scanning for vulnerable hosts, installing agents and launching an attack allows a novice to launch flood attacks with just a few commands. Attack toolkits are exceedingly simple to use and some of them have been here for years. Still, they generate effective attacks with little or no tweaking.

- **Security Pitfalls**

  Mostly, systems are connected to the internet via a public network. It is easy for attackers to access such machines and install vulnerability scanners. Most of these systems can be targeted as agents, to launch DDoS attacks because of their ineffective security measures. Systems should be setup for regular updates to fix patches, latest antivirus/anti-spyware software and a firewall to prevent unauthorized access.

- **IP Spoofing**

  IP spoofing is an easy technique for attackers to hide their identity. The IP address of the source or destination field in the packet can be tampered, causing the victim to assume the origin of the attack traffic to be legitimate. An attack launched can't be traced back as

the origin is spoofed. The tremendous growth in the number of systems connected to the internet has increased utility of IP spoofing. As such prevention of IP spoofing is not possible but approaches like ingress filtering and egress filtering minimize DDoS attacks. IP spoofing coupled with varied attack patterns and multitude of machines has made DDoS attacks extremely powerful.

- **Multitude of Agent Machines**

  With millions of computers connected to the internet and this number rising rapidly, the probability that the attacker finds novice internet users is high, increasing the strength of a DDoS attack. With so many agents, the attacker can vary his attack by deploying subsets of agents at a time, varying strategies or sending a few packets at a time. A few packets beat the concept of trace back and even if 10,000 machine identities are known, it is highly impossible to stop these sources. The solution would be to have secure systems preventing agents from being installed. With the existing internet layout, this seems too far a practical solution. Even if companies come up with solutions it will not be in the near future that systems will be impervious to attacks.

- **Limited Resources**

  Every system has limited resources and the attackers target these resources by overwhelming them with the high-volumes traffic. CPU cycles, network bandwidth and power are a few of the resources that are targeted. If legitimate traffic could be discerned from attack traffic then controlled traffic would not swamp the limited resources.

- **Legitimate or Illegitimate Traffic**

  Unlike the vulnerability attacks where the packets have to be crafted (Targa3, worms) to launch an attack, flooding attacks can compromise a large number of systems with

volumes of traffic. As this traffic behaves like legitimate traffic, the filtering process becomes extremely difficult.

- **Internet Topology**

  The current internet topology uses a *Hub & Spoke* with hubs carrying large volumes of traffic. If these hubs are taken down by an attack there will be a devastating effect on the global connectivity.

Summarizing all the aforementioned reasons we have, almost all the businesses rely heavily on internet. "Black hats" have an abundance of tools (easily available and automated) and accomplices (agents) to originate an attack and bring these companies down. A sufficient volume of attack with IP spoofing and a mixture of attack pattern can beat the filtering mechanisms and leave the victims swamped.

2.6 DDoS Defense Hurdles

- **Distributed Response**

  The challenges we are facing today is because of the vast nature of the distributed network. DDoS attacks utilize the distributed nature of the internet to launch an attack. With thousands of systems distributed all over the world to generate an attack, a response system to DDoS attack should be distributed and possibly coordinated. Since internet is administered in a distributed manner, a wide range deployment of a response system is highly unlikely.

- **Lack of Detailed Attack Information**

  It is believed that disclosure of any DDoS attack will damage the reputation of a company. Therefore many occurrences of DDoS attacks are not reported. Lack of appropriate information prevents researchers from knowing the current trend of attacks.

Information like the kind of attack, the number of agents and the time for which the attack persisted is very crucial for developing defense mechanisms.

- **Lack of Defense System Benchmarks**

  Presently there is no standardized procedure to determine the merits of any mechanism proposed by companies.

2.7 Case Study of IPremier

The following is a case study of a web-based commerce company, IPremier, which closed down after a DDoS attack was launched against it. A review on this case study will leave us with a situation which requires us to decide the appropriate defense mechanism against an attack.

IPremier was a venture started by two students in 1994, in Washington, Seattle. The company was one of the top two retail businesses selling luxury, rare and vintage goods on web. It was one of the few profitable entities in the so-called "new economy". Qdata, was the company in charge of providing internet security services including firewall service, filtering and blocking packets & monitoring traffic for IPremier. In the year 2003, a DDoS attack at a rate of one email/sec was launched against IPremier. A SYN flood attack lasted for an hour incapacitating the servers all the while. Qdata was not up to date with the software patches and intrusion detection signatures. It also couldn't maintain a proper log of the attack packets and hence was not able to locate the attackers.

With an intention to obtain the log files to trace back to the source of the attack IPremier left the attacked ports open for more than a week. The attacker behaved as normal traffic and installed an agent on the server through the port. This agent was remotely commanded to launch Distributed Denial of Service attacks against the competitors of IPremier. When the competitors

realized that they were being attacked by IPremeier, they believed that IPremeier had launched the attack as the source address of the attack packets was the IPremeier server. The competitor filed a case against IPremeier and had it shut down forever. A company that was at its pinnacle crashed for making an improper decision. It left the server ports unsecured with an urge to find the attacker.

IPremeier had two options to handle the attack situation. The company adopted reactive measures and left its ports unsecured with an intention to traceback the attacker. It could have adopted preventive measures and updated its security system instead of having left loopholes for another attack. For obvious reasons prevention would have been an ideal option but this is not always the case. Threre will always exist glitches in applications, protocols and operating systems which can be exploited to launch an attack

Rate limiting, packet filtering, and tweaking software parameters can in some cases, help limit the impact of DoS attacks, but usually only at points where the DoS attack is consuming fewer resources than are available. The open, interconnected nature of the internet and its underlying protocols, will always have scope for exploits and there is every possibility that preventive measures will fail. If an attack can't be prevented, there has to be an alternate countermeasure, one can resort to. In many cases, the only defense is a reactive one. Reactive measures help to locate the systems where the agent is installed. Once the attack traffic is detected and characterized at the victim, reactive measures should be taken at the earliest. Higher tiers can be asked to block unwanted traffic and with their cooperation the source of attack traffic can be tracked and shut down.

With the given explanation it is clear that an optimal mix of the preventive and reactive measures can be the best bet against a DDoS attack. The following chapter 'DEFENSE

AGAINST DDoS ATTACKS: Preventive & Reactive' gives an insight on the existing preventive & reactive measures against DDoS attacks.

CHAPTER 3: DEFENSE AGAINST DDOS ATTACKS: PREVENTIVE AND REACTIVE

There are two types of defense mechanisms against DDoS attacks: preventive and reactive. Preventive/Proactive approaches include all measures to avert an attack from happening. Reactive approaches are those procedures an organization adheres to once it discovers that some of its systems have been compromised [22]. Preventive and reactive measures are not mutually exclusive. An adequate allocation of resources for both the mechanisms is helpful in building a robust infrastructure for an organization.

Following are some preventive measures that can improve the security of systems and make it difficult for an attacker to channel a DDoS attack through them. It is of vital importance that these suggestions be implemented to protect the systems in an organization.

3.1 System Administrators

- **Updates**

  Updates are very crucial to correct flaws existing in an OS or other applications. Most vendors put out security fixes at a rapid rate, therefore applying the latest patches provides for a more secure system. These patches minimize a vulnerability attack but not a flood attack. A flood attack can be minimized by ensuring a robust design of an organization.

- **AntiVirus/Antispyware Software Updates**

  Signatures are patterns against which antivirus and antispyware programs match the files on the disk to identify as a virus, worm or trojan. Tweaking an existing virus can create a new version of the virus which can be left unidentified without the updates of antivirus softwares.

- **Block Incoming Packets Addressed to the Broadcast Address**

  In certain situations, a node in a network may require to send a broadcast message to its peers but there is no legitimate reason for an external device to send a broadcast message to every node on the network unless and until it wants to swamp the resources of the network through an IP directed broadcast, Smurf attack. Therefore Cisco routers, have an option to configure on every port of the router a 'no ip directed broadcast'. With this option enabled, the directed broadcast is routed through the network as a unicast packet until it arrives at the target subnet, where it is converted into a link-layer broadcast. The 'no ip directed-broadcast' command is the default in Cisco IOS software version 12.0 and later [21].

- **Firewall**

  A firewall examines packets passing to a computer or network and discards them if they do not meet certain criteria. Depending on the position of the firewall, access lists for filtering the packets are determined. The four types of firewall methods include Packet filtering, Circuit level gateways, Application level gateway and Stateful Inspection. In packet filtering, packets are filtered based on the ip address and port numbers. Known ports on which previous attacks have been performed should be blocked. Ports like the IRC port which has been associated with DDoS attacks and reserved IP address or Private IP address which will never run on public network should be blocked. New attacks which use a new set of port numbers, have to be detected first to be blocked. With tools for processor utilization or Network I/O performance and traffic monitoring, the statistics of attack traffic can be determined and excessive traffic can be blocked. In the Stateful inspection firewall method a connection is established only if the request is initiated from

within a network. This way if systems are scanned for vulnerabilities, only the address of the firewall is shown up and an intruder can not remotely access any system to install an agent or malware.

- **Ingress & Egress Filtering**

  Ingress filtering is a process of filtering inbound traffic. The ingress router of an ISP is connected to a company's network to filter data entering it allowing attack traffic to be filtered at a higher tier. Egress filtering is a process of filtering outbound traffic and it controls the data leaving a network. Egress filtering is achieved by setting outbound filters on the egress router to ensure that only assigned IP address space leaves the network. Therefore systems within the network with egress filtering can't launch an attack by IP spoofing [45].

  This kind of filtering is provided on the CISCO routers using access lists. Egress filtering is company dependent; hence a company should decide for itself what traffic would require leaving the intranet of the company. Based on the company's requirements routers should be configured to provide for egress filtering [21].

- **Demilitarized Zone (DMZ)**

  "A DMZ is a small subnetwork that sits between a trusted internal network, such as a corporate private LAN, and an untrusted external network, such as the public internet" [23]. Typically DMZ contains HTTP servers, FTP servers, SMTP servers & DNS servers which can be accessed by public traffic. Both external and internal machines can access servers in the DMZ. DMZs allow internal machines initiate requests outbound to the DMZ which in-turn responds, forwards or re-issues the request to the public as a proxy server but the LAN firewall prevents DMZ from initiating inbound traffic. DMZ is the

only zone public can access, they can't access the network of the company directly hence the internal servers are safe. In the event that an outside user penetrated the DMZ host's security, the Web pages might be corrupted but no other company information would be exposed [24].

- **Unnecessary Services**

  Services are programs that run when the computer starts up and continues to run as they aid the operating system in functionality. There are many services that load and are not needed which take up memory space and CPU time. Disabling such services free up system resources and speed up the overall performance of a computer. These services are frequently the target of attempts to compromise and deny service to the machine.

3.2 Resource Distribution within an Organization

- **Network Organization**

  A well organized network will not have bottlenecks that can be easily targeted. It will have all critical applications across several servers distributed throughout the network in different subnetworks and connected to the internet via different ISPs. The attacker has to overwhelm all the servers to launch a denial service attack. Since critical applications are spread throughout the network, servers affected by the attack can be isolated and replaced with a robust system without loss of service. The separation of critical services from noncritical services can be performed on many levels. Following is an example of the distribution of resources within an organization [25].

  - Separating public services from private services.
  - Dividing n-tier architectures into their components: web, applications and database servers.

- Using single-purpose servers for each service (SMTP, HTTP, FTP, DNS and others)

- Splitting Internet, extranet and intranet services.

- **Proof of Work**

  Asymmetric protocols consume more resources on the server side than on client side. These protocols are highly vulnerable as they require the server to allocate its resources without the client having to allocate any of its resources to furnish a request. The identification of the client is spoofed and attacker generates a large number of requests to tie up the servers resources rendering its services incapable for legitimate clients. An approach to protect against such attacks is to generate an asymmetry in favor of the server, requiring the client to allocate its resources before the request is granted. The server requires a proof of work from the client. A commonly used approach is to send the client cryptographic puzzles to solve which take up a fair amount of time and resources of the client. Verifying the answer doesn't require a lot of resource allocation on the server side. The cryptographic puzzles are not onerous for the clients to solve or notice but it definitely reduces the degree of a DDoS attack. A DDoS attack is slowed down drastically for illegitimate users to issue enough requests to cause a flood attack. This scheme works best for a small or medium scale DDoS attack [6].

- **Honeypots**

  Honeypots are fake computer systems that are used to collect data on intruders. A honeypot, loaded with fake information, appears to the hacker to be a legitimate machine. While it appears to contain operating system vulnerabilities that make it an attractive target for hackers, it actually prevents access to valuable data, administrative controls and other computers [26].

3.3. Detection Mechanisms

- **Intrusion Detection System**

  Intrusion detection is a first line of defense for a company's systems which detect unauthorized access attempts. There are two general types of intrusion detection systems (IDSs): network-based and host-based. A network-based IDS is a passive device that sits on the network and sniffs all packets crossing a given network segment. By looking at the packets, it looks for signatures that indicate a possible attack and sets off alarms on questionable behaviors. When it observes an event, IDS can send pages, email messages, take action to stop the event and record it for future forensic analysis. A host-based IDS runs on an individual server and actively reviews the audit log looking for possible indications of an attack. It also monitors key system files for evidence of tampering. It does not leave a network vulnerable between the moment of intrusion detection and the moment of consequential response [27].

  Most IDSs are built on two general technologies: pattern matching and anomaly detection. Pattern matching technologies have a database of signatures of known attacks. When it finds packets that have a given pattern it sets off an alarm. Anomaly detection systems determine what 'normal' traffic for a network is and any traffic that does not fit within the norm is flagged as suspicious [27].

- **Tuning System Parameters**

  With all the tools in place to hamper a DDoS attack, the victim should have provisions to detect the hoard of illegitimate packets hammering the perimeter of a network. Network traffic monitoring tools, CPU utilization tools and Network performance tools can come

handy to analyze the statistics by comparing with a model of normal traffic and block the traffic from disrupting the services or exploiting vulnerability [6].

- **Processor Utilization**

  Programs like ps, top, netstat, vmstat, are useful for checking processor utilization. These programs help to understand the CPU utilization for an application. If an application consumes a high amount of CPU time (90%), there is every possibility that a vulnerable application is being targeted. With a model of normal utilization it can be easy to determine if it was an attack.

- **Disk I/O Performance**

  vmstat, iostat and top are some useful programs which indicate the disk I/O performance of a system. If disk- monitoring programs, as iostat, infer high disk activity as compared to a model of normal disk activity, then it can be assumed that a vulnerable application is under attack.

- **Server Processes**

  Web Servers typically have one process for listening to requests and many child processes to handle actual HTTP requests. If there is a very high rate of incoming requests, the number of processes should be increased to prevent overloading of the existing processes. Programs like top, ps, and netstat help to check overloading of the server processes.

- **Muti-Router Traffic Grapher (MRTG)**

  "MRTG is a public domain package for producing graphs of various router statistics via a web page". It maps the inbound and outbound data throughput rates on the device it is polling. It also monitors CPU utilization of the device it is polling. MRTG gives a visual

representation of the traffic on the network. Attack graph is indicated by spikes in the graphs which can be compared to a model of normal traffic throughput to identify an attack. Any layman can find this simpler than understanding jargon of the output for command line programs like ps and top [28].

Once the attack has been detected the victim should resort to either contacting its ISP to filter the attack traffic or identifying the attacker through traceback mechanisms. A comparison of various traceback mechanisms is discussed in the next chapter.

CHAPTER 4: COMPARISON AND CONTRAST OF VARIOUS TRACEBACK SCHEMES

DDoS attacks have become highly prevalent recently due to their intractability and relative ease of execution [3]. Because of IP spoofing, the origin of the attack packets is forged to an incorrect address causing traceback to the origin impossible. Traceback mechanisms form a critical part of the defense against DDoS attacks. These mechanisms don't rely on the source address of the packets but on the path through which the packets have traversed which makes it possible to get closer to the origin of the attack. By following the stream of packets from router to router within the network it is possible to trace back and locate the particular source that might be conducting an attack. Figure 4.1 shows the Internet topology with attack traffic being represen

ted by an arrow and the trace back path from the victim to the attacker by a dotted line. Once the agent computer or the attacker itself is identified, the attack can be stopped easily and necessary legal actions can be taken against it. Even finding out partial path information would be useful because attacks could be throttled at far routers [3].

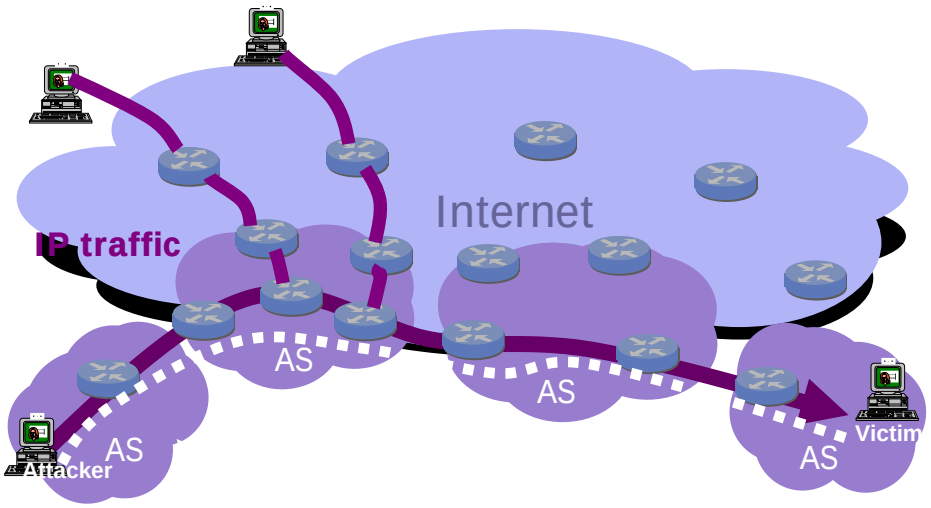


Figure 4.1 Internet Topology

The hierarchical attacking structure that detaches the control traffic from the attacking traffic effectively hides attackers even if the zombie computers are identified. Most of the methods are very limited because it is necessary to have access to all routers along the path from victim to attacker, and this is often not the case. The attacker's packets may be traversing a number of domains under different administrative control. So, it is necessary to contact other network administrators, who have other demands of their time and may not be able to respond to an attack against a target for which they are not responsible [29].

A number of recent studies have been carried to solve the IP traceback problem. Table 4.1 lists desired general properties of a traceback mechanism

Table 4.1 Comparison of various Traceback Mechanisms

Mechanism	Methodology	Legacy Routers	Network Overhead	Router Overhead	Number of packets	Scalability
FMS	Marking	No	Low	Moderate	High	No
AMS	Marking	No	Low	Low	High	Yes
iTrace	Out-of-band Marking	No	Low	Low	High	Yes
Algebraic	Marking	Yes	Low	Low	Very high	No
SPIE	Logging	No	Medium	Very High	Very low	Yes
FIT	Marking	Yes	Low	Low	Low	Yes
Hybrid IP Traceback	Marking and Logging	No	Low	Moderate	Low	Yes
Distributed Link-List	Marking and Logging	Yes	Low	Moderate	Moderate	No
PPPM	Marking and Logging	No	Low	Moderate	Moderate	No
Topology-based	Deterministic Marking	No	Low	High	Moderate	No
Goodrich	Marking	Yes	Low	Low	High	No

4.1 Related Work

Researchers have proposed various schemes to address the IP traceback problem. In this section, we identify the characteristics of a good traceback mechanism and we argue why previously proposed mechanisms do not achieve one or more of them.

- **Incremental Deployment**: A traceback mechanism should function even when only partially deployed across routers in the Internet. If a traceback algorithm does not provide benefits for incremental deployment, an ISP would have no incentive to start deployment. For various administrative and technical reasons, not all routers might be willing to participate in the traceback mechanism. Even if most of the routers are willing, it is unrealistic to assume that all routers will start implementing the mechanism at the same time. It is more realistic that initially only few routers participate and slowly others start participating. Unfortunately, neither edge marking mechanisms nor logging schemes provide strong properties for incremental deployment [30].
- **Few Packets:** Because of highly distributed nature of DDoS attacks, with each attacker sending only a few packets, a massive attack can be launched. Hence, complete traceback using only a small number of packets is especially useful for forensics. So far, only the SPIE mechanism enables single-packet traceback, and all other traceback approaches require on the order of thousands [3], [31, 32, 33] or tens of thousands of packets.
- **Router Overhead:** The changes needed at the routers and router overhead need to be minimal. If a traceback mechanism only requires a minimal hardware change and has a negligible overhead for packet forwarding, it is more likely to be accepted by router manufacturers and eventually by the ISPs. Probabilistic packet marking techniques impose much lesser overhead on routers than logging mechanisms.

- **Scalability:** Scalability in terms of the number of attackers and in terms of high link speed is essential. However, both PPM and logging traceback schemes suffer from scalability problems.
- **Reconstruction Complexity:** The time taken to reconstruct the attack graph should be small. Until the reconstruction is done the network and the victim will be under attack. Hence, a mechanism that only needs few packets, but lot of time to reconstruct might not be desirable.
- **Robustness:** A traceback mechanism is robust if it has very low rate of false negatives and false positives even in presence of large number of attackers.

4.2 Design Motivation

An Autonomous System (AS) is a group of IP networks operated by one or more network operator(s), which has a single and clearly defined external routing policy. The classic definition of an Autonomous System is a set of routers under a single technical administration, using an IGP (Interior Gateway Protocol) and common metrics to route packets within the AS, and using an exterior gateway protocol to route packets to other ASes. An Autonomous System Number (ASN) is a globally unique number in the Internet to identify an AS. An ASN is used in both the exchange of exterior routing information between neighboring ASes and as an identifier of the AS itself in the global Internet. AS numbers are 16-bit integers, assigned and managed by Internet Assigned Numbers Authority (IANA) [34].

Autonomous System Border Routers (ASBRs) are connected to more than one AS, to exchange routing information with routers in other ASes. ASBRs advertise the exchanged external routing information throughout their ASes. The traffic to/from an ASBR is controlled by

its ASBRs. Any traffic originating from (to) an AS to (from) a node outside the AS has to pass through an ASBR of the AS.

4.2.1 Marking with AS Numbers Has the Following Advantage

- The number of ASes is far less than the number of routers in the Internet. The Internet consists of around 35,000 ASes as compared to 1.6e+08 hosts [35]. The current rate of increase in ASes in the internet topology is 45% each year [36]. Since both the ASBR and internet routers are growing exponentially, obtaining an AS map of the Internet is feasible, while obtaining the Internet map itself is very difficult, if not impossible.
- As shown in Figure 4.2, in more than 99.5% of cases, a packet passes through less than six ASes before reaching its destination [37, 38].

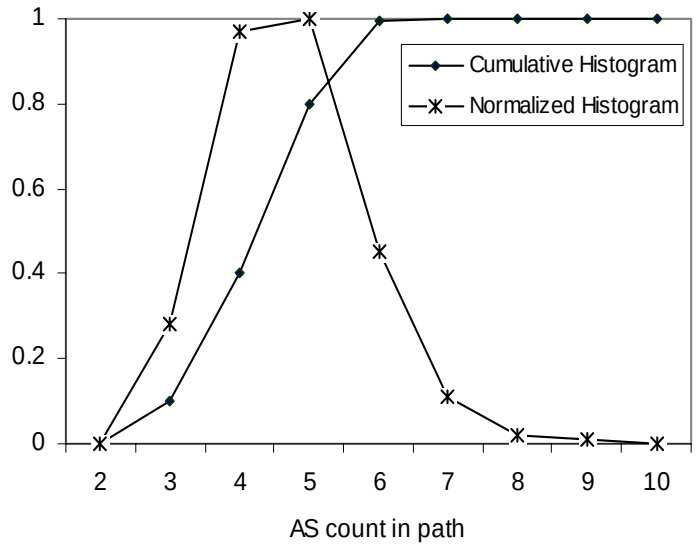


Figure 4.2 Normalized and Cumulative Histograms of Autonomous System path lengths

- Privately owned ASes may not always like to disclose their network details. If each router in the AS participates in the marking scheme, then one can easily infer the network architecture by observing the markings.

- AS number is 16 bits in length while IPv4 address is 32-bit long (IPv6 address is 128 bits). Thus, encoding AS number needs less header space than encoding IP address, consequently, with the same mechanism, AS path construction needs far less packets than whole network path.
- There is no scope for false positives because when a packet is marked, it carries the whole AS number of the router and the victim can reconstruct the attack path without any uncertainty. This relates to the third scheme, STM, proposed in this thesis report.

4.3 Header Overloading

We overload IP header to store router markings. Our mechanisms use 25 bits for marking.

- **The TOS Field**

  The type of service field is an 8 bit field in the IP header that is currently used to allow special handling of traffic (for example, minimize delay, maximize throughput). This field has been rarely used in the past and we believe that setting this field arbitrarily makes no measurable difference in packet delivery.

- **The ID Field**

  The ID field is a 16-bit field used by IP to permit reconstruction of fragments. Naive tampering with this field breaks fragment reassembly. Because less than 0.25% of all Internet trace is fragments [39], we believe that overloading this field is appropriate. A more in-depth discussion of the issues related to its overloading can be found in Savage's work [40].

- **The Unused Fragment Flag**

  There is an unused bit in the fragment flags field that current Internet standards require to be zero. Setting this bit to one has no effect on current implementations; with the exception that

when receiving the packet, some systems will think it is a fragment. The packet, however, is still successfully delivered, because it looks to those systems as though it is fragment 1 of 1.

4.4 Upstream Router Map

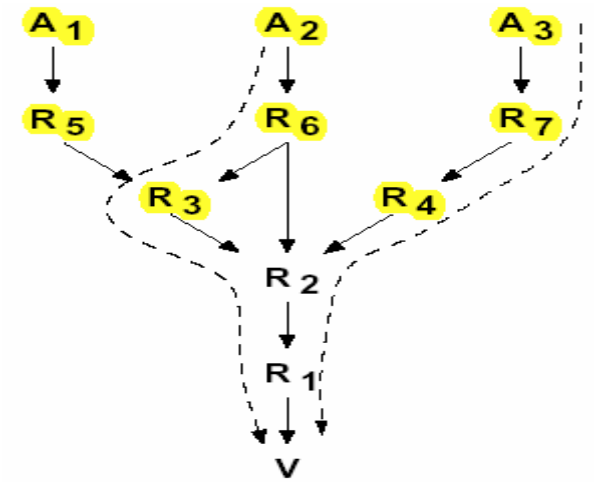


Figure 4.3 The Directed Acyclic Graph (DAG) rooted at V

In the proposed schemes, the victim has a map of the upstream routers. The Directed Acyclic Graph (DAG) rooted at V as shown in Figure 4.3, represents the network as seen from a victim V and a distributed denial-of-service attack from A2 and A3. V could be either a single host under attack or a network border device such as a firewall representing many such hosts. Nodes Ri represent the ASBR, which we refer to as upstream routers from V. For every ASBR, Ri, we refer to the set of ASBRs immediately before Ri in the graph as the children of Ri, e.g. R3, R6 and R4 are R2's children. The leaves {Ai} represent the potential attack origins, or attackers. The attack path from Ai is the ordered list of ASBRs between Ai and V that the attack packet has traversed, e.g. the dotted lines in the graph indicate two attack paths: (R6, R3, R2, R1) and (R7, R4, R2, R1). The distance of Ri from V on a path is the number of routers between Ri and V on the path. The attack graph is the graph composed of the attack paths. The packets are

referred to as attack packets. A router is a false positive if it is in the reconstructed attack graph but not in the real attack graph. A router is a false negative if it is in the true attack graph but not in the reconstructed attack graph [41].

4.5 Simulations

To test the behavior of all the proposed schemes in real settings, we conducted a set of experiments on simulated attacks using a real traceroute dataset obtained from CAIDA [42]. The traceroute dataset contains 127,634 distinct traceroutes from a single source with the average path length of 25. We use Cisco Border Gateway Protocol (BGP) RIBs collected from University of Oregon Route Views Project [42] to convert IP addresses to AS numbers and hence obtained the AS topology. The dataset had 7247 different ASes with an average of 4.6 ASes per path.

In all the tests, we assumed the single source of the traceroute dataset as the victim and the whole traceroute dataset as the map of upstream routers from the victim. In each test, we randomly selected a set of ASes ranging from 100-1500, as the attack originating AS and three destinations from each AS as attackers. We then simulated a Distributed Denial of Service attack which required attackers to launch an attack traffic against the victim. The attack packets were marked by intermediate ASBRs while traversing from the attacker to the victim. Simulations also included the reconstruction of the attack graph by the victim using the markings in the packets.

The System configuration on which these simulations were performed are as follows:

Hardware: Intel Pentium 4 machine with clock speed of 3 GHz and 2 GB RAM.

Software: Perl v 5.8.3 was used on a Red Hat Linux 3.2.3 Operating System.

CHAPTER 5: HIT: HIERARCHICAL IP TRACEBACK

In this chapter, we propose HIT, a new probabilistic packet-marking approach for IP traceback. HIT achieves effective attributes due to the fact that it mainly employs Autonomous System Border Routers, ASBRs, in the marking procedure. ASBRs are few in number and more powerful when compared to normal routers and have higher incentives to implement traceback mechanisms.

Hierarchical IP Traceback (HIT) is a much more effective scheme as compared to many other major traceback mechanisms such as Authenticated Marking Scheme (AMS), Fragment Marking Scheme (FMS) and Fast Internet Traceback (FIT). The efficiency of HIT is measured based on the properties like incremental deployment, network overhead, router overhead and number of packets. In contrast to previous work, HIT simultaneously achieves all the following properties: tens of packets to trace an attack path, scales to thousands of distributed attackers, is incrementally deployable, and requires no per-flow or per-packet state at routers.

HIT implements Probabilistic Packet Marking scheme in two phases of trace back. The first phase identifies the attack originating ASes while the second identifies the attacker within each AS. Once an attack originating AS is identified, the corresponding administrator can be notified who then proceeds to traceback to the attacker network location. The hierarchical nature of HIT makes reconstruction very efficient and scalable.

The marking approaches for identifying the attack originating AS and the attacker itself are termed as Inter-AS level marking and Intra-AS level marking respectively. The Inter-AS level marking employs edge sampling algorithm while the Intra-AS level marking utilizes node sampling. To allow for practical deployment, HIT overloads 25 bits of the IP header (the IP identification field (16 bit), TOS field (8) and the unused Fragment Flag (1 bit)) to reserve bits

for 5 fields of marking. Figure 5.1 depicts our choice for partitioning (25 bits) of the IP header as follows:

- 11 bits for the edge marking, the components of which are 11 bit hashed ASNs
- 3 bits for distance field represents a maximum of 8 ASes a packet can traverse from the source to the victim. In more than 99.5% cases, a packet passes through less than 6 ASes before reaching its destination.
- 3 bits for generating 8 hash functions.

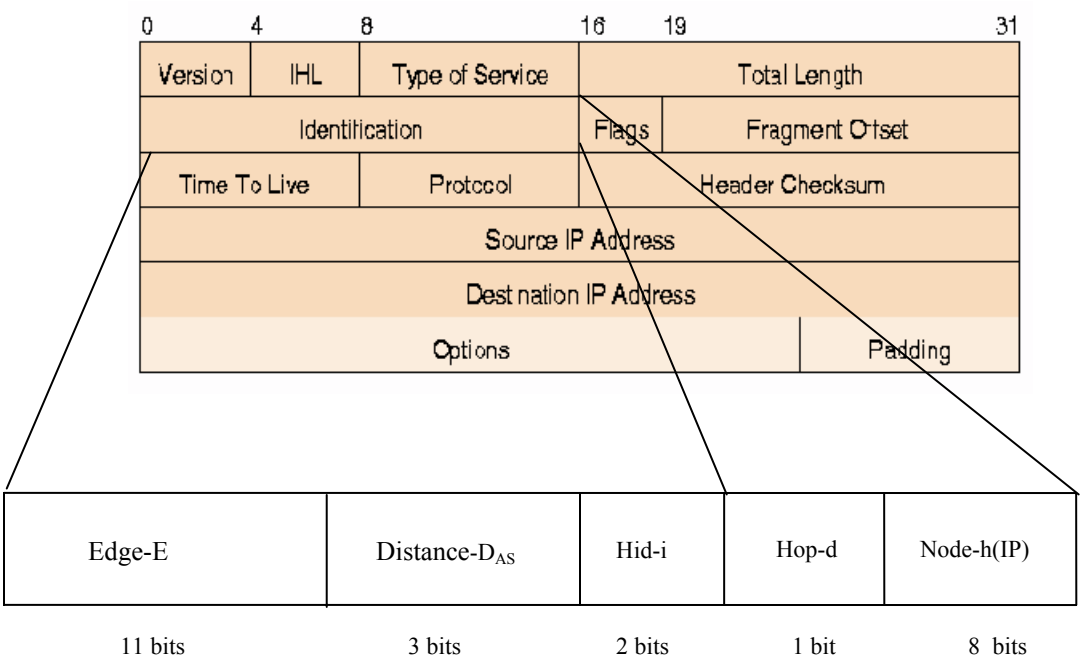


Figure 5.1 Overloading of the IP header

5.1 Inter-AS Marking

At this level only the ASBRs are involved in marking. An ASBR marks a packet with its ASN (Autonomous System Number) only if the packet is forwarded to a router belonging to another AS as shown in Figure 5.2 [34]. Thus, a packet may get marked only when it exits an AS.

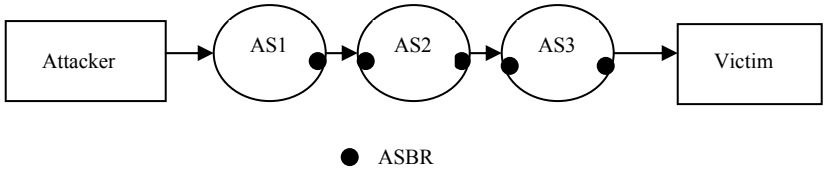


Figure 5.2 Marking of a Packet Traversing from Attacker to Victim

The Inter-AS marking approach is similar to Authenticated Marking Scheme – II [41], but instead of encoding the IP address of a router, we encode the ASBR, $h^i\left(AS_j\right)$. This scheme reserves bits for three fields, the Edge, Distance and Hash identification field for marking. To accommodate for these markings, bits of the IP header are overloaded as shown in Figure 5.1.

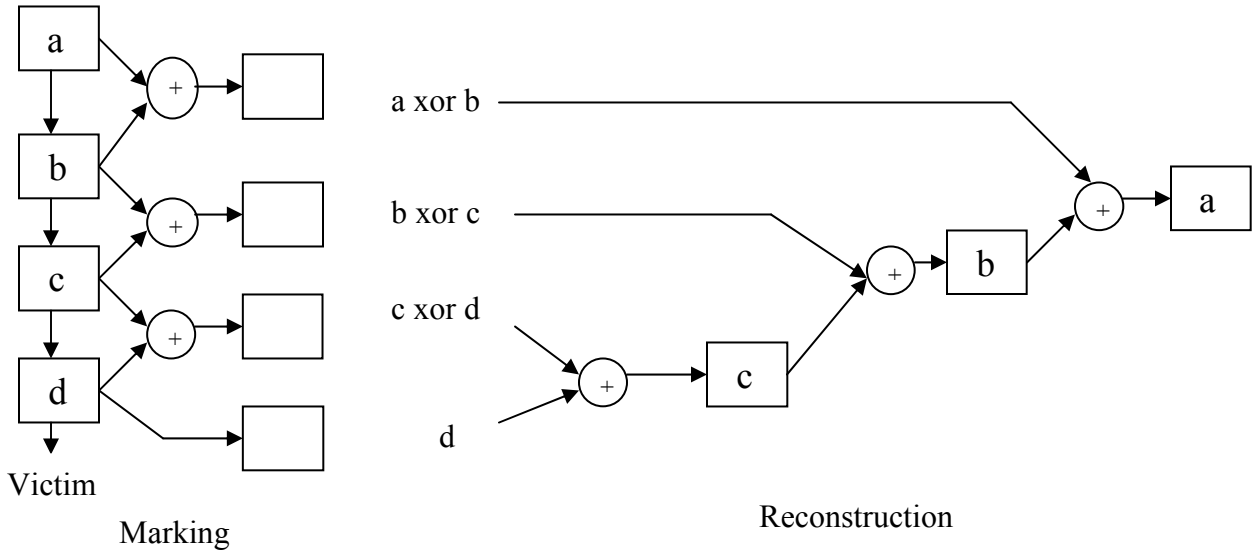


Figure 5.3 XOR Encoding [41]

The marking and reconstruction procedure shown in Figure 5.3 are described as follows: Every ASBR's marks the outgoing packet, *P*, with a probability p when forwarding the packet. If an ASBR decides to mark a packet, it encodes its ASN with a hashing algorithm $h^i\left(AS_j\right)$, to mark the edge field, *E*. The distance field is set to 0. The neighboring ASBR overwrites the edge field, *E*, with $h^i\left(AS_j\right) \oplus h^i\left(AS_{j+1}\right)$ and increments D_{AS}. If an ASBR doesn't intend to mark,

D_{AS} is incremented and other fields are left unscathed. The edge field received at the victim will always have xor of two neighboring routers, except for samples from the ASBRs one hop from the victim. Figure 5.4 describes the marking procedure at ASBR AS_j

```
for each packet P,
  let u be a random number from [0, 1)
   if (u<=p) then
      let i be a random integer from [0, 8)
         P.Distance = 0
         P.Hid = i
         P.Edge = h^i(AS_j)
   else
     If (P.Distance = 0) then
       P. Edge = P. Edge xor P. h^i(AS_j)
       P. Distance = P. Distance + 1
```

Figure 5.4 Marking Procedure at ASBR AS_j

5.2 AS Reconstruction

After enough packets have been sent, the victim would have received at least one marking from every router. The victim reconstructs the ordered paths using the upstream AS topology *G*. The victim arranges the edge markings of packets based on the distance field, D_{AS}. We denote edge markings with i^{th} hash id at a distance *d* as $E^i(d)$.

For every $E^i(0)$, the victim, *v*, compares the markings with the hash of its neighbors, $h^i(neighbors(v))$ from *G*. If the two values match, then the neighbor is considered to be in the reconstructed graph, R_0. R_0 are the set of ASBRs, 0 hops away from the victim. For each ASBR, *x*, in R_d and for each edge marking, $E^i(d+1)$, the victim computes $z = h^i(x) \oplus E^i(d+1)$ and compares with the $h^i(neighbors(x))$ from *G*. If the victim finds a match then the neighbor is placed in the reconstructed graph R_{d+1}. This process is continued recursively until there are no

edge markings at a distance, *d.*

```
let R_d be empty for 0 ≤d≤ maxd
for each neighbor n of Victim v in G
 let count = 0
  for l=0 to 2^w -1
    if (h^i(n) = E^i(0)) then
      count = count +1
    if count > m then
      insert n into R_0
for d=0 to maxd - 1
 for each x in R_d
   for each neighbor u of x in G
     let count = 0
      for l=0 to 2^w-1
        for each edge E^i (d+1)
          z=h^i(x) xor E^i (d+1)
           if (h^i(u) = z) then
             count = count +1; break
            if count > m then
              insert u in R_d+1
Output Rd for 0 ≤d≤ maxd
```

Figure 5.5 Reconstruction Procedure at Victim v

5.3 Intra-AS Marking

HIT reserves two fields for intra AS marking, an eight bit node field and a single bit distance field. At intra AS level, normal routers in the attack originating AS are involved in marking. These routers employ node sampling scheme and mark the attack packets with a probability q. The marking routers embed the node field with the hash of their IP addresses, $h(IP)$, and the distance field with hop count obtained from the TTL field in the IP header. These routers set the 5 least-significant bits of the packet's TTL field ($TTL_{[5..0]}$) to a global constant *c*, and store the sixth bit of the TTL in the distance field d. This mechanism is the same as that used in FIT [30]. When a packet arrives at its destination, the distance at which the packet was marked

is computed as: $d = \left(d|c - TTL_{[5.0]}\right) \bmod 64$, where $d|c$ denotes concatenation of the one bit distance field d with the five bit TTL replacement constant *c*. Because legacy routers decrement the TTL, the HIT distance is representative of the exact number of hops from a marking router, rather than just the number of hops of traceback enabled routers [30]. Figure 5.6 describes the marking procedure at router IP_j

```
for each packet P,
 let u be a random number from [0,1)
  if (u<=q) or (P. d|c - TTL[5..0]) mod 64> 32) then
    P.Node = h(IPj)
    P.Distance = TTL[5]
      TTL[4..0] = c
else
      TTL = TTL -1
```

Figure 5.6 Marking Procedure at Router IP_j

5.4 Reconstruction

Once the victim identifies the attack originating AS it gives a notification message to the corresponding AS administrator regarding compromised nodes launching attack traffic within its domain. The source AS then monitors the packets within its domain. The attack graph inside the AS is reconstructed based on the hashed IP address of the marking routers and their respective intra AS hop counts. Since node sampling is employed within the source AS, the markings that are received are the hash of the IP address of the marking nodes. For a particular hop count, the markings are compared with the hash values obtained from the upstream router topology information. The neighbor graph information could be obtained through routing protocols such as RIP and OSPF. Similar to the inter-AS level, the number of false positives and false negatives are obtained by comparing the reconstructed attack graph and the original attack graph at each intra AS hop count. Figure 5.7 describes the reconstruction procedure at victim *v*.

```
Let S_d be empty for 0 ≤d≤ maxd
for d:= 0 to maxd - 1
  for each marking n at d
    for each child u of y at d in G
      if( n = h(u))
        insert u in R_d

Output Rd for 0 ≤d≤ maxd
```

Figure 5.7 Reconstruction Procedure at Victim v

5.5 Performance Evaluation

This section presents the general performance metrics considered to evaluate a traceback mechanism. It discusses two metrics that have been used previously and details on a new metric: *Total Time to Traceback*, proposed in this section. In the section, we present the simulation results for the discussed metrics.

5.5.1 Performance Metrics

Three performance metrics, number of false positives (FP), number of packets required to reconstruct the attack graph (Pkts) and total time to traceback to the attacker (TTT) are considered for evaluating the performance of HIT.

- **Number of False Positives at Inter AS Level (FP)**

  We evaluate the performance of HIT with the number of false positives the map reconstruction algorithm can produce. A false positive is said to occur when two ASBRs or IP addresses share a common subset of hash values which are received by the reconstructing victim. The victim will not be able to differentiate between the two ASBRs and will thus add both of them to the reconstructed map. It is desirable to keep the false positive rate low even in presence of several hundreds of attackers. The number of false positives generated during reconstruction of the attack graph is drastically reduced if a number of hash functions are

employed in the scheme. The intuition is that the probability for any AS to have the same hash value as another for one hash function is $1/2^x$, where x is the number of bits used to encode the ASN information. Therefore the probability that the same can happen for y independent hashing functions is $(1/2^x)^y$. We use an m-threshold scheme, wherein a node u in G_m will only be considered to be in the attack path if more than m of its hash values from 2^w hash functions match the right markings in the attack packets, where w is the number of bits allocated for hash id field.

Let T_y denote the in-degree of element y in R_{d-1}, the reconstructed graph at a distance d-1 and $|E_d|$ be the number of unique edge segments received by the victim at a distance d. Because the hash value is 11 bits, the probability of a false positive for m threshold scheme is given by

$$\left(1/2^{11}\right)^m \qquad [5.1]$$

The expected number of false positives among y's children for m- threshold scheme is given by

$$T_y * \Pi_{1\le l\le 2^w} |E_d|/2^{11-w} \qquad [5.2]$$

The total number of expected false positives in y's children for all y in the sets $\{R_d\}_{0\le d\le maxd}$ is

$$\sum_{y=0}^{|R_d-1|} T_y * \Pi_{1\le l\le 2^w} |E_d|/2^{11-w} \qquad [5.3]$$

Computational Complexity of the algorithm is

$$O(\sum |R_d| * |E_{d+1}|) \qquad [5.4]$$

- **Number of False Positives at Intra AS Level (FP)**

  The average size of an AS ranges from 250 to 1000 nodes with an average diameter of 10. Therefore the hash of IP address (32 bits) to 8 bits will result in nearly zero false positives. Theoretically, the average number of false positives is given by

  $$Number\ of\ Nodes\ at\ a\ hop\ count\ /\ 2^8 \qquad [5.5]$$

- **Number of Packets Required to Reconstruct (Pkt)**

  The second performance metric for map reconstruction is the number of packets that must be sent to enable a victim to reconstruct the IP addresses of the routers on a single path. This number calculated below provides an upper bound on the number of packets needed to reconstruct a map containing multiple paths.

  Let p be the marking probability of an ASBR for a path with d routers. The probability that a given packet will deliver a sample from some router is at least

  $$dp(1-p)^{d-1} \qquad [5.6]$$

  The number of packets required for a victim to reconstruct a path of length d has an upper bound of

  $$\ln d \Big/ p(1-p)^{d-1} \qquad [5.7]$$

  There are a very few paths in the internet topology which exceed a path length 25 [40]. With p=1/25 and d=25, number of packets required to reconstruct a path of 25 is 215. A packet traverses through less than 6 ASBR on an average. With p=1/7 and d=7, packets required are 35.

- **Total Time to Traceback (TTT)**

  While number of packets needed to reconstruct is an important criteria, we believe that the Total Time to Traceback is much more crucial. This is the time taken for the victim to

construct the attack graph and to trace the attacker, once the victim detects that an attack has been launched. Thus,

$$T = T_{Pkt} + T_R \quad [5.8]$$

where T_{Pkt} is time taken to receive enough number of packets and T_R is the reconstruction time. Thus, the network would be under attack for a duration TTT (units). T_R is traceback mechanism dependant. There are two related consequences to a flooding attack – the network load induced and the impact on the victim's CPU. To load the network, an attacker generally sends small packets as rapidly as possible since most network devices (both routers and NICs) are limited not by bandwidth but by packet processing rate. Thus, T_{Pkt} is small. Thus, to reduce load on the network and curb the attack fast, the traceback mechanism needs to be efficient, so that T_R will be small.

5.5.2 Simulation Results

- **AS Traceback**

As more than 99.5% of paths encountered in the Internet have an AS path length less than or equal to six, we propose to set the marking probability to 1/6.

Figure 5.8 shows the number of packets needed to reconstruct the AS attack graph for different schemes. For m>2/4 scheme, after as few as 30 packets are received from each AS path, a victim can construct more than 98% of ASes in the attack graph. For m>6/8 scheme the number is slightly higher. The victim needs around 60 packets to reconstruct around 95% of the routers.

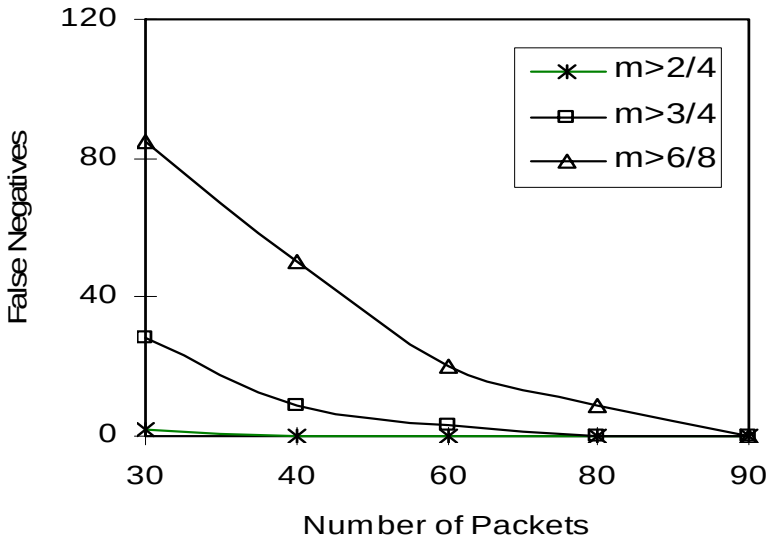


Figure 5.8 Number of Packets vs False Negatives

The performance of various schemes in terms of false positives is shown in Figure 5.9. As predicted, the m>6/8 scheme performs the best with just around 30 false positives while m>2/4 scheme gives more number of false positives, around 200 in the presence of 500 attacking ASes.

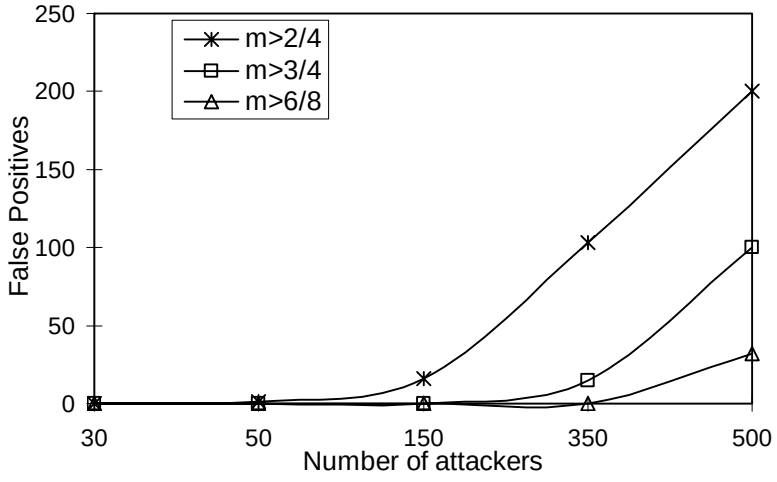


Figure 5.9 Number of Attackers vs False Positives

Figure 5.10 shows the time required to reconstruct the attack graph. The m>6/8 scheme takes the maximum time because of the presence of a large number of combinations. Both m>3/4 and m>2/4 take similar time durations to reconstruct the attack graph. It is observed that even in the worst case, the reconstruction of attack graph takes less than 10 seconds.

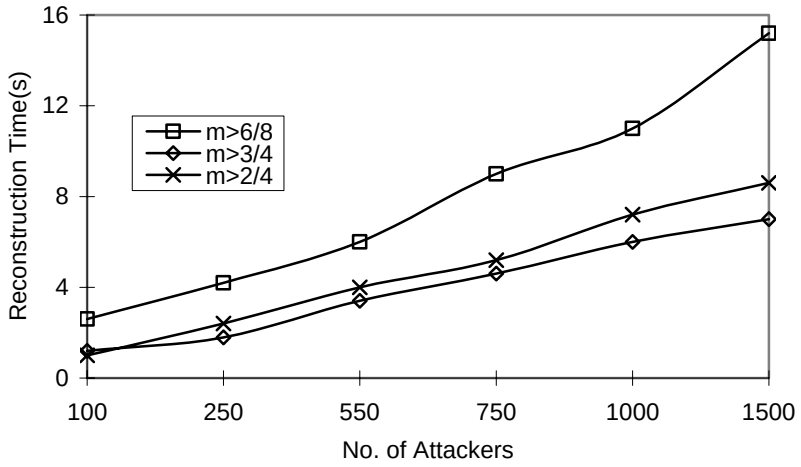


Figure 5.10 Number of Attackers vs Reconstruction time

- **IP Traceback**

This section presents the simulation results for reconstruction mechanism of the IP path within the originating AS. IP marking is based on node sampling rather than edge sampling. Figure 5.11 shows the number of false positives generated for different sized ASes. For an AS size of 1500 nodes there are approximately 3 false positives for a scheme without a distance field while there are negligible false positives generated for a scheme with the distance field.

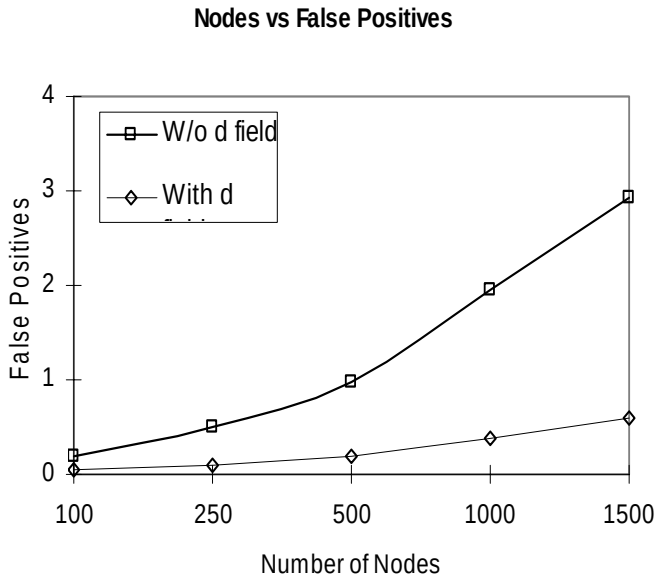


Figure 5.11 Number of Nodes vs False Positives

- **Comparison with AMS.**

  In this section, we compare the performance of HIT with AMS. AMS attempts to reconstruct the whole IP attack path, while HIT first reconstructs the AS path and then reconstructs the IP path within the attack originating AS. To make an accurate comparison between HIT and AMS, we ran AMS on the same data set as HIT.

  Figure 5.12 shows the number of packets required for a victim to traceback the attack originating ASes. While HIT reconstructs to the attacker with less than 100 packets (for both AS path reconstruction and originating IP location within the AS), AMS requires more than 2000 to effectively reconstruct the attack graph

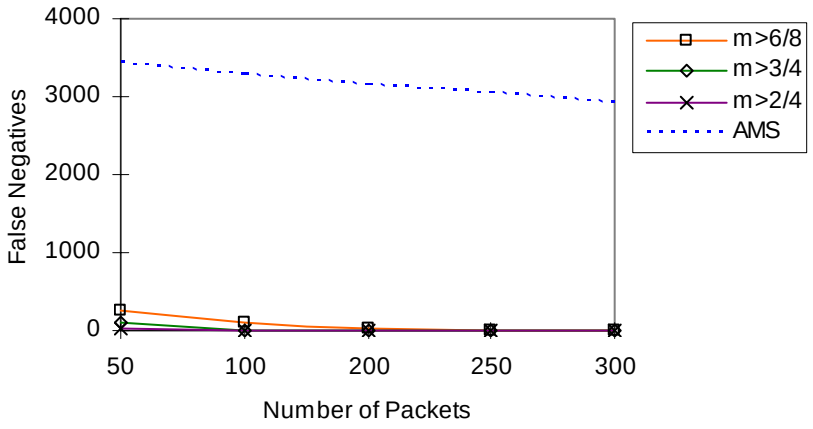


Figure 5.12 Number of Packets vs False Negatives

Figure 5.13 compares the performance of HIT and AMS in terms of false positives. HIT (m > 2/4) gives more false positives than AMS (m>6/8). But, performance of HIT (m > 3/4) and HIT (m > 6/8) are comparable to that of AMS (m>6/8). In the case of scheme, HIT, the false positive rate is zero in the IP-traceback stage therefore we considered the false positives at the AS level to be the total number of false positives for the scheme.

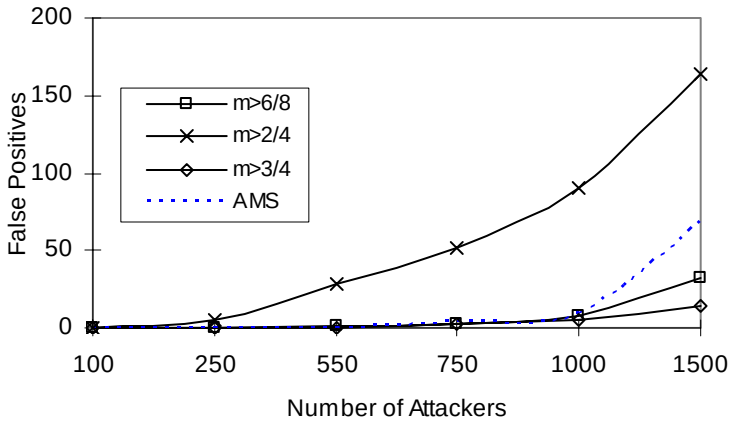


Figure 5.13 Number of Attackers vs False Positives

Figure 5.14 compares the reconstruction time for various threshold values of HIT with m>6/8, threshold for AMS. The performance gain with HIT is obvious and this is a result of dealing with smaller topologies first at AS-level and just the originating AS topology for the IP traceback phase.

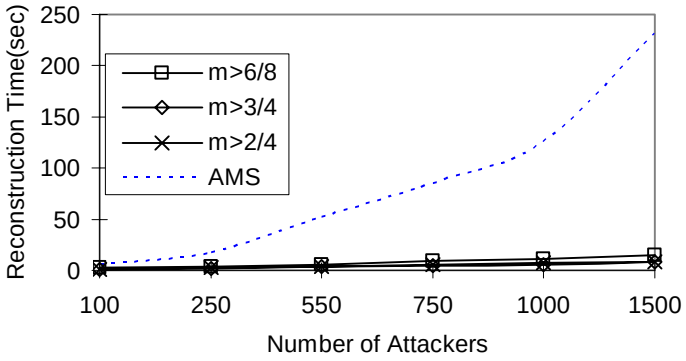


Figure 5.14 Number of Attackers vs Reconstruction Time

To compare the overall performance of different traceback schemes, we use a new metric – Total Time to Traceback – as defined in Section 5.5.1. Figure 5.15 compares the TTT for both AMS and HIT. For this purpose, we assumed that each attacker sends packets at a rate of 10 pkts/sec. Even in case of 1500 attackers, HIT (m>3/4) is able to traceback the attackers within 25 seconds while AMS needs around 275 seconds.

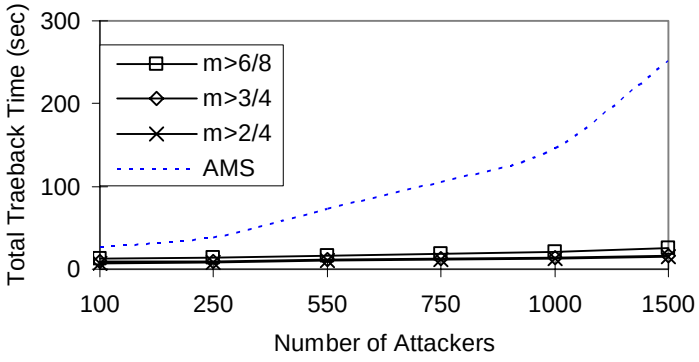


Figure 5.15 Number of Attackers vs Total Traceback Time (sec)

CHAPTER 6: FAST: FAST AUTONOMOUS SYSTEM TRACEBACK

This chapter describes, FAST (Fast Autonomous System Traceback), a novel packet marking approach in which an AS path fingerprint is embedded in each packet, enabling a victim to identify packets traversing the same AS paths through the Internet on a per packet basis, regardless of the source IP address spoofing [2]. FAST features many unique properties. It is a per-packet deterministic mechanism: each packet traveling along the same AS path can carry only any of the pre-determined identifiers. This allows the victim to identify the source AS of the packet in real-time thus prompting pro-active actions to curb the attack.

This approach embeds in each packet an identifier, based on the router path that a packet traverses. The victim needs to receive only a few packets from a given AS path to reconstruct the path. What makes this possible is that our packet marking is deterministic – all packets traversing the same path carry one of the subset of markings. All previous marking schemes that we are aware of are probabilistic in nature, in which the victim needs to collect a large number of packets to reconstruct the path. In our approach, the victim can traceback in real-time to the originating AS upon receiving just over 8 to 10 packets. FAST is extremely light-weight, both on the routers for marking, and on the victims for decoding and filtering. Above all, FAST employs only AS border routers for marking. The advantages of ASBRs being involved in marking has been discussed in the previous chapter.

6.1 Node Append

The simplest marking algorithm – conceptually similar to the IP Record Route option [40] – is to append each node's address to the end of the packet as it travels through the network from attacker to victim. Consequently, every packet received by the victim arrives with a complete ordered list of the routers it traverses – a built-in attack path. The node append

algorithm is both robust and extremely quick to converge (a single packet), however it has several serious limitations when normal routers are involved in marking. Principal among these is the unfeasibly high router overhead incurred by appending data to packets in flight [40]. This problem arises only when normal routers with lower processing speeds and memory are involved in marking.

6.2 Marking

FAST employs ASBRs in node append mode of marking. An ASBR marks a packet with its ASN (Autonomous System Number) only if the packet is forwarded to a router belonging to another AS [34]. Thus, a packet may get marked only when it exits an AS. This scheme reserves 25 bits of the IP header for 3 fields of marking. The 25 bits of the IP header are overloaded as follows

Table 6.1 Encoding Node Field into the IP header

Node Append - N	Hop	Hid
20 bits	3 bits	2 bits

In FAST each ASBR encodes its ASN, $h^i(AS)$ with hashing algorithm id, Hid, to mark the Node field in a packet P. The hashing algorithm compresses the ASN from 16 bits to 4 bits. Therefore each ASBR appends its 4 bit hashed ASN in the Node field. Since 99.7% of AS paths are of length less than 6 this scheme reserves 20(4*5) bits for the Node field. The ASBR of the AS containing the victim is not involved in marking hence on an average 5 ASBRs are involved in marking in FAST. The Hop field is incremented by 3 every time a router marks the packet. This field serves as an offset for the neighboring ASBR to mark in the Node field. Figure 6.2 shows the FAST packet marking, for packets traversing from the attacker A1 to attack

originating AS through ASBRs AS1, AS4 and AS5. Figure 6.3 describes the marking procedure at ASBR AS_j.

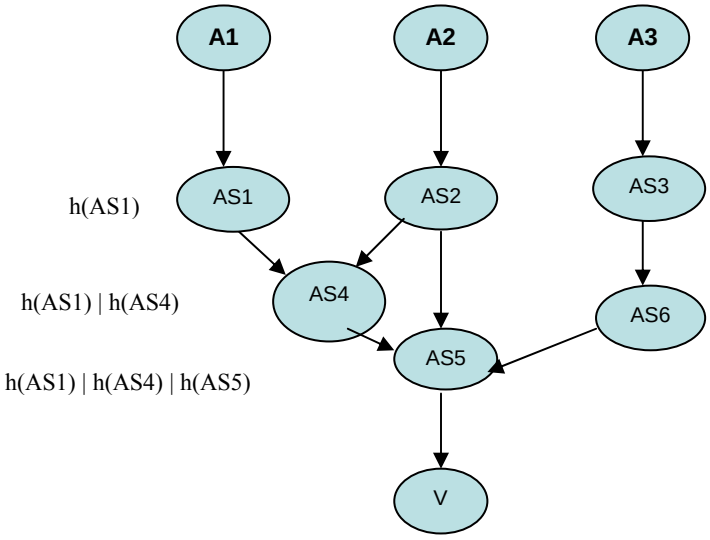


Figure 6.1 FAST Packet Marking

```
P.Hop = 0
for each packet P,
  let i be a random integer from [0,4)
    P.Node = h^i (AS_j)
    P.Hop  = P.Hop + 3
    P.hid  =  i
```

Figure 6.2 Marking Procedure at ASBR AS_j

6.3 Reconstruction

Once the attacker receives around 8-10 packets from each path it can begin the process of reconstruction. The victim stores the upstream router map from which it can obtain sets of paths of various path lengths. The victim compares the markings of a particular path length with hash of ASes in suspect attack paths of the same path length. If the count of paths identified is greater than or equal to a threshold *m*, then the paths are considered as attack paths.

```
For all markings received at v

If(count(Nodes in suspected attack path) = count (Markings in a packet))
   For marking x at hop i in the packet & For node n at hop i in suspected attack path
      If (x = hash(n)) for all i then
         Push(path) into array a

   If count(path) in a > m then
      Push(path) into array attack path
```

Figure 6.3 Reconstruction Procedure at Victim v

6.4 Performance Evaluation

Path reconstruction is the most critical performance aspect of a traceback scheme. Three performance metrics, number of false positives (FP), number of packets required to reconstruct the attack graph (Pkts) and reconstruction time (RT) are considered for evaluating the performance of FAST.

6.4.1 Performance Metrics

- **Number of False Positives at Inter AS Level (FP)**

  We evaluate the performance of FAST with the number of false positives the map reconstruction algorithm can produce. A false positive is said to occur when two ASBRs share a common subset of hash values which are received by the reconstructing victim. The victim will not be able to differentiate between the two ASBRs and will thus add both of them to the reconstructed map. It is desirable to keep the false positive rate low even in presence of several hundreds of attackers.

  Theoretically, the number of false positives generated for A number of attackers at path length P is given by

$$\left[1-\left(1-1/2^{P*h}\right)^{|A|}\right]*N \qquad [6.1]$$

where h is the number of bits to which an ASN is hashed and N is the numbers of suspected attack originating ASes at path length P. The traceroute datasets obtained from CAIDA had 1564 unique ASN at hop count 4. These constants when substituted in the theoretical formula for 150 attackers and 4 bits ASN hash gives negligible false positives.

- **Number of Packets Required to Reconstruct (Pkt)**

  The second performance metric for map reconstruction is the number of packets that must be sent to enable a victim to reconstruct the IP addresses of the routers on a single path. Since this scheme utilizes a deterministic marking approach the number of packets to identify the attack paths is 8-10.

6.4.2 Simulation Results

We considered the AS topology and the distribution of attackers on the Internet according to the graph in figure 4.2. Simulations for the number of false positives generated for various path lengths and attackers were conducted. Also, simulations of the reconstruction time for a varied number of attackers were performed.

Figure 6.4 gives the relation between the AS path lengths and the number of false positives for two schemes. The scheme with L=4 bits has fewer false positives as compared to the scheme with L=3 bits for various path lengths. In both the schemes the number of false positives reduce with increase in AS path length. In the scheme with L=4 bits the false positives are zero at AS path length 4, while for the other scheme the false positives reduce to zero at 6 AS hop counts. This graph justifies the theoretical formula, an increase in AS path length causes a decrease in the number of false positives.

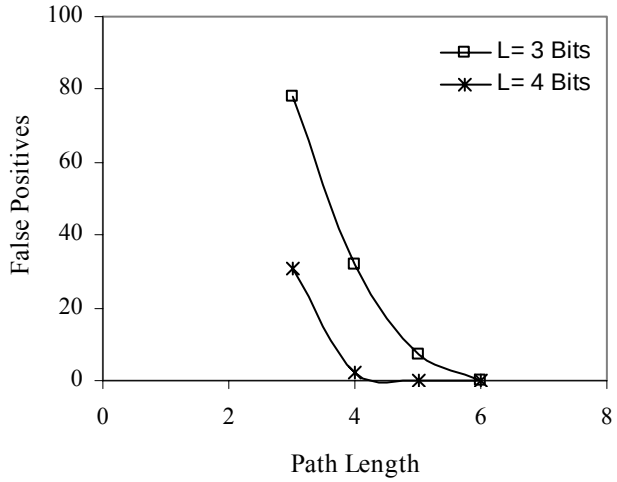


Figure 6.4 AS Path Length vs Number of False Positives

Figure 6.5 gives a relation between the number of attackers and the number of false positives for two schemes. The attack originating ASes were selected randomly in sets ranging from 100 to 1000 ASes from the dataset obtained from CAIDA. The scheme with L=4 bits has fewer false positives compared to the scheme with L=3 bits for various attackers. The former scheme has a larger sample space as the 16 bit ASN is hashed to 4 bits causing fewer collisions as compared to the latter scheme where the ASN is hashed to 3 bits.

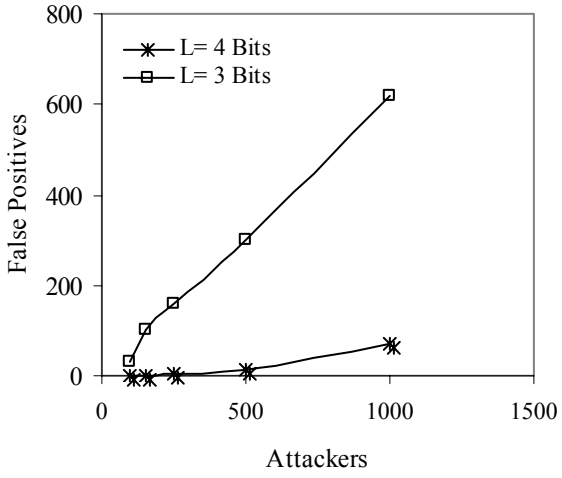


Figure 6.5 Number of Attackers vs Number of False Positives

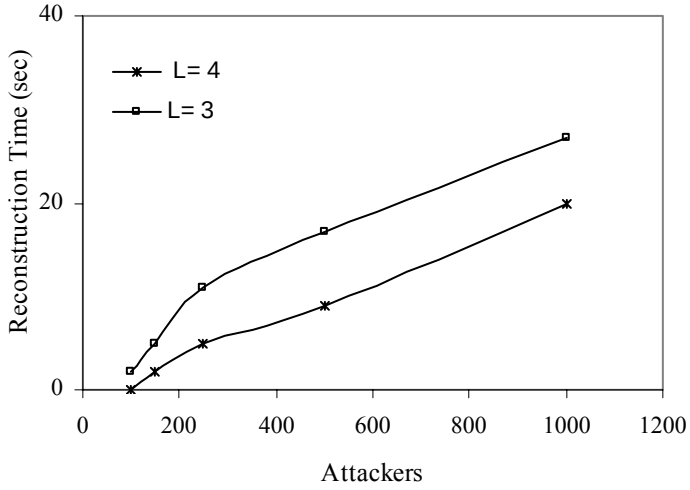


Figure 6.6 Number of Attackers vs Reconstruction Time (sec)

Figure 6.6 shows the relationship between the number of attackers and reconstruction time. Scheme with L=4 bits takes lesser time to reconstruct to the attacker in comparison to the scheme with L=3 bits for different sets of attackers. For both the schemes the reconstruction time is in tens of seconds. With 1000 randomly picked attackers launching DDoS attacks, FAST only takes 20sec to reconstruct for the scheme with L=4 bits.

CHAPTER 7: STM: SIMPLE TRACEBACK MECHANISM

Simple Traceback Mechanism is a simple scheme to trace back to the attackers launching DDoS attacks. Like HIT, this scheme also traces back to the attacker in two phases. In the first phase the victim traces back to the attack originating ASes and in the second phase the attackers within these domains are identified.

7.1 Packet Marking:

In STM, as in all other PPM schemes, routers overwrite some fields in the IPv4 header. The IP identification field of the IP header is overloaded as follows when the ASN is marked: 12 bits for the hash of ASN, 3 bits for the hash id and 1 bit for the flag field. The same field is overloaded as follows when the IP address is encoded and used for marking the packet: 11 bits for hash of IP address, 3 bits for hash id, 1 bit for distance and 1 bit for the flag id to identify if the marking in the packet is the hash of ASN or IP address.

Table 7.1 Encoding Node Field into IP header

Node- h(ASN)/h(IP)	Hid	Flag	Distance
12 bits	2 bits	1 bit	1 bit

The STM router within an AS marks the forwarded packet with a probability of p or q, which are global constants for all the STM routers (set to .01 and .04 in our experiments). Three fields are marked by an STM router, the 12 bit ASN node field, 2 bit hash id field 1 bit flag field and the 1 bit distance field when a packet is marked with probability p. Each router calculates a hash the ASN it belongs to and sets the flag to zero. A router marking a packet with probability q, encodes the IP node, distance and flag field. The marking router calculates the hash of its IP address, sets the flag to one and sets the 5 least significant bits of the packet's TTL to a global

constant c and stores the 6th bit of the TTL in the distance field [30]. This allows the victim to determine the distance since the router's mark.

```
for each packet P,
 let u be a random number from [0,1)
 let I be a random number from [0,8)
  if (p<u<=q) OR (P. f_ID|c - TTL_[5..0]) mod 64> 32) then
    P.Node = h(IP_j)
    P.Distance = TTL_[5]
    TTL_[4..0] = c
    P.ASN=0
    P.Flag=1
   P.Hid=0
else
      TTL = TTL – 1
If (u<=p)
  P.ASN = h(ASN_j)
  P.Hid=i
  P.Flag=0
```

Figure 7.1 Marking Procedure at Router IP_j

7.2 Path Reconstruction

The victim receives packets with ASN, IP, flag and distance fields. Upon receiving enough packets the victim utilizes the upstream ASBR map to identify the attack originating AS. All the ASBRs in the upstream map are scanned to match markings in the ASN field of the packets with flag id set to zero. If there is a match these ASBRs are placed in the reconstructed map. The victim receives markings of the nodes in the attack originating AS, from the $h(IP)$ and distance fields in the packets. The distance at which the packet was marked is computed as: $d = (d|c - TTL_{[5.0]}) \bmod 64$, where $d|c$ denotes concatenation of the one bit distance field d with the five bit TTL replacement constant *c* [30]. The victim scans through the space of all

possible IP addresses at a particular distance to match the markings from the packets. If a match exists it is placed in the reconstructed attack graph.

7.3 Performance Metrics

Path reconstruction is the most critical performance aspect of a traceback scheme. Performance is measured in terms of false positives, reconstruction time and number of packets to traceback to the attacker.

- **False Positives**

  At the AS level the false positives are determined by the size of the AS topology and the number of hashing algorithms employed for encoding.

$$\textit{Size of the AS topology} / (2^{12})^{8} \quad [7.1]$$

  Within an AS the number of false positives is determined by the count of normal routers at a distance, d, from the victim.

$$\textit{Routers within an AS at a distance, d, from victim} /(2^{11})^{8} \quad [7.2]$$

- **Number of Packets to Reconstruct the Attack Path**

  At both levels, node sampling is the mode of marking. The probability of receiving a packet from a node i hops from the ASBR, given marking probability p is

$$P_m[i, p] = p(1-p)^{i-1} \quad [7.3]$$

  We assume that samples from all the routers appear with the same likelihood as the furthest router. The upper bound for the required number of packets for victim to reconstruct the path is given by

$$\ln d \Big/ p(1-p)^{i-1} \quad [7.4]$$

The experimental results strictly followed the theoretical analysis. 30 packets were required to reconstruct a path to the attack originating AS. Since node sampling is the mode of marking, SMT can identify a router far away before identifying all the routers downstream. SMT could be at its best as a two packet mechanism, the first to reach the AS and the second to reach the attacker.

- **Reconstruction Time**

  This parameter is dependent on the number of attackers, topology, and number of packets required to identify the attackers. The total time for reconstruction is the time taken to identify the attack originating AS and the attackers within an AS. A modern workstation can calculate the SHA-1 hash of all 2^{32} IP address in approximately 30 minutes [30]. There are around 2^{13} ASN in the internet today therefore on an average time taken to reconstruct to an AS and the attacker within an AS will be less than a second. The experimental results tallied with the theoretical analysis.

7.4 STM Advantages & Disadvantages

Since STM adopts 1 bit distance field of FIT [30] and therefore has the advantage of freeing 4 additional bits in the IP header, which can be used for larger hash hence reducing the number of false positives. STM also uses node sampling which reduces computational overhead over AMS and other PPM schemes. STM uses the hash of an IP address in contrast to FIT which requires the comparisons of fragments of the hashed IP address. This offers a significant advantage over FIT which suffers from combinatorial explosion of computations. STM has a better edge in reconstruction time over FIT as the number of nodes scanned to identify an attacker is far less.

The main disadvantage of STM arises due the node sampling mode of marking which makes this scheme susceptible to pollution attacks, the attacker sends malicious fragments that interfere with path reconstruction.

CHAPTER 8: SIMULATIONS

This chapter gives an overview on the background work done for simulations. All the simulations for generating an AS topology and for implementing the proposed algorithms were developed using Perl. The simulator has been modularized, so as to enable the user to choose an AS topology or Internet topology of any size. These topologies form the test bed to implement existing and new algorithms. The Internet topology with IP paths was obtained from Route Views, a project which was originally conceived as a tool for Internet operators to obtain real-time information about the global routing system from the perspectives of several different backbones and locations around the Internet [42]. Data from Route Views is used by organizations to map IP addresses to origin ASes for various topological studies. Route Views collect Cisco BGP RIBs which are in the format of 'network/ASN'.

```
3.0.0.0/8 80
4.0.0.0/8 3356
4.17.225.0/24 6496
4.17.226.0/23 6496
4.17.251.0/24 6496
4.17.252.0/23 6496
4.21.252.0/23 19198
```

Figure 8.1 BGP Table

Routing information in this file indicates that the network 3.0.0.0/8 belongs to the AS80, 4.0.0.0/8 to 3356 and so on. In order to map IP addresses from a path to their corresponding ASN, the range of the IP addresses which belongs to an ASN was determined using the concepts of subnets.

Addressable Range		ASN
3.0.0.1	3.255.255.254	80
4.0.0.1	4.255.255.254	3356
4.17.225.1	4.17.225.254	6496
4.17.226.1	4.17.227.254	6496
4.17.251.1	4.17.251.254	6496
4.17.252.1	4.17.253.254	6496
4.21.252.1	4.21.253.254	19198

Figure 8.2 Address Range for ASN

A real traceroute dataset was obtained from the Cooperative Association for Internet Data Analysis (CAIDA). The traceroute dataset contained complete and incomplete distinct traceroutes from a single source to multiple destinations.

```
T 128.8.7.4 206.27.119.251 0 0 1086048002 R 48.602 16 240 S 0
C 128.8.7.28,0.505,1 128.8.0.9,0.324,1 128.8.0.82,0.475,1 128.8.0.234,0.287,1
206.196.177.49,0.317,1 206.196.178.45,0.830,1 65.114.173.1,0.654,1 205.171.9.61,0.656,1
205.171.8.182,4.914,1 205.171.17.33,4.911,1 205.171.8.229,25.315,1 205.171.20.170,25.088,1
205.171.20.66,24.958,1 63.145.140.150,104.469,1 63.148.10.2,51.061,1
T 128.8.7.4 203.80.119.251 0 0 1086048002 N 0 0 0 G 5
I 128.8.7.28,0.744,1 128.8.0.9,0.366,1 128.8.0.82,0.694,1 128.8.0.234,0.272,1
206.196.177.49,0.374,1 206.196.178.45,0.989,1 65.114.173.1,0.684,1 205.171.9.93,0.713,1
205.171.209.114,1.877,1 205.171.251.38,2.118,1 192.205.32.29,2.860,1 12.123.9.82,3.510,1
12.122.10.30,25.957,1 12.122.9.142,26.560,1 12.122.10.14,64.354,1 12.123.199.113,63.474,1
```

Figure 8.3 Traceroute Dataset

Complete paths were identified and used for simulations. Each of the IP addresses from the complete traceroute paths was then mapped to the network it belonged to and the corresponding ASN was taken into an AS file which was used in forming the AS topology. There were IP addresses for which there were no corresponding legitimate AS and therefore it was not possible to fully convert every trace route path to an AS path. Some IP addresses could not be mapped to an AS because they lacked a matching prefix in RouteViews [43]. Other

addresses could not be mapped because they fell within reserved address space. In our conversion from IP paths to AS paths we replaced such addresses with their adjacent ASes. The AS file was given as input to another Perl tool to generate the AS topology.

```
14742 16097 21651 6364
26793 3549
209 4628 27 7527 21687 6922 600 14751 8039 17819 81 2901 7462 16871 14861 14517 104
```

Figure 8.4 AS Topology

The first line in the above file indicates that AS14742 has children AS16097, AS21651 and AS6364. Similarly the IP topology is also generated.

All the three algorithms proposed were tested on this topology. The outputs for all the algorithms are as follows:

HIT

```
For 25
--------
FP 4.8
FN 0
RT 0

For 50
--------
FP 12.8
FN 0
RT 0

For 75
--------
FP 18.8
FN 0
RT 0
```

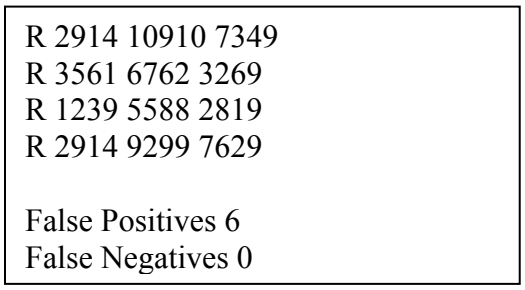
For 25 attack paths,
Marking probability: .25,
Packets sent through each path: 150.
HIT with 1 hashing algorithm gives the following results

False Positives: 4.8
False Negatives: 0
Reconstruction time: 0.

Similar results were obtained for implementing HIT and AMS with hashing algorithms.

Figure 8.5 HIT Results

FAST



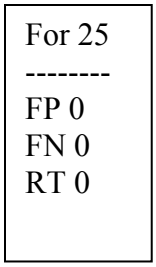
For 50 attack paths, m>3/4.

The reconstructed attack paths are 2914 10919 7349, 3561 6762 3269 and so on.

False Positives 6
False Negatives 0

Figure 8.6 FAST Results

STM



For 25 attack paths,
Marking probability: .25,
Packets sent through each path 30

False Positives: 0
False Negatives: 0
Reconstruction time: 0

Figure 8.7 STM Results

CHAPTER 9: CONCLUSIONS AND FUTURE DIRECTIONS

This thesis presented three trace back schemes against Distributed Denial of Service attacks. All the three schemes were implemented on an AS and IP topology. The schemes aimed at providing a realistic solution to the problem which arises due to the fact that the attacker's packets may traverse through a number of domains under different administrative controls. Most of the methods are very limited because for those schemes it is necessary to have access to all routers along the path from victim to attacker, and this is often not the case. The proposed schemes give control to the Autonomous system administrator to monitor its traffic and do the required to halt the DDoS attack.

The Internet Hierarchical IP Traceback mechanism was developed to provide experimental results for evaluating its performance metrics in comparison with those of Authenticated Marking Scheme. Theoretically and experimentally HIT proved to be a better scheme because of its inherent nature of running on the AS topology. Reconstruction time was one of the main parameters which evaluated the performance of this scheme. Even, in the worst case the reconstruction time for HIT was in the order of 10s of seconds. This is a very important metric because of the tremendous growth in automated and distributed nature of the DDoS attacks which requires the victim to reconstruct to the attacker within seconds.

The performance of FAST was evaluated on a dataset of varying hop length. With the internet topology distributed such that most of the AS paths have path lengths of four and five, FAST performs very efficiently at these path lengths. FAST also requires a very few packets to reconstruct to the attack originating AS.

Simple Traceback Mechanism was developed to improve the performance of Fast Internet Traceback Scheme. It is an efficient scheme as the victim reconstructs to the attacker in

no time and with a very few packets. But this scheme has its own drawbacks as it is susceptible to pollution attacks. Encryption and authentication schemes have to be employed with SMT to handle its limitations.

Future work can incorporate schemes like algebraic approach to traceback on an AS topology. A mix of reactive and preventive schemes (filtering) can be implemented against DDoS attacks.

REFERENCES

[1] Mary Landesman. What is a DDoS attack, http://antivirus. about. com/od/what is a virus/a/ddosattacks.htm, December 2004.

[2] Abraham Yaar, Adrian Perrig and Dawn Song. Pi: A Path Identification Mechanism to Defend against DDoS Attack, IEEE Symposium on Security and Privacy, Pg 93-107, May 2003.

[3] Drew Dean, Matt Franklin, and Adam Stubblefield. An Algebraic Approach to IP traceback, ACM Transactions on Information and System Security, Volume 5, pg 119-137, May 2002.

[4] Rik Farrow. DDoS is neither dead nor forgotten, http://www.spirit.com/Network/net1200.html, September 2000.

[5] http://www.cnn.com/2005/TECH/internet/03/22/university.hackers.ap/index.html, March 2005.

[6] Jelena Mirkovic, Sven Dietrich, David Dittrich, Peter Reiher and Radia Perlman, "*Intenet Denial of Sevice Attack & Defense Mechanism*s", First edition, Prentice Hall Professional Technical Reference, USA, Dec 2004.

[7] Distributed Denial of Service Attacks and their defenses, http://www.lancs.ac.uk/postgrad/pissias/netsec/ddos/.

[8] Worm Propagation, http://ee.tamu.edu/~reddy/ee689_04/lec7.pdf

[9] Internet Security Systems. SYN flood, http://www.iss.net/security_center/advice/Exploits/TCP/SYN_flood/default.htm

[10] Norman. Attacks against weaknesses in the TCP/IP protocol, http://www.norman.com/documents/wp_smurf.shtml

[11] Rik Farrow. Distributed Denial of Service Attacks (DDoS), http://chinese-school.netfirms.com/computer-article-denial-of-service.html

[12] CERT Incident Note IN-99-07 http://www.cert.org/incident_notes/IN-99-07.html#trinoo, November 2001.

[13] Distributed Attack Tools, http://packetstorm.decepticons.org/distributed/.

[14] Paul J. Criscuolo. Distributed Denial of Service: Trin00, Tribe Flood Network, Tribe Flood Network 2000, and Stacheldraht, http://www.ciac.org/ciac/documents/CIAC- 2319_Distributed_Denial_of_Service.pdf.

[15] Sven Dittrich, An analysis of the "Shaft" distributed denial of service tool, http://adelphi.edu/~spock/shaft_analysis.txt, November 2001.

[16] Analysizing Distributed Denial of Service Tools: The shaft case http://www.adelphi.edu/~spock/lisa2000-shaft.pdf.

[17] http://staff.washington.edu/dittrich/misc/mstream.analysis.txt

[18] Phatbot Trojan Analysis, http:://www.lurhq.com/phatbot.html

[19] http://www.sans.org/infosecFAQ/threats/face.htm

[20] CERT. Code Red Worm exploiting Buffer Overflow in IIS Indexing Service DLL. http://www.cert.org/advisories/CA-2001-19.html, January 2002.

[21] Nicholas Weaver. A Brief History of The Worm. http://www.securityfocus.com/infocus/1515, November 2001.

[22] http://www.crime-research.org/library/Richard.html

[23] http://www.webopedia.com/TERM/L/local_area_Network_LAN.html

[24] Bradely Mitchell. DMZ- Demilitarized Zone. http://compnetworking.about.com/cs/networksecurity/g/bldef_dmz.htm. August 2004.

[25] Managing the Threats of Denial-of-Service Attacks, CERT/CC http://www.cert.org/archive/pdf/Managing_DoS.pdf, January 2002.

[26] William W. Martin, Honey Pots and Honey Nets - Security through Deception http://rr.sans.org/attack/deception.php, February 2002.

[27] Eric Cole. Hackers Beware, New Riders Publishing, USA, Feb 2002.

[28] Peter Harrison. Monitoring Server Performance http://www.siliconvalleyccie.com/linux-hn/mrtg.htm#_Toc92809393

[29] Thomas E. Daniels Benjamin Kuperman Packet Tracker, Final Report, CERIAS Technical Report 2000-22. http://www.silicondefense.com/research/itrex/archive/tracingpapers/buchholz00 packettracker.pdf, November 2001.

[30] Abraham Yaar, Adrian Perrig and Dawn Song. FIT: Fast Internet Traceback, Proceedings of Infocom, March 2005.

[31] Hal Burch and Bill Cheswick, “Tracing anonymous packets to their approximate source”, Unpublished paper, December 1999.

[32] Michael Goodrich. Efficient packet marking for large-scale IP traceback, Proceedings of the 9th ACM Conference on Computer and Communications Security, Pg 117–126, November 2001.

[33] J. Li, M. Sung, J. Xu, and L. Li. Large-scale IP traceback in high-speed Internet: Practical techniques and theoretical foundation, Proceedings of the IEEE Symposium on Security and Privacy, Pg 115-129, May 2004.

[34] Arjan Durresi, Vamshi Parchuri, Leonard Barolli, Rajagopal Kannan and S.S.Iyengar. Efficient and secure Autonomous System based traceback, Journal of Interconnection Networks, Volume 5, Pg 151-164, May 2004.

[35] M. Fayed, P. Krapivsky, J. Byers, M. Crovella, D. Finkel and S. Render. On the size distribution of Autonomous Systems, Technical Report, Boston University, Jan 2003.

[36] Damein Magoni and Jean Jacques Pansiot. Analysis of the autonomous system network topology, ACM SIGCOMM Computer Communication Review, Vol 3, Pg 26-37, Jul 2001.

[37] J. Mogul and S. Deering. Path MTU Discovery, RFC 1191, Nov 1990.

[38] Alex C. Snoeren, Craig Partridge, Luis A. Sanchez, Christine E. Jones, Fabrice Tchakountio, Beverly Schwartz, Stephen T. Kent and W. Timothy Strayer. Hash based IP Traceback, Proceedings of ACM SIGCOMM 2001, Pg 3-14, August 2001.

[39] S. Kent, C. Lynn, J. Mikkelson, and K. Seo. Secure Border Gateway Protocol (Secure-BGP) - Real World Performance and Deployment Issues, Proceedings of Symposium on Network and Distributed System Security, February 2000.

[40] Stefan Savage, David Wetherall, Anna Karlin and Tom Anderson. Practical Network Support for IP Traceback, Proceedings of ACM SIGCOMM 2000, Pg 295-306, August 2000.

[41] Dawn Xiaodong Song and Adrian Perrig. Advanced and Authenticated Marking Schemes for IP trace back, IEEE infocom 2001, pg 878-886, April 2001.

[42] David Meyer. University of Oregon Route Views Archive Project. http://archive.routeviews.org/, June 2004.

[43] Young Hyun, Andre Broido and K Claffy. Traceroute and BGP AS Path Incongruities, Internetworking 2003 conference, June 2003.

[44] Jimmy Sproles and Will Byars. Cyber Terrorism. http://www-cs.etsu.edu/gotterbarn/stdntppr/#Professionals, March 1998.

[45] Douglas Schweitzer. Internet Relay Chat – IRC the place to be.
http://www/pcflank.com/art32.htm.
[46] Mike Chapple. Egress Filtering.
http://searchsecurity.techtarget.com/tip/129843.html
[47] Gary C. Kessler. Defenses Against Distributed Denial of Service Attacks
http://www.garykessler.net/library/ddos.html, Nov 2000.

APPENDIX: PROGRAMS

HIT.pl

```
#!/usr/bin/perl -w -I./Cryptix-1.16/blib/arch
BEGIN { push @INC, qw(. .. ../lib ../../lib ../../../lib) }
package Crypt::MD5;
use Exporter;
use DynaLoader;
@ISA = qw(Exporter DynaLoader);
bootstrap Crypt::MD5;
$MD5 = new Crypt::MD5;
ref($MD5) || print "Error - $MD5\n";

$deepest = -1;
%attacknodes = ();
$victim = 0;

# Parameters
$samples = 5;
@tests = ( 100,250, 550, 750, 1000,1500);

$markprob = 0.25; # Probability of router to mark a packet
$num_pack = 455;# Number of packets each attacker sends
$match_threshold = 3; # if ( $match > $match_threshold ) range of match is 0
.. 8
$probabilistic = 1;

$storeprob = 2500 / 125000;
# read in all the long paths
print "Read all paths\n";
open( INPUT, "zcat HIT.dat.gz |" ) || die "Could not open and zcat
HIT.dat.gz\n";
@allpaths = ();
while ( $_ = <INPUT> ) {
      if ( $storeprob > rand()) {
            chop;
            @ips = split /\s/, $_;
            if ( $victim == 0 ) {
                  $victim = $ips[0];
            } elsif ( $victim != $ips[0] ) {
                  die "$victim != $ips[0]\n";
            }
            push @allpaths, [@ips];
      }
}
close( INPUT );
print "Done reading all paths, stored $#allpaths paths\n";

print "Read neighbors\n";
%neighbor = ();
if ( 1 ) {
      open( NEIGHBOR, "zcat HITneigh.dat.gz |" ) || die "Could not open
neighbor file\n";
      while ( <NEIGHBOR> ) {
```

```
            chop;
            @ns = split /\s/, $_;
            $neighbor{$ns[0]} = [()];
            for ( $i=1; $i <= $#ns; $i++ ) {
                  if ( $ns[0] != $ns[$i] ) {
                        push @{ $neighbor{$ns[0]}}, $ns[$i];
                  }
            }
      }
      close( NEIGHBOR );
}
print "Done reading neighbors\n";

for $v ( @tests ) {
      $avgfp = 0;
      $avgfn = 0;
        $avgtime = 0;
      print "$v\n";
      for $w ( 1 .. $samples ) {
            # print " .\n";
            simulate( $v );
            $avgfp += $fp;
                $avgtime += $delta_t;
            $avgfn += $fn;

      }
      $avgfp /= $samples;
      $avgfn /= $samples;
      $avgtime /= $samples;
         print "FP  $avgfp\n";
         print "FN  $avgfn\n";
        print "RT $avgtime\n";
}

exit( 0 );

sub compute_routers_at_depth {
      my ( $vip, $distance ) = @_; # vip = victim ip, prev = next hop
      my $i = 0;
      my $att;
      my $curdepth = $routers_at_depth_list[$distance];

      if ( $distance > $deepest ) {
            print "Ups - something is wrong $distance $vip\n";
      }
      if ( ! exists $curdepth->{$vip} ) {
            $curdepth->{$vip} = 0;
            $routers_at_depth_num[$distance]++;
      }
      if ( ! exists $attacknodes{ $distance, $vip } ) {
            # We are at the leaf
            # print "  Leaf\n";
      } else {
            # look through all the attackers
            $att = $attacknodes{ $distance, $vip };
            # Recurse
            for ( $i=0; $i <= $#{$att}; $i++ ) {
```

```
                compute_routers_at_depth( $att->[$i], $distance + 1 );
            }
        }
}

sub MD5_8bitv1 {
    my ($str, $s) = @_;
      my $ret = 0;

      $str = $str . "1" . $s;
      $MD5->reset();
      $MD5->add($str);
      $ret = unpack("L",$MD5->digest()) % (2**11);

    return $ret;
}

sub MD5_8bitv2 {
    my ($str, $s) = @_;
      my $ret = 0;

      $str = $str . "2" . $s;
      $MD5->reset();
      $MD5->add($str);
      $ret = unpack("L",$MD5->digest()) % (2**11);

    return $ret;
}


sub simulate {
      my ($num_attackers) = @_;
      my @attackerpath = ();
      my $num_paths = -1;
      my %attackercount = ();
      my $adjust = 0;
      my @tmp = ();
      my @marked_hash = (); # 3-dimensional array [depth][frag#][maked#]
      my $j;
      my $i;
      my $k;
      my $path;
      my $marked;
      my $depth;
      my $ip;
      my @tmp2;
      my $frag;
      my $ipnum;
      my $hash;
      my $ipnumf;
      my $hashf;
      my $mharr;
      my $found;
      my %tmp;
      my $start_time;
      my $end_time;
      my @suspected_set = ();
```

```
    my $susp;
    my $neigh;
    my $s;
    my $matches;
    my $n;
    my $neighhash;
    my $susphash;
    my @fp = (); # False positive at level
    my @fn = (); # False negative at level
     $delta_t = 0;
    my $rout;
    my @selected = ();

    # print "Simulating $num_attackers attackers\n";
    %attacknodes = ();
    $deepest = -1;
whileloop: while ( $num_attackers > 0 ) {
      do {
            $j = int(rand( $#allpaths + 1 ));
            @tmp = grep { $_ == $j } @selected;


      } until ( $#tmp < 0 );
      $num_attackers--;
      push @selected, $j;
      $k = $allpaths[$j];
      $adjust = 0;
      for ( $i=0; $i < $#{$k}; $i++ ) {
           if ( $k->[$i] != $k->[$i+1] ) {
              push @tmp, $k->[$i];
               if ( ! exists $attacknodes{ $i - $adjust, $k->[$i]} ) {
                $attacknodes{ $i - $adjust, $k->[$i] } = [($k->[$i+1])];
                  $attackercount{ $i - $adjust, $k->[$i], $k->[$i+1] } = 1;
                  }
              else {
                @tmp2 = grep {$_ == $k->[$i+1]} @{$attacknodes{ $i -
                                                    $adjust, $k->[$i] }};
              if ( $#tmp2 < 0 ) {
               push @{ $attacknodes{ $i - $adjust, $k->[$i]} }, $k->[$i+1];
               $attackercount{ $i - $adjust, $k->[$i], $k->[$i+1]} = 1;
                        }
              else {
               $attackercount{ $i - $adjust, $k->[$i], $k->[$i+1]}++;
                   }
                 }
               }
           else {
                  $adjust++;
                }
           }
      push @tmp, $k->[$#{$k}];
      if ( $#tmp > $deepest ) {
            $deepest = $#tmp;
      }
      push @attackerpath, [@tmp];
  }
```

```
# print "Simulate sending and marking packets\n";
for $j ( 0 .. $deepest ) {
      @tmp2 = ();
      for $i ( 0 .. 3 ) {
            $tmp2[$i] = [()];
      }
      $marked_hash[$j] = [@tmp2];
}
if ( $probabilistic ) {
      for ( $i=0; $i <= $#attackerpath; $i++ ) {
            $path = $attackerpath[$i];
            # print "simulating path @{$path}\n";
            for ( $k = 0; $k < $num_pack; $k++ ) {
                  $marked = 0;
                  $depth = 0;
                  for ( $j = $#{$path}; $j >= 1; $j-- ) {
                    if ( rand() < $markprob ) {
                              $marked = 1;
                              $depth = $j;
                              $ip = $path->[$j];
                        }
                  }
                  if ( $marked == 1 ) {
                        # The packet was marked
                        $ipnum = $ip;
                        $frag = int(rand( 4 ));
                        if ( $frag == 4 ) { $frag = 3; }
                        $hash = MD5_8bitv1( $ipnum, $frag );
                        if ( $depth > 1 ) {
                              $ipnumf = $path->[$depth - 1];
                              $hashf = MD5_8bitv2( $ipnumf, $frag );
                              $hash = $hash ^ $hashf;
                        }
                        $mharr = $marked_hash[$depth][$frag];
                        # Only add it if it is not yet in it
                        $found = 0;
                        for $m ( 0 .. $#{$mharr} ) {
                              if ( $mharr->[$m] == $hash ) {
                                    $found = 1;
                                    last;
                              }
                        }
                        if ( $found == 0 ) {
                              # print "Added a sample\n";
                              push @{$mharr}, $hash;
                        }
                  }
            }
      }
} else {
      # deterministic marking
      for ( $i=0; $i <= $#attackerpath; $i++ ) {
        $path = $attackerpath[$i];
        for ( $j = $#{$path}; $j >= 1; $j-- ){$ipnum = $path->[$j];
                  for $frag ( 0 .. 3 ) {
                        $hash = MD5_8bitv1( $ipnum, $frag );
                        if ( $j > 1 ) {
```

```
                                    $ipnumf = $path->[$j - 1];
                                    $hashf = MD5_8bitv2( $ipnumf, $frag );
                                    $hash = $hash ^ $hashf;
                              }
                              $mharr = $marked_hash[$j][$frag];
                              # Only add it if it is not yet in it
                              $found = 0;
                              for $m ( 0 .. $#{$mharr} ) {
                                    if ( $mharr->[$m] == $hash ) {
                                          $found = 1;
                                          last;
                                    }
                              }
                              if ( $found == 0 ) {
                                    push @{$mharr}, $hash;
                              }
                        }
                  }
            }
      }
      # print "Done simulating\n";

      # print "Computing routers_at_depth\n";
      @routers_at_depth_num = ();
      @routers_at_depth_list = ();
      for ( $i=0; $i <= $deepest; $i++ ) {
            %tmp = ();
            $routers_at_depth_list[$i] = {%tmp};
      }
      compute_routers_at_depth( $victim, 0 );
      # print "Routers at depth:\n";
      for $i ( 0 .. $#routers_at_depth_num ) {
            # print "  $i: $routers_at_depth_num[$i]\n";
      }

# start analyzing
      $start_time = time();
      $suspected_set[0] = [($victim)];
      for $depth ( 1 .. $deepest ) {
            $suspected_set[$depth] = [()] ;

            for $susp ( @{ $suspected_set[$depth-1] } ) {
                  if ( ! exists $neighbor{ $susp } ) {
                        $s =  $susp ;
                        die "$s has no neigbors\n";
                  }
                  $neigh = $neighbor{$susp};
                   for $n ( @{$neigh} ) {
                        $matches = 0;
                        for $frag ( 0 .. 3 ) {
                              $neighhash = MD5_8bitv1( $n, $frag );
                  if ( $depth > 1 ) {
                              $susphash = MD5_8bitv2( $susp, $frag );
                               $neighhash = $neighhash ^ $susphash;
                              }
                              @tmp = grep {$_ == $neighhash}
                                        @{$marked_hash[$depth][$frag]};
```

```
                              if ( $#tmp >= 0 ) {
                                    $matches++;
                              }
                        }
                        if ( $matches > $match_threshold ) {
                        @tmp = grep {$_ == $n} @{$suspected_set[$depth]};
                              if ( $#tmp < 0 ) {
                                    push @{$suspected_set[$depth]}, $n;
                              #$tmp =  $n ;
                              #print "Found a suspect at level $depth:$tmp\n;
                              }
                        }
                  }
            }
      }
      $end_time = time();
      # print "Analyzing results\n";
      $fp = $fn = 0;
      for $i ( 1 .. $deepest ) {
            $fp[$i] = 0;
            $fn[$i] = 0;
            $susp = $suspected_set[$i];
            $rout = $routers_at_depth_list[$i];
            for $j ( 0 .. $#{$susp} ) {
                  if ( exists $rout->{ $susp->[$j] } ) {
                        delete $rout->{ $susp->[$j] };
                  } else {
                        # $k = ipnum2str( $susp->[$j] );
                        # print "False positive at level $i: $k\n";
                        $fp[$i]++;
                        $fp++;
                  }
            }
            foreach $j ( keys %{$rout} ) {
                  # $k = ipnum2str( $j );
                  # print "False negative at level $i: $k\n";
                  $fn[$i]++;
                  $fn++;
            }
      }
#     print "Number of false positives:\n";
#     for $i ( 1 .. $deepest ) {
#           print "   $i: $fp[$i]\n";
#     }
#     print "Number of false negatives:\n";
#     for $i ( 1 .. $deepest ) {
#           print "   $i: $fn[$i]\n";
#     }
      $delta_t = $end_time - $start_time;
#     print "The analysis took $delta_t seconds\n";
#     print "Done.\n";
}
```

FAST.pl

```
#!/usr/bin/perl -w -I./Cryptix-1.16/blib/arch
BEGIN { push @INC, qw(. .. ../lib ../../lib ../../../lib) }
#initialize MD5 stuff
package Crypt::MD5;
use Exporter;
use DynaLoader;
@ISA = qw(Exporter DynaLoader);
bootstrap Crypt::MD5;
$MD5 = new Crypt::MD5;
ref($MD5) || print "Error - $MD5\n";

package FAST;
$MD5 = new Crypt::MD5;
%MD5hasher = ();
$NUMPACKETS= 12;

#Hash Function
sub hash
{
  my $hash;
  $MD5->reset();
  $MD5->add($_[0],$_[1]);
  $hash=unpack("L",$MD5->digest()) % 16;
  return $hash;
}

 sub round
{
    my($number) = shift;
    return int($number + .5);
}

open(FILE,'Fast100AS.dat');
open(FILE1,'>FASTpaths.dat');
@line=<FILE>;
chop @line;
@Fid =(); @Mark=();
for ($i=0;$i<=$#line;$i++)
{#print" line $i--------------------";
 @record=();$j=0;
 @recordit = split /\s/, $line[$i];
  for($k=0;$k<=$#recordit;$k++)
   {
    if($recordit[$k] ne $recordit[$k+1])
     {
      $record[$j]=$recordit[$k];if($j ne 0) { print FILE1 "$record[$j] ";}
      $j++;
     }
   }
 for($pkt=0;$pkt<$NUMPACKETS;$pkt++)
   {$dummy=rand(rand(rand(rand(time()))));
    srand($dummy ^ $$);
    $Marking[$pkt]='';$fid=round(rand(3));
    #print"Pkt =$pkt\n";
```

```
     for($j=1;$j<=$#record;$j++)
      {
       $Marking[$pkt]=$Marking[$pkt].hash($record[$j],$fid).'.';
      }$cnt=-1;#print "Marking $Marking[$pkt]\n";
        for($p=0;$p<=$#Mark;$p++)
         {#print "$Marking[$pkt] eq $Mark[$p] and $fid eq $Fid[$p]\n";
          if($Marking[$pkt] eq $Mark[$p] and $fid eq
$Fid[$p]){$p=$#Mark+1;}else {$cnt =$cnt+1;}
         }
          if($cnt eq
$#Mark){push(@Mark,$Marking[$pkt]);push(@Fid,$fid);}#print"Mark @Mark\n Fid
@Fid\n";}

    }
print FILE1 "\n";
}

$MarkCut=$#Mark;
1;
********************************************************************************

#!/usr/bin/perl -w -I./Cryptix-1.16/blib/arch

require 'FAST.pl';

BEGIN { push @INC, qw(. .. ../lib ../../lib ../../../lib) }
#initialize MD5 stuff
package Crypt::MD5;
use Exporter;
use DynaLoader;
@ISA = qw(Exporter DynaLoader);
bootstrap Crypt::MD5;
$MD5 = new Crypt::MD5;
ref($MD5) || print "Error - $MD5\n";

$MD5 = new Crypt::MD5;
%MD5hasher = ();
$victim = 27;

sub hash
{
  my $hash;
  $MD5->reset();
  $MD5->add($_[0],$_[1]);
  $hash=unpack("L",$MD5->digest()) % 16;
  return $hash;
}
 open(File,"FASTNoRep.dat") || die "Could not open neighbor file 1\n";
 @line1=<File>; chop @line1;
for($r=0;$r<$#line1;$r++)
   {
    @arr= split /\s/, $line1[$r];@harr0=();@harr1=();@harr2=();@harr3=();
    for($i=0;$i<=$#arr;$i++)
    {
     push @harr0,hash($arr[$i],0);#print "rr= @harr0\n";
     push @harr1,hash($arr[$i],1);
     push @harr2,hash($arr[$i],2);
```

```
     push @harr3,hash($arr[$i],3);
    }
   push @hsec0,[@harr0];#print "sec $hsec0[$r]->[1]\n";
   push @hsec1,[@harr1];
   push @hsec2,[@harr2];
   push @hsec3,[@harr3];
 }
push @Table,[@hsec0],[@hsec1],[@hsec2],[@hsec3];
$Tablecnt=$#{($Table->[0])->[0]};
$t=($Table[0]->[0])->[3];
#print "This is $t";
close (File);

$Start=time();
 for($i=0;$i<=$FAST::MarkCut;$i++)
  {
   @Marks=();
   @Marks = split /\./, $FAST::Mark[$i];#print"Marks @Marks\n";
   for($hid=0;$hid<=$#line1;$hid++)
    {
     $p=$#{$Table[$FAST::Fid[$i]]->[$hid]};
     if($#Marks <= $p)
     {
      $cnt=-1;
      for($j=0;$j<=$#Marks;$j++)
       {
       if($Marks[$j] ne ($Table[$FAST::Fid[$i]]->[$hid])->[$j+1])
        {
           $j=$#Marks+1;
        }
       else{$cnt=$cnt+1;}
       }
     if($cnt eq $#Marks){
                         print "$line1[$hid]\n";
                          }
    }
  }
}
 $End=time();
$Time=$End-$Start;
print" Time = $Time\n";

*****************************************************************************


#!/usr/bin/perl -w -I./Cryptix-1.16/blib/arch

open(file,'FASTATT.dat');
#open(file2,'FASTpaths.dat');
open(file2,'Fast100AS.dat');
@val=<file>;
@path=<file2>;
chop @val;chop @path;
$m=3;$fp=0;$Recon=0;
$start=time();
for($i=0;$i<=$#val;$i++)
```

```
{
 $cnt=1;
 for($j=$i+1;$j<=$#val;$j++)
 {
  if($val[$i] eq $val[$j] and $val[$i] ne ' ')
   {
    $cnt=$cnt+1;$val[$j]=' ';
   }
 }$dummy=0;
  if($cnt >= $m)
   {print"R $val[$i]\n";
    $Recon=$Recon+1;
    for($k=0;$k<=$#path;$k++)
     {
    #  print "$val[$i] eq $path[$k]\n";
        @first=split /\s/,$val[$i];@sec=split /\s/,$path[$k];$node=$val[$i];
     # if($val[$i] eq $path[$k])
        if($#sec eq $#first)
        {$b=-1;
         for($n=0;$n<=$#sec;$n++)
         {
           if($first[$n] eq $sec[$n])
            {# print "$first[$n] eq $sec[$n]\n";
              $b=$b+1;
            }
          }
          if($b eq $#sec) {$k=$#path+1;}# print "ATT $node \n";}
           else {$dummy=$dummy+1;}
          }
         else {$dummy = $dummy+1;}#print "Dum =$dummy\n";}
       }
     if($dummy eq $#path+1){$fp=$fp+1;}
  }
}
$end=time();
$Time = $end-$start;
print"False Positives $fp\n";
$Fn=$Recon-($#path+1)-$fp;print "False Negatives $Fn =$Recon-($#path+1)-$fp";
print"\n$Time";
```

STM.pl

```
#!/usr/bin/perl -w -I./Cryptix-1.16/blib/arch
BEGIN { push @INC, qw(. .. ../lib ../../lib ../../../lib) }
#initialize MD5 stuff
package Crypt::MD5;
use Exporter;
use DynaLoader;
#use strict;
#use warnings;

@ISA = qw(Exporter DynaLoader);
bootstrap Crypt::MD5;
$MD5 = new Crypt::MD5;
ref($MD5) || print "Error - $MD5\n";

$MD5 = new Crypt::MD5;
%MD5hasher = ();

$victim = 27;
$p=.25;
@test=(100);
open(File,"05.ASNpath.dat") || die "Could not open neighbor file 1\n";
open(File3,"Topology.dat");
open(File1,"031.IPpaths.dat") || die "Could not open neighbor file 1\n";
#open(File,"temp.dat");open(File1,"temp1.dat");open(File2,"temp2.dat");
open(File2,"zcat HIT.dat.gz |")|| die "Could not open AS topology file 1\n";

@line=<File>; chop @line;
@line1=<File1>; chop @line1;
@line2=<File2>; chop @line2;
@line3=<File3>; chop @line3;

#Toplology of AS
#for($k=0;$k<$#line;$k++)
#{
# @records=split /\s/,$line[$k];
# push(@attacker,$records[$#records]);
# for($j=1;$j<=$#records;$j++)
#  {
#   if((grep{$_ == $records[$j]}@topology) eq
0){push(@topology,$records[$j]);}
#  }
#}
@topology=split /\s/,$line3[0];

#Main Program
for $v (@test)
{
 $Fp=0;
 $Fn=0;
 $Rt=0;
 simulate($v);
#$Fn = $#attackpath-($#Rep-$Fp);
#$Rt=$end-$start;
```

```
print" $v\n.....\nFP = $Fp\n";print" FN $Fn\n";
print" RT $Rt\n";
}
*****************************************************************************

#Hash Function
sub hash
{
  my $hash;
  $MD5->reset();
  $MD5->add($_[0],$_[1]);
  $hash=unpack("L",$MD5->digest()) % (2**$_[2]);
  return $hash;
}

sub Norep
{my $k;my @record=();
 for($k=0;$k<=$#_;$k++)
  {
   if($_[$k] ne $_[$k+1])
     {
       push(@record,$_[$k]);
     }
  }
return @record;
}
*****************************************************************************

sub simulate
{
 @attackpath=();@Pks=();@hashes=();@Nodes=();@AttAS=();@AttIP=();
 $dummy=rand(rand(rand(rand(time()))));
 srand($dummy ^ $$);

 for($i=0;$i<=$_[0];$i++)
 {
  $r=int(rand(125000));@recordAS=();@recordIP=();
  @recordAS=split /\s/,$line[$r];print "line $i\n";print
"@recordAS\n";push(@ActAS,@recordAS);@attackpath=Norep(sort(@ActAS));
  @recordIP=split /\s/,$line1[$r];print
"@recordIP\n";push(@ActIPs,@recordIP);@ActIP=Norep(sort(@ActIPs));
 for($pkt=0;$pkt<=500;$pkt++)
  {
   @Pks=();$d=0;
   for($j=$#recordAS;$j>=0;$j--)
   {
     $cnt=-1;
     $r=rand(1);
     $hid=int(rand(4));
     if($r < .01)
      {$flag=0;$temp=$recordAS[$j];if((grep{$_ == $recordAS[$j]}@attackpath)
eq 0){push(@attackpath,$recordAS[$j]);}}#print"This is AS marking
$recordAS[$j]\n";}
     else{if($r<.04 and $r > .01){$flag=1;$temp=$recordIP[$j];}}#print"This
is IP marking]\n";}
          # print "Num of Nodes $#Nodes\n";
    }
```

```
         for($q=0;$q<=$#Nodes;$q++)
          {#print"$Nodes[$q]->[0] eq hash($temp,$hid,13) and $Nodes[$q]->[1]
eq $hid and $Nodes[$q]->[2] eq $flag\n";
          if($Nodes[$q]->[0] eq hash($temp,$hid,13) and $Nodes[$q]->[1] eq
$hid and $Nodes[$q]->[2] eq $flag){}else{$cnt=$cnt+1;}#print"cnt=$cnt\n node-
$#node
          }
          if($cnt == $#Nodes)
          {@Pks=();
           push(@Pks,hash($temp,$hid,13),$hid,$flag); push
@Nodes,[@Pks];#print"@Pks\n";
          }
   }
 }

print"Attackpath @attackpath\n";
print"Marking is done\n";
*****************************************************************************

#Reconstruction
$start=time();

@Sys=();@s=();
for($q=0;$q<=$#Nodes;$q++)
{
 if($Nodes[$q]->[2] eq 0)
 {
  @Sys=grep{hash($_,$Nodes[$q]->[1],13)  eq $Nodes[$q]->[0]}@topology;
   push(@s,grep{$_ >= 0}@Sys);
 }
}
#print"s=@s\n";

@Rep=();@Cnt=();
for($t=0;$t<=$#s;$t++)
{
  @Cnt=grep{$_ == $s[$t]}@s;
   if($#Cnt>=1)
    {
      if((grep{$_ == $Cnt[0]}@Rep) eq 0)
      {
       push(@Rep,$Cnt[0]);
      }
    }
}
print"The attack originating ASes\n@Rep\n";

for($k=0;$k<$#line2;$k++)
{
$cnt=-1;
 @records=split /\s/,$line2[$k];
 @recordsIP=split /\s/,$line1[$k];
 for($s=0;$s<=$#records;$s++)
  {
   if((grep{$_ == $records[$s]}@Rep) ne 0)
    {
     $cnt=$cnt+1;
```

```
    }
  }
 if($cnt==$#records){push (@IP,@recordsIP);push @count,
[@recordsIP];}#print"recordIP = @records\n";}
}
print "\nNumber of paths in IP topology $#count\n";
#Remove Repetitions in IP\n

@out1=Norep(sort(@IP));
print "The IP topology\n @out1\n";

@s=();@Sys=();
for($q=0;$q<=$#Nodes;$q++)
{
 if($Nodes[$q]->[2] eq 1)
 {
  @Sys=grep{hash($_,$Nodes[$q]->[1],13) == $Nodes[$q]->[0]}@out1;#print"Sys
@Sys\n";
   push(@s,@Sys);
 }
}

@Suspect=();@Cnt=();
for($t=0;$t<=$#s;$t++)
{
  @Cnt=grep{$_ eq $s[$t]}@s;
   if($#Cnt>=1)
    {
      if((grep{$_ eq $Cnt[0]}@Suspect) eq 0)
      {
       push(@Suspect,$Cnt[0]);
      }
    }
}
print"Suspect @Suspect\n";
*************************************************************************

print"Calculating the number of False Positives\n";
for($l=0;$l<=$#ActIP;$l++)
{
 push(@Pre,(grep{$_ eq $ActIP[$l]}@Suspect));
}

$end=time();
$Time=$end-$start;
print"\nAttackers in the suspected set \n @Pre\n";
print"\nActual Attack graph\n @ActIP\n";
$Fn=$#ActIP-$#Pre;
$Fp=$#Suspect+$Fn-$#ActIP;print
print "\nFalse Negatives $Fn\nFalse Positives $Fp";
print"\nRT = $Time";
}
```

STM2.pl

```
#!/usr/bin/perl -w -I./Cryptix-1.16/blib/arch
BEGIN { push @INC, qw(. .. ../lib ../../lib ../../../lib) }
#initialize MD5 stuff
package Crypt::MD5;
use Exporter;
use DynaLoader;

@ISA = qw(Exporter DynaLoader);
bootstrap Crypt::MD5;
$MD5 = new Crypt::MD5;
ref($MD5) || print "Error - $MD5\n";

$MD5 = new Crypt::MD5;
%MD5hasher = ();

$victim = 27;
$p=.25;
@test=(50,150,250,500,1000);
open(File,"05.ASNpath.dat") || die "Could not open AS file 1\n";
open(File3,"Topology.dat");
open(File1,"031.IPpaths.dat") || die "Could not open IP file 1\n";
#open(File,"temp.dat");
#open(File1,"temp1.dat");open(File2,"temp2.dat");
open(File2,"zcat HIT.dat.gz |")|| die "Could not open AS topology file 1\n";

@line=<File>; chop @line;
@line1=<File1>; chop @line1;
@line2=<File2>; chop @line2;
@line3=<File3>;chop @line3;
**************************************************************************

#Toplology of AS
#for($k=0;$k<$#line2;$k++)
#{@records=();
# @records=split /\s/,$line2[$k];
# push(@attacker,$records[$#records]);
# for($j=0;$j<=$#records;$j++)
#  {
#   if((grep{$_ == $records[$j]}@topology) eq
0){push(@topology,$records[$j]);}
#  }
#}
@topology=split /\s/,$line3[0];print"@topology";
**************************************************************************

#Main Program
for $v (@test)
{
 $Fp=0;
 $Fn=0;
 $Rt=0;
 simulate($v);
```

```
#$Fn = $#attackpath-($#Rep-$Fp);
#$Rt=$end-$start;
print" $v\n.....\nFP = $Fp\n";print" FN $Fn\n";
print" RT $Rt\n";
}
*****************************************************************************

#Hash Function
sub hash
{
  my $hash;
  $MD5->reset();
  $MD5->add($_[0],$_[1]);
  $hash=unpack("L",$MD5->digest()) % (2**$_[2]);
  return $hash;
}
*****************************************************************************

#Remove Repetitions
sub Norep
{my $k;my @record=();
 for($k=0;$k<=$#_;$k++)
  {
   if($_[$k] ne $_[$k+1])
     {
       push(@record,$_[$k]);
     }
  }
return @record;
}
*****************************************************************************

#IP Topology
sub order
{
 my $i;
 for($i=0;$i<25;$i++)
 {
  my $k=0;$t=0;$s=0;
  while($t<=$#count)
    {$we=index($IPd[$i][$t],'.');
    if($IPd[$i][$t] ne '#' and $we > 1)
    {
    for ($j=$t;$j<$#count;$j++)
      {
      if ($IPd[$i][$t] eq $IPd[$i][$j+1] && $IPd[$i][$j+1] ne '#' )
           {
             $IPd[$i][$j+1]= '#';
           }
      }

    $OrderedIP[$i][$s]=$IPd[$i][$t];print"Ordered $OrderedIP[$i][$s]\n";
    $countIP[$i]++;
    $s++;
   }
      $t++;
  }
```

```
 print " Count of different nodes at hop $i countas[$i] = $countIP[$i]\n";
 }
}
*****************************************************************************

# MARKING
sub simulate
{
 @attackpath=();@Pks=();@hashes=();@Nodes=();@AttAS=();@AttIP=();
 $dummy=rand(rand(rand(rand(time()))));
 srand($dummy ^ $$);

 for($i=0;$i<=$_[0];$i++)
 {
  $r=int(rand(125000));@recordAS=();@recordIP=();
  @recordAS=split /\s/,$line[$r];print "line $i\n";print
"@recordAS\n";push(@ActAS,@recordAS);@attackpath=Norep(sort(@ActAS));
  @recordIP=split /\s/,$line1[$r];print
"@recordIP\n";push(@ActIPs,@recordIP);@ActIP=Norep(sort(@ActIPs));
 for($pkt=0;$pkt<=1000;$pkt++)
  {
   @Pks=();$d=0;
   for($j=$#recordAS;$j>=0;$j--)
   {
     $cnt=-1;
     $r=rand(1);
     $hid=int(rand(4));
     if($r < .01) {$flag=0;$temp=$recordAS[$j];$d=0;}

     if($r < .04 and $r >= .01){$flag=1;$temp=$recordIP[$j];$d=0;}#print"This
is IP marking]\n";}
          # print "Num of Nodes $#Nodes\n";
     if($r >= .04){$d=$d+1;}
  }
         for($q=0;$q<=$#Nodes;$q++)
          {#print"$Nodes[$q]->[0] eq hash($temp,$hid,12) and $Nodes[$q]->[1]
eq $hid and $Nodes[$q]->[2] eq $flag and $Nodes[$q]->[3] eq $d\n";
          if($Nodes[$q]->[0] eq hash($temp,$hid,12) and $Nodes[$q]->[1] eq
$hid and $Nodes[$q]->[2] eq $flag and $Nodes[$q]->[3] eq
$d){}else{$cnt=$cnt+1;}#print"cnt=$cnt\n node- $#node
          }
          if($cnt == $#Nodes)
          {@Pks=();
           push(@Pks,hash($temp,$hid,12),$hid,$flag,$d); push
@Nodes,[@Pks];#print"@Pks\n";
          }
   }
 }

print"Attackpath\n @attackpath\n";
print"Marking is done\n";
*****************************************************************************

#Reconstruction
$start=time();
@Sys=();@s=();
for($q=0;$q<=$#Nodes;$q++)
```

```
{
 if($Nodes[$q]->[2] eq 0)
 {
  @Sys=grep{hash($_,$Nodes[$q]->[1],12) eq $Nodes[$q]->[0]}@topology;
   push(@s,grep{$_ >= 0}@Sys);
 }
}
#print"s=@s\n";

@Rep=();@Cnt=();
for($t=0;$t<=$#s;$t++)
{
  @Cnt=grep{$_ == $s[$t]}@s;
   if($#Cnt>=2)
    {
      if((grep{$_ == $Cnt[0]}@Rep) eq 0)
      {
       push(@Rep,$Cnt[0]);
      }
    }
}
print"The attack originating ASes\n@Rep\n";
*****************************************************************************

#IP topology with each attacking AS
for($k=0;$k<$#line2;$k++)
{
$cnt=-1;@records=();@recordsIP=();
 @records=split /\s/,$line2[$k];
 @recordsIP=split /\s/,$line1[$k];
 for($s=0;$s<=$#records;$s++)
  {
   if((grep{$_ == $records[$s]}@Rep) ne 0)
    {
     $cnt=$cnt+1;
     }
  }
 if($cnt==$#records)
  {push @count, [@recordsIP];
   for($dist=0;$dist<=$#recordsIP;$dist++)
   {
     $IPd[$dist][$pos]= $recordsIP[$dist];print "$IPd[$dist][$pos]\n"; #$$
push (@IP,@recordsIP); print"recordIP = @records\n";
   }
  $pos=$pos+1;
 }
}

order();

print "\nNumber of paths in IP topology $#count\n";
#Remove Repetitions in IP\n

#@out1=Norep(sort(@IP));
#print "The IP topology\n @out1\n";
*********************************************************************
```

```
#Identifying the suspect set
@s=();
for($q=0;$q<=$#Nodes;$q++)
{
 if($Nodes[$q]->[2] eq 1)
 {
  for($IPcnt=0;$IPcnt<$countIP[$Nodes[$q]->[3]];$IPcnt++)
   {#print"d= $Nodes[$q]->[3]  hash($OrderedIP[$Nodes[$q]-
>[3]][$IPcnt],$Nodes[$q]->[1],12)",hash($OrderedIP[$Nodes[$q]-
>[3]][$IPcnt],$Nodes[$q]->[1],12)," == $Nodes[$q]->[0]\n";
    if(hash($OrderedIP[$Nodes[$q]->[3]][$IPcnt],$Nodes[$q]->[1],12) ==
$Nodes[$q]->[0])
     {
      # @Sys=grep{hash($_,$Nodes[$q]->[1],12) == $Nodes[$q]-
>[0]}@out1;#print"Sys @Sys\n";push(@s,@Sys);
       push(@s,$OrderedIP[$Nodes[$q]->[3]][$IPcnt]);#print"Entered\n"
     }
   }
 }
}
print"HELLO @s\n";

@Suspect=();@Cnt=();
for($t=0;$t<=$#s;$t++)
{
  @Cnt=grep{$_ eq $s[$t]}@s;
   if($#Cnt>=2)
    {
      if((grep{$_ eq $Cnt[0]}@Suspect) eq 0)
      {
       push(@Suspect,$Cnt[0]);
      }
    }
}
print"Suspect @Suspect\n";
*************************************************************************

print"Calculating the number of False Positives\n";
for($l=0;$l<=$#ActIP;$l++)
{
 push(@Pre,(grep{$_ eq $ActIP[$l]}@Suspect));
}

$end=time();
$Time=$end-$start;
print"\nAttackers in the suspected set\n @Pre - $#Pre\n";
print"\nActual Attack graph\n @ActIP - $#ActIP\n";
$Fn=$#ActIP-$#Pre;
$Fp=$#Suspect+$Fn-$#ActIP;
print "\nFalse Negatives $Fn\False Positives $Fp\n RT = $Time";
}
```

STM3.pl

```
#!/usr/bin/perl -w -I./Cryptix-1.16/blib/arch
BEGIN { push @INC, qw(. .. ../lib ../../lib ../../../lib) }
#initialize MD5 stuff
package Crypt::MD5;
use Exporter;
use DynaLoader;

@ISA = qw(Exporter DynaLoader);
bootstrap Crypt::MD5;
$MD5 = new Crypt::MD5;
ref($MD5) || print "Error - $MD5\n";

$MD5 = new Crypt::MD5;
%MD5hasher = ();

$victim ='128.8.7.4';
$p=.25;
@test=(5);
#open(File,"05.ASNpath.dat") || die "Could not open neighbor file 1\n";
#open(File3,"Topology.dat");
#open(File1,"031.IPpaths.dat") || die "Could not open neighbor file 1\n";
open(File,"temp.dat");
open(File1,"temp1.dat");open(File2,"temp2.dat");
#open(File2,"zcat HIT.dat.gz |")|| die "Could not open AS topology file 1\n";

@line=<File>; chop @line;
@line1=<File1>; chop @line1;
@line2=<File2>; chop @line2;
@line3=<File3>;chop @line3;
**************************************************************************

#Toplology of AS
for($k=0;$k<$#line2;$k++)
{@records=();
 @records=split /\s/,$line2[$k];
 push(@attacker,$records[$#records]);
 for($j=0;$j<=$#records;$j++)
  {
   if((grep{$_ == $records[$j]}@topology) eq
0){push(@topology,$records[$j]);}
  }
}
#@topology=split /\s/,$line3[0];#print"@topology";
**************************************************************************

#Main Program
for $v (@test)
{
 $Fp=0;
 $Fn=0;
 $Rt=0;
 simulate($v);
#$Fn = $#attackpath-($#Rep-$Fp);
```

```
#$Rt=$end-$start;
print" $v\n.....\nFP = $Fp\n";print" FN $Fn\n";
print" RT $Rt\n";
}
*****************************************************************************

#Hash Function
sub hash
{
  my $hash;
  $MD5->reset();
  $MD5->add($_[0],$_[1]);
  $hash=unpack("L",$MD5->digest()) % (2**$_[2]);
  return $hash;
}
*****************************************************************************

sub Norep
{my $k;my @record=();
 for($k=0;$k<=$#_;$k++)
  {
   if($_[$k] ne $_[$k+1])
     {
       push(@record,$_[$k]);
     }
  }
return @record;
}
*****************************************************************************

sub order
{
 my $i;
 for($i=0;$i<25;$i++)
 {
  my $k=0;$t=0;$s=0;
  while($t<=$#count)
    {$we=index($IPd[$i][$t],'.');
    if($IPd[$i][$t] ne '#' and $we > 1)
    {
    for ($j=$t;$j<$#count;$j++)
      {
      if ($IPd[$i][$t] eq $IPd[$i][$j+1] && $IPd[$i][$j+1] ne '#' )
           {
             $IPd[$i][$j+1]= '#';
           }
      }

    $OrderedIP[$i][$s]=$IPd[$i][$t];#print"Ordered $OrderedIP[$i][$s]\n";
    $countIP[$i]++;
    $s++;
   }
      $t++;
  }
 print " Count of different nodes at hop $i countas[$i] = $countIP[$i]\n";
 }
}
```

```
*************************************************************************


sub compute_routers_at_depth {
        my ( $vip, $distance ) = @_; # vip = victim ip, prev = next hop
        my $i = 0;
        my $att;
        my $curdepth = $routers_at_depth_list[$distance];

        if ( $distance > $deepest ) {
                print "Ups - something is wrong $distance $vip\n";
        }
        if ( ! exists $curdepth->{$vip} ){
                $curdepth->{$vip} = 0;
                $routers_at_depth_num[$distance]++;
        }
        if ( ! exists $attacknodes{ $distance, $vip } ) {
                # We are at the leaf
                # print "  Leaf\n";
        } else {
                # look through all the attackers
                $att = $attacknodes{ $distance, $vip };
                # Recurse
                for ( $i=0; $i <= $#{$att}; $i++ ) {
                        compute_routers_at_depth( $att->[$i], $distance + 1
);
                }
        }
}

*************************************************************************

# MARKING
sub simulate
{
 @attackpath=();@Pks=();@hashes=();@Nodes=();@AttAS=();@AttIP=();
 $dummy=rand(rand(rand(rand(time()))));
 srand($dummy ^ $$);

 for($i=0;$i<=$_[0];$i++)
 {
  $r=int(rand(100));@recordAS=();@recordIP=();
  @recordAS=split /\s/,$line[$r];#print "line $i\n";print "@recordAS\n";
  push(@ActAS,@recordAS);@attackpath=Norep(sort(@ActAS));
  @recordIP=split /\s/,$line1[$r];#print "@recordIP\n";
  push(@ActIPs,@recordIP);@ActIP=Norep(sort(@ActIPs));
 for($pkt=0;$pkt<=300;$pkt++)
  {
   @Pks=();$d=0;
   for($j=$#recordAS;$j>=0;$j--)
   {
     $cnt=-1;
     $r=rand(1);
     $hid=int(rand(4));
     if($r < .01) {$flag=0;$temp=$recordAS[$j];$d=0;}
```

```
     if($r < .04 and $r >= .01){$flag=1;$temp=$recordIP[$j];$d=0;}#print"This
is IP marking]\n";}
          # print "Num of Nodes $#Nodes\n";
     if($r >= .04){$d=$d+1;}
  }
         for($q=0;$q<=$#Nodes;$q++)
          {#print"$Nodes[$q]->[0] eq hash($temp,$hid,12) and $Nodes[$q]->[1]
eq $hid and $Nodes[$q]->[2] eq $flag and $Nodes[$q]->[3] eq $d\n";
          if($Nodes[$q]->[0] eq hash($temp,$hid,12) and $Nodes[$q]->[1] eq
$hid and $Nodes[$q]->[2] eq $flag and $Nodes[$q]->[3] eq
$d){}else{$cnt=$cnt+1;}#print"cnt=$cnt\n node- $#node
          }
          if($cnt == $#Nodes)
          {@Pks=();
           push(@Pks,hash($temp,$hid,12),$hid,$flag,$d); push
@Nodes,[@Pks];#print"@Pks\n";
          }
   }
 }

print"Attackpath\n @attackpath\n";
print"Marking is done\n";
*************************************************************************

$start=time();

@Sys=();@s=();
for($q=0;$q<=$#Nodes;$q++)
{
 if($Nodes[$q]->[2] eq 0)
 {
  @Sys=grep{hash($_,$Nodes[$q]->[1],12) eq $Nodes[$q]->[0]}@topology;
   push(@s,grep{$_ >= 0}@Sys);
 }
}
#print"s=@s\n";

@Rep=();@Cnt=();
for($t=0;$t<=$#s;$t++)
{
  @Cnt=grep{$_ == $s[$t]}@s;
   if($#Cnt>=2)
    {
      if((grep{$_ == $Cnt[0]}@Rep) eq 0)
      {
       push(@Rep,$Cnt[0]);
      }
    }
}
print"The attack originating ASes\n@Rep\n";
*************************************************************************

$pos=0; %attacknodes = ();
 $deepest = -1;
 %attackercount=();

for($j=0;$j<=$#line2;$j++)
```

```
{
 $cnt=-1;@records=();@recordsIP=();
  @records=split /\s/,$line2[$j];
  @recordsIP=split /\s/,$line1[$j];
 for($s=0;$s<=$#records;$s++)
  {
   if((grep{$_ == $records[$s]}@Rep) ne 0)
    {
     $cnt=$cnt+1;
     }
  }
 @tmp=();
  if($cnt == $#records)
  {
    $k =[@recordsIP];print"@recordsIP\m";
     $adjust = 0;
          for ( $i=0; $i < $#{$k}; $i++ ) {
                  if ( $k->[$i] ne $k->[$i+1] ) {
                     push @tmp, $k->[$i];
           if ( ! exists $attacknodes{ $i - $adjust, $k->[$i]} ) {
               $attacknodes{ $i - $adjust, $k->[$i] } = [($k->[$i+1])];
                 $attackercount{ $i - $adjust, $k->[$i], $k->[$i+1] } = 1;
                  }
     else {
      @tmp2 = grep{$_ eq $k->[$i+1]}@{$attacknodes{$i - $adjust, $k->[$i] }};
              if ( $#tmp2 < 0 ) {
                push @{ $attacknodes{ $i - $adjust, $k->[$i]} }, $k->[$i+1];
                  $attackercount{ $i - $adjust, $k->[$i], $k->[$i+1]} = 1;
                  } else {
                    $attackercount{ $i - $adjust, $k->[$i], $k->[$i+1]}++;
                          }
                  }
                 } else {
                          $adjust++;
                         }
          }
          push @tmp, $k->[$#{$k}];
         $att1=$attacknodes{0,$victim};
         print"att1",$att1->[0],"\n";

          if ( $#tmp > $deepest ) {
                  $deepest = $#tmp;print"deepest - $deepest\n";
          }
          push @IPtopology, [@tmp];
  }
}

 print "Computing routers_at_depth\n";
        @routers_at_depth_num = ();
        @routers_at_depth_list = ();
        for ( $i=0; $i <= $deepest; $i++ ) {
                %tmp = ();
                $routers_at_depth_list[$i] = {%tmp};
        }
        compute_routers_at_depth( $victim, 0 );
         print "Routers at depth:\n";$all_routers=0;
```

```
        for $i ( 0 .. $#routers_at_depth_num )
{$all_routers=$all_routers+$routers_at_depth_num[$i];
               print "  $i: $routers_at_depth_num[$i]\n";
             }
       $att1=$attacknodes{0,$victim};
        print"att1",$att1->[0],"\n";
      print "Coun $all_routers\n";

     @Arr=();$t=0;$n=1;$k=1;
       push(@Arr,$victim);$vip=$victim;

      for($i=0;$i<$all_routers;$i++)
        {
          if($k eq 0){$k=$routers_at_depth_num[$n];$n=$n+1;$t=$t+1;}
            $arr1=$attacknodes{$t,$vip};
            for($rp=0;$rp<=$#{$arr1};$rp++)
             {
              push(@Arr,$arr1->[$rp]);#print "Arr @Arr";
             }
            $k=$k-1;
           $vip=$Arr[$i+1]
        }
       print"ARR @Arr\n -$#Arr";

$first=0;$last=$routers_at_depth_num[0];
for($hop=0;$hop<$#routers_at_depth_num ;$hop++)
 {@A=();
  for($cnt=$first;$cnt<$last;$cnt++)
  {
   push(@A,$Arr[$cnt]);
  }
  push @DistArr,[@A];print"A @A";
  $first=$last;
  $last=$first+$routers_at_depth_num[$hop+1];
}
*************************************************************************

@s=();
for($q=0;$q<=$#Nodes;$q++)
{
 if($Nodes[$q]->[2] eq 1)
 {
  for($IPcnt=0;$IPcnt<=$#{$DistArr[$Nodes[$q]->[3]]};$IPcnt++)
   {print"d= $Nodes[$q]->[3]  hash(",$DistArr[$Nodes[$q]->[3]]-
>[$IPcnt],",$Nodes[$q]->[1],12)",hash($DistArr[$Nodes[$q]->[3]]-
>[$IPcnt],$Nodes[$q]->[1],12)," == $Nodes[$q]->[0]\n";
    if(hash($DistArr[$Nodes[$q]->[3]]->[$IPcnt],$Nodes[$q]->[1],12) ==
$Nodes[$q]->[0])
     {
      # @Sys=grep{hash($_,$Nodes[$q]->[1],12) == $Nodes[$q]-
>[0]}@out1;#print"Sys @Sys\n";push(@s,@Sys);
       push(@s,$DistArr[$Nodes[$q]->[3]]->[$IPcnt]);#print"Entered\n"
     }
   }
 }
}
print"HELLO @s\n";
```

```
@Suspect=();@Cnt=();
for($t=0;$t<=$#s;$t++)
{
  @Cnt=grep{$_ eq $s[$t]}@s;
   if($#Cnt>=2)
    {
      if((grep{$_ eq $Cnt[0]}@Suspect) eq 0)
      {
       push(@Suspect,$Cnt[0]);
      }
    }
}
print"Suspect @Suspect\n";
**************************************************************************

print"Calculating the number of False Positives\n";
for($l=0;$l<=$#ActIP;$l++)
{
 push(@Pre,(grep{$_ eq $ActIP[$l]}@Suspect));
}

$end=time();
$Time=$end-$start;
print"\nAttackers in the suspected set present in actual attack graph\n @Pre
- $#Pre\n";
print"\nActual Attack graph\n @ActIP - $#ActIP\n";
$Fn=$#ActIP-$#Pre;print "\nFn=$#ActIP-$#Pre\n";
$Fp=$#Suspect+$Fn-$#ActIP;print"Fp=$#Suspect+$Fn-$#ActIP\n";
 print "\nFalse Negatives $Fn\nFalse Positives $Fp";
print"\nRT = $Time";
}
```

VITA

The author, Kishori Nanduri was born in Hyderabad, Andhra Pradesh, India. She graduated from Ratna Junior College, Hyderabad, in 1998.

In May 1998, the author attended Jawaharlal Nehru Technological University, Hyderabad, India, with a major in electrical and electronics engineering. She obtained a Bachelor of Technology degree in electrical and electronics engineering in 2002. She continued her graduate study in electrical and computer engineering at Louisiana State University where she is currently a candidate for the degree of Master of Science in Electrical Engineering.