RV COLLEGE OF ENGINEERING®

Bengaluru - 560059

(Autonomous Institution Affiliated to
Visvesvaraya Technological University, Belagavi)

A Minor Project Report On

"IMPLEMENTATION OF A DEFENSE MECHANISM FOR SYN FLOOD ATTACK"

Submitted by

Akash Hegde
1RV18SCN01
Krithika L
1RV18SCN08

Under the Guidance of

Dr. Sowmyarani C. N
Associate Professor

In partial fulfilment for the award of degree of
Master of Technology
in
Computer Network Engineering
Department of Computer Science and Engineering
2019

RV COLLEGE OF ENGINEERING®

Bengaluru - 560059

(Autonomous Institution Affiliated to Visvesvaraya Technological University, Belagavi)

CERTIFICATE

Certified that the minor project work titled "Implementation of a Defense Mechanism for SYN Flood Attack" was carried out by Akash Hegde – 1RV18SCN01 and Krithika L – 1RV18SCN08 who are bonafide students of RV College of Engineering, Bengaluru, in partial fulfillment for the award of degree of Master of Technology in Computer Network Engineering of Visvesvaraya Technological University, Belagavi during the year 2019. It is certified that all corrections indicate during the internal assessment have been incorporated in the report deposited in the departmental library. The minor project report has been approved as it satisfies the academic requirements in respect of the work prescribed by the institution for the said degree.

Dr. Sowmyarani C. N Associate Professor Department of CSE	Dr. Ramakanth Kumar P Head of Department Department of CSE	Dr. K.N. Subramanya Principal R V College of Engg

EXTERNAL VIVA

Name of the Examiner	Signature with Date
1. _____	_____
2. _____	_____

Rashtreeya Sikshana Samithi Trust (RSST)

RV COLLEGE OF ENGINEERING®,

(Autonomous Institution affiliated to VTU, Belagavi)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Bengaluru– 560059

DECLARATION

We, Akash Hegde and Krithika L, students of second semester M.Tech, in the Department of Computer Science and Engineering, R.V. College of Engineering, Bengaluru declare that the minor project entitled "Implementation of a Defense Mechanism for SYN Flood Attack" has been carried out by us and submitted in partial fulfillment of the course requirements for the award of degree in Master of Technology in Computer Network Engineering of RV College of Engineering, Bengaluru affiliated to Visvesvaraya Technological University, Belagavi during the academic year 2018-19. The matter embodied in this report has not been submitted to any other university or institution for the award of any other degree or diploma.

Date of Submission: **Signature of the Student**

Student Name : Akash Hegde
USN: 1RV18SCN01
Student Name : Krithika L
USN: 1RV18SCN08
Department of Computer Science and Engineering,
R.V. College of Engineering,
Bengaluru-560059

ACKNOWLEDGEMENT

Any achievement, be it scholastic or otherwise does not depend solely on the individual efforts but on the guidance, encouragement and cooperation of intellectuals, elders and friends. A number of personalities, in their own capacities have helped us in carrying out this Minor Project work. We would like to take this opportunity to thank them all.

First and foremost we would like to express our sincere gratitude to our guide **Dr. Sowmyarani C. N,** Associate Professor, Department of CSE, R.V.C.E, Bengaluru, for her able guidance, regular source of encouragement and assistance throughout this project

We would like to thank **Dr. Rajashree Shettar,** Associate Dean, Department of Computer Science & Engineering, R.V.C.E, Bengaluru, for her valuable inputs and guidance.

We would like to thank **Dr. Ramakanth Kumar P**, Head of Department, Department of Computer Science & Engineering, R.V.C.E, Bengaluru, for his valuable suggestions and expert advice.

We extend our cordial regards and thanks to our principal **Dr. K. N. Subramanya**. We also thank all the members for this Minor Project who have contributed directly or indirectly.

We thank my parents, and all the faculty members of Department of Computer Science & Engineering for their constant support and encouragement.

Last, but not the least, we would like to thank my peers and friends who provided us with valuable suggestions to improve our Minor Project.

Akash Hegde
1RV18SCN01
Krithika L
1RV18SCN08

ABSTRACT

The TCP SYN flood attack occurs when there is a continuous flow of SYN packets from a source to a destination, which renders the destination node unable to serve any other clients. A detailed study of the TCP SYN flood attack is made and its distributed variant, distributed denial of service (DDoS), where in there exist multiple attackers and a single target.

A brief study is made on iptables rules and the procedure to set up these iptables rules and configure them. A multi-machine set up is made for the demonstration of the SYN flood attack, where in the source attacker is used to flood the destination target. It is observed that the iptables rules help in effective rejection of the SYN packets from the flooding. The TCP connection is closed immediately and a RST+ACK response is sent to the attacker machine. It is also observed that iptables rules help in easier filtering of the packets, by checking the source IP addresses.

A comparative analysis is made between the SYN flood attack and the ICMP flood attack, with respect to the total attack time and the average round trip time for varying number of flooding packets. It is observed that the SYN flood attack occurs faster than the ICMP flood attack and also has significantly lesser average round trip time than that of the ICMP flood attack.

LIST OF TABLES

Table No.	Table Name	Page No.
8.1	Total attack time for flooding of 100 packets	42
8.2	Total attack time for flooding of 200 packets	42
8.3	Total attack time for flooding of 300 packets	43
8.4	Average round trip time for flooding of 100 packets	45
8.5	Average round trip time for flooding of 200 packets	46
8.6	Average round trip time for flooding of 300 packets	46

LIST OF FIGURES

Figure No.	Figure Name	Page No.
1.1	Distributed variant of denial-of-service (DoS) attack	2
1.2	TCP SYN flood attack	3
1.3	UDP flood attack	4
1.4	ICMP flood attack	5
1.5	Ping of Death attack	5
1.6	Peer-to-peer attack	6
3.1	TCP header	11
3.2	TCP state transmission diagram	12
3.3	SYN flooding	13
3.4	SYN flood – spoofing attack	14
3.5	SYN flood – distributed attack	15
5.1	Structure chart for implementing SYN flood defense mechanism	28
8.1	Output of normal TCP 3-way handshake	38
8.2	Logged output for normal TCP 3-way handshake	38
8.3	Summary of the transmitted packets	39
8.4	RST+ACK received upon flooding	40
8.5	DDoS implementation of SYN flood	40
8.6	DDoS implementation of ICMP flood	41
8.7	Graph of total attack time for flooding of 100 packets	43
8.8	Graph of total attack time for flooding of 200 packets	44
8.9	Graph of total attack time for flooding of 300 packets	44
8.10	Graph of average round trip time for flooding of 100 packets	47
8.11	Graph of average round trip time for flooding of 200 packets	47
8.12	Graph of average round trip time for flooding of 300 packets	48

Table of Contents

Certificate	(i)
Declaration	(ii)
Acknowledgement	(iii)
Abstract	(iv)
List of Tables	(v)
List of Figures	(vi)
Chapter 1 **Introduction**	**1**
1.1 Denial of Service (DoS) Attack	1
1.2 Types of DoS Attacks	2
1.2.1 TCP SYN Flood Attack	2
1.2.2 UDP Flood Attack	3
1.2.3 ICMP Flood Attack	4
1.2.4 Ping of Death Attack	5
1.2.5 Teardrop Attack	6
1.2.6 Peer-to-Peer Attack	6
1.3 Problem Statement	7
1.4 Objectives	7
Chapter 2 **Literature Survey**	**8**
Chapter 3 **Overview of the SYN Flood Attack**	**13**
3.1 Attack Mechanisms	13
3.2 Defense Mechanisms	15
3.2.1 SYN Cache	16
3.2.2 SYN Cookies	16
3.2.3 Hybrid Approach	16
3.2.4 Reduce the SYN-RECEIVED time	16
3.2.5 Filtering	17
3.2.6 Firewall Approach	17
3.2.7 Active Monitoring	18
Chapter 4 **Requirement Specifications**	**19**
4.1 Software Requirements	19
4.1.1 Wireshark	19
4.1.2 Scapy	20
4.1.3 iptables	21
4.1.4 Python	25
4.1.5 Shell Script	25
4.2 Hardware Requirements	26

Chapter 5	
Design	**27**
5.1 Structure Chart	27
5.2 Functional Description of the Modules	28
5.2.1 Implement a Defense Mechanism for SYN Flood	28
5.2.2 Set up the Working Environment	29
5.2.3 Execute SYN Flood Attack	29
5.2.4 Execute Defense Mechanism	30
Chapter 6	
Implementation	**31**
6.1 Programming Language Selection	31
6.2 Platform Selection	32
6.3 Code Conventions	33
6.3.1 Naming Conventions	33
6.3.2 File Organization	33
6.3.3 Comments	33
Chapter 7	
Software Testing	**34**
7.1 Test Environment	34
7.2 Unit Testing	34
7.2.1 Unit Testing of Setting Up the Environment	34
7.2.2 Unit Testing of SYN Flood Attack	35
7.2.3 Unit Testing of the Defense Mechanism	35
7.3 Integration Testing	35
7.4 System Testing	36
Chapter 8	
Experimental Analysis and Results	**37**
8.1 Observations and Results	37
8.2 Comparative Analysis with ICMP Flood	41
8.3 Inference from the Results	48
Chapter 9	
Conclusion	**49**
References	**50**

Chapter 1

Introduction

Denial of Service (DoS) attacks have become very common amongst hackers who use them as a path to fame and respect in the underground groups of the Internet. Denial of Service attacks basically means denying valid Internet and network users from using the services of the target network or server. It basically means, launching an attack, which will temporarily make the services, offered by the network unusable by legitimate users.

In others words, one can describe a DoS attack as an attack which clogs up so much memory on the target system that it cannot serve legitimate users, or if the target system is sent some data packets, it cannot be handled by the system and thus causes it to either crash, reboot or more commonly deny services to legitimate users.

DoS Attacks are of the following different types –

- Those that exploit vulnerabilities in the TCP/IP protocols suite.
- Those that exploit vulnerabilities in the IPv4 implementation.
- There are also some brute force attacks, which try to use up all resources of the target system and make the services unusable.

1.1 Denial of Service (DoS) Attack

The United States Computer Emergency Response Team defines symptoms of denial-of-service attacks to include:

- Unusually slow network performance (opening files or accessing web sites).
- Unavailability of a particular web site.
- Inability to access any web site.
- Dramatic increase in the number of spam emails received—(this type of DoS attack is considered an e-mail bomb).

Denial-of-service attacks can also lead to problems in the network 'branches' around the actual computer being attacked. For example, the bandwidth of a router between the Internet and a LAN may be consumed by an attack, compromising not only the intended computer, but also the entire network. If the attack is conducted on a sufficiently large scale, entire geographical regions of Internet connectivity can be compromised without the attacker's knowledge or intent by incorrectly configured or flimsy network infrastructure equipment. Figure 1.1 shows a distributed variant of a DoS attack, commonly referred to as a distributed DoS attack (DDoS).

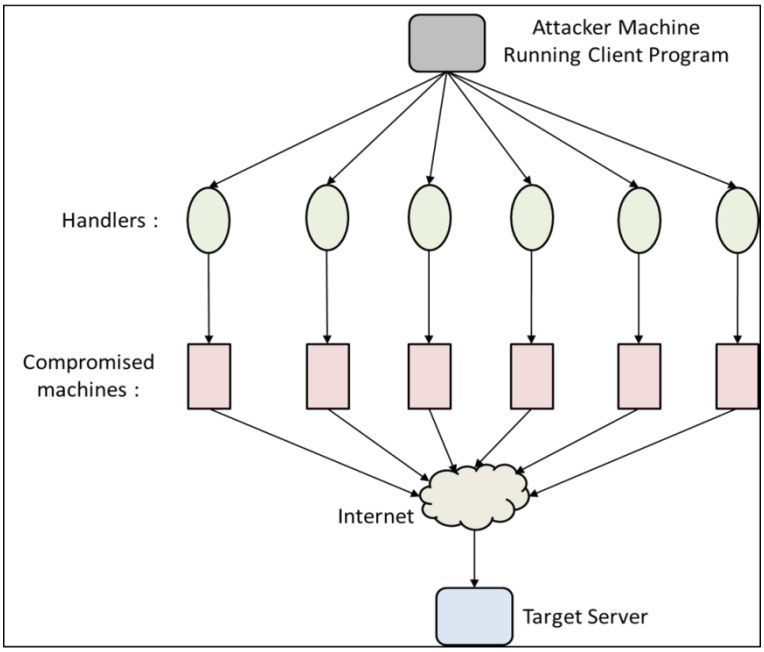


Figure 1.1 Distributed variant of denial-of-service (DoS) attack

1.2 Types of DoS Attacks

There are several types of DoS attacks that can occur in the network. Some of these DoS attacks are described in the following subsections.

1.2.1 TCP SYN Flood Attack

The TCP SYN flood attack uses the three-way handshake protocol for the attack purpose. In order to establish a connection with the destination, initially SYN packets are sent. After the SYN-ACK is received, the destination will not send any responses

further. In order to receive the data from the source, a connection queue is used at the destination host.

As a matter of fact, there will be no further responses from the destination and the queue will have to be maintained all the time, which results in the wastage of the compute resources and the inability to service the legitimate requests. Figure 1.2 shows how a TCP SYN flood attack usually occurs.

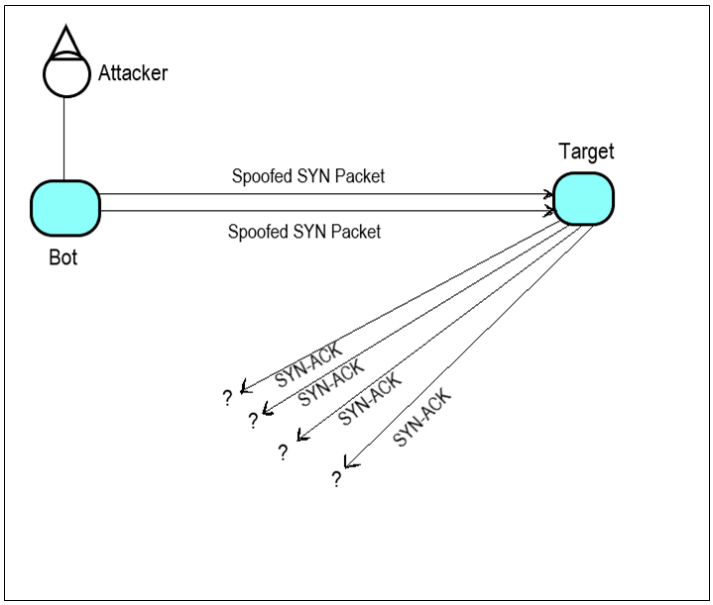


Figure 1.2 TCP SYN flood attack

1.2.2 UDP Flood Attack

A UDP flood attack is caused when the attacker will send a large amount of UDP packets to the victim system port in a random fashion such that the target will receive the UDP packets and will confirm the application that is waiting at the destination port to be sent.

When the absence of existence of such ports at the source is found out, then the ICMP packets are generated in order to reach the original source. When a large amount of such UDP packets are received, the target will crash and the attack

will become successful. Figure 1.3 depicts a general scenario of how a UDP flood attack occurs.

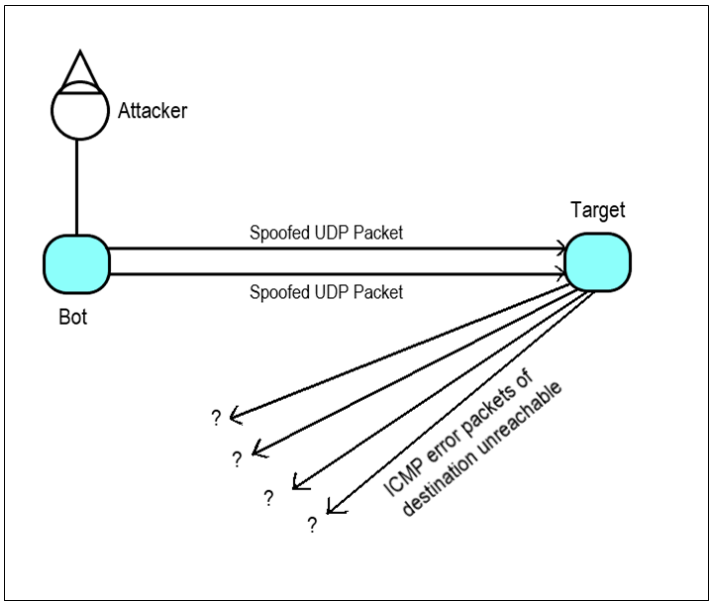


Figure 1.3 UDP flood attack

1.2.3 ICMP Flood Attack

In this attack, initially the router that responds to the ICMP requests is found out by the attacker. Knowing this router, the attacker will send the requests to the router's broadcast address that contains the spoofed source IP address.

When the message is broadcasted to all the devices in the network by the router, they send back their responses immediately. This will cause the generation of a large amount of traffic making the bandwidth to choke within the network. Figure 1.4 depicts how an ICMP flood occurs.

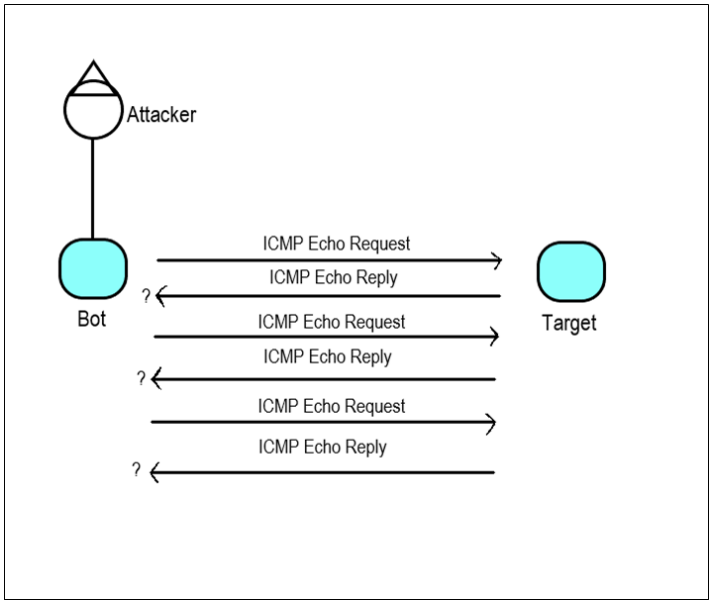


Figure 1.4 ICMP flood attack

1.2.4 Ping of Death Attack

A Ping of Death attack occurs when the attacker sends a large number of ping requests, or ICMP echo requests to a target; these requests contain a large or illegal packet size, which makes the target crash or continuously respond with ICMP echo replies to the attacker. This forces other clients to wait indefinitely. Figure 1.5 shows how a Ping of Death attack occurs.

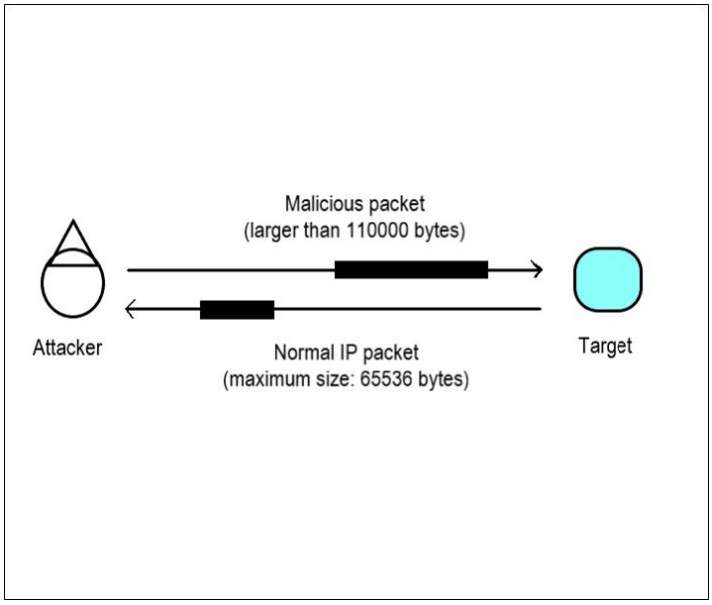


Figure 1.5 Ping of Death attack

1.2.5 Teardrop Attack

In a teardrop attack, the manipulated IP fragments that contain the overlapped fragments as well as oversized payloads, are sent by the attacker to the target. As a result of this, the target system may crash and lose huge amount of data. This kind of attack will affect almost all the operating systems and the all types of servers.

1.2.6 Peer-to-Peer Attack

In a peer-to-peer (P2P) attack, the target system is injected with the useless data. This can be called as poisoning the network. It is for the attacker to inject a huge amount of bogus look-up key-value pairs into the index of the target system as all the P2P networks use a look-up service; this may cause the target system to become slow, will introduce a delay in producing the query results and also may produce invalid results. Figure 1.6 shows how a P2P poisoning occurs. There exist multiple peers in the network and the target is poisoned by the attacker through these peers.

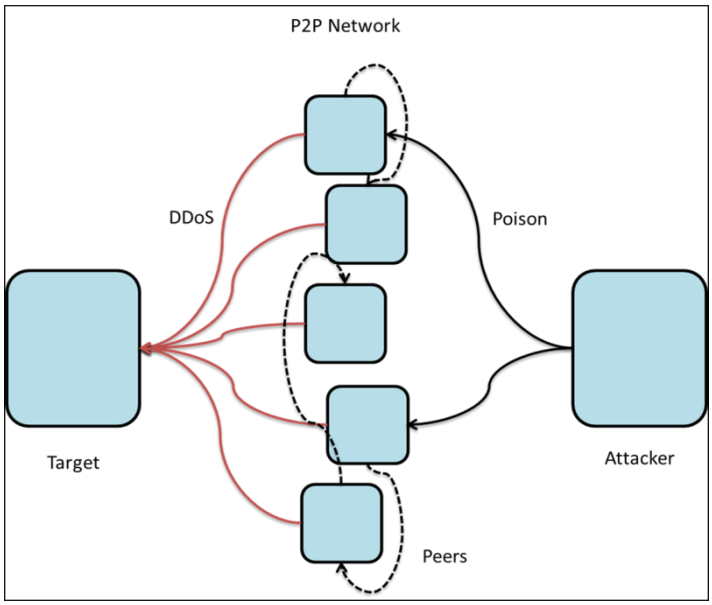


Figure 1.6 Peer-to-peer attack

1.3 Problem Statement

The scope of this report is to explain the concept of SYN Flood attack generation and detection by using Scapy and Wireshark respectively. Suspicious traffic is denied by using iptables rules.

1.4 Objectives

A SYN queue flood attack takes advantage of the TCP protocol's "three-way handshake". A client sends a TCP SYN (S flag) packet to begin a connection to the server. The target server replies with a TCP SYN-ACK (SA flag) packet, but the client does not respond to the SYN-ACK, leaving the TCP connection "half-open". In normal operation, the client should send an ACK (a flag) packet followed by the data to be transferred, or an RST reply to reset the connection. On the target server, the connection is kept open, in a "SYN_RECV" state, as the ACK packet may have been lost due to network problems.

In a DDoS, multiple attackers make many such half-connections to the target server, in a storm of requests. When the server's SYN buffer is full with half-open TCP connections, it stops accepting SYN connections, thus resulting in denial of service to legitimate clients. Such DDoS attacks are generally carried out using "botnets" of other compromised systems across the Internet, which through backdoors and Trojans are directed to send artificial SYN flood traffic to targeted servers. To defend against such attacks, a strong monitoring system is required, as there is a very fine line between legitimate and fake clients. SYN queue flood attacks can be mitigated by tuning the kernel's TCP/IP parameters**.**

The main goal is to provide a better solution after identifying such malicious IP addresses, increase the amount of time between SYN received and SYN+ACK sent by the server to the client. This will make the attacker feel that the server has come under attack and is losing its performance capabilities; whereas this will not be the case as the server will be catering the needs of other genuine clients at a normal rate.

Chapter 2

Literature Survey

In the literature there are many methods and frameworks which are proposed in order to detect the SYN flood attacks.

The authors of [1] have designed and implemented an OpenFlow-based mitigation system for TCP SYN flood attack. The OpenFlow mechanism built in to the network infrastructure achieves centralized and flexible network management by decoupling the data plane and the control plane. The packet behaviour is controlled by a controller software which uses header information of Layer 1 to Layer 4 to distinguish the packets. In [2], the researchers have used Bloom Filter to detect spoofed packets in SYN flood attack. It works on the principle of multiple layers of the filter to identify the anomalies of spoofed packet identification.

The work of [3] discusses about the Efficient Spoofed Mitigation Scheme (ESMS) to detect and control spoofed packets during DDoS attacks. It uses TCP control probing and a Bloom Filter trust model in order to provide accurate and robust information. In [4], efficient packet filtering technique is explained using firewall in order to defend against TCP SYN flood attacks. The method uses iptables to write firewall scripts to deny the suspicious traffic. Similar work done in [5] discusses about the effective mitigation of both DoS and DDoS attacks using iptables rules.

The authors of [6] detected the SYN flooding attacks at leaf routers that connect end hosts to the Internet, which utilizes the normalized difference between the number of SYNs packets and the number of FIN (RST) packets in a time interval. If the rate of SYNs packets is much higher than that of FIN (RST) packets by a non-parametric cumulative sum algorithm, the router recognizes that some attacking traffic is mixed into the current traffic.

In the work [7] an early stage detecting method (ESDM) is proposed. The ESDM is a simple but effective method to detect SYN flooding attacks at the early stage. In the ESDM the SYN traffic is forecasted by autoregressive integrated moving average model, and non-parametric cumulative sum algorithm is used to find the SYN flooding attacks according to the forecasted traffic. The ESDM achieves shorter detection time and small storage space. However, these existing methods or defense mechanisms which oppose to the SYN flooding attack are effective only at the later stages, when attacking signatures are obvious.

Similar work is presented in [8], where the fast and effective method for detecting SYN flood attacks is given. Moreover, a linear prediction analysis is proposed as a new paradigm for DoS SYN flood attack detection. The proposed mechanism makes use of the exponential back off property of TCP used during timeouts. By modelling the difference of SYN and SYN & ACK packets, it is shown that this approach is able to detect an attack within short delays. Again this method is used at leaf routers and firewalls to detect the attack without the need of maintaining any state. However, considering the fact that the sources of attack can be distributed in different networks, there is a lack of analysis for the traffic near the sources and also the detection of the source of SYN flooding attack in TCP based low intensity attacks is missing.

Moreover, a quite similar approach was used in [9], which also considers a non-parametric cumulative sum algorithm; however the authors apply it to measure the number of only SYN packets, and by using an exponential weighted moving average for obtaining a recent estimate of the mean rate after the change of SYN packets.

In [10] three counters algorithms for SYN flooding defense attacks are given. The three schemes include detection and mitigation. The detection scheme utilizes the inherent TCP valid SYN– FIN pairs behavior, hence is capable of detecting various SYN flooding attacks with high accuracy and short response time. The mitigation scheme works in high reliable manner for victim to detect the SYN packets of SYN flooding attack. Although the given schemes are stateless and

required low computation overhead, making itself immune to SYN flooding attacks, the attackers may retransmit every SYN packet more than one time to destroy the mitigation function. It is necessary to make it more robust and adaptive.

In [11], the authors built a standard model generated by observations from the characteristic between the SYN packet and the SYN+ACK response packet from the server by a program for the activity of the server. The authors of [12] proposed a method to detect the flooding agents by considering all the possible kinds of IP spoofing, which is based on the SYN/SYN-ACK protocol pair with the consideration of packet header information. The Counting Bloom Filter is used to classify all the incoming SYN-ACK packets to the sub network into two streams, and a nonparametric cumulative sum algorithm is applied to make the detection decision by the two normalized differences, with one difference between the number of SYN packets and the number of the first SYN-ACK packets and another difference between the number of the first SYN-ACK packets and the number of the retransmission SYN-ACK.

Moreover, in [13] a simple and efficient method to detect and defend against SYN flood attacks under different IP spoofing types is proposed. The method makes use of a storage-efficient data structure and a change-point detection method for distinguishing complete three-way TCP handshakes from incomplete ones. The presented experiments consistently show that their method is both efficient and effective in defending against TCP-based flooding attacks under different IP spoofing types. However there is a lack of process automation within the scheme for setting the parameters. Additionally, the method is not evaluated in a reasonably large real network.

Chapter 3

Overview of the SYN Flood Attack

The Transmission Control Protocol (TCP) is connection-oriented and reliable, in-sequence delivery transport protocol. It provides full duplex stream of data octets and it is the main protocol for the Internet. Most services nowadays on Internet rely on TCP. For example, mail (SMTP, port 25), old insecure virtual terminal service (telnet, port 23), file transport protocol (FTP, port 21) and most important in this case also is the Hyper Text Transfer Protocol (HTTP, 80) better known as the World Wide Web services (WWW). Almost everything uses TCP in some way to do their communications over the network – at least the interactive ones.

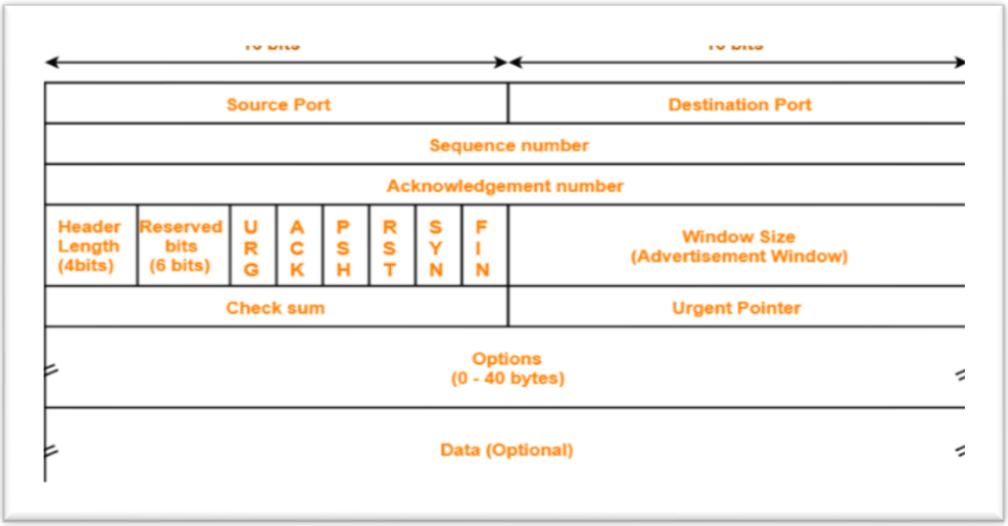


Figure 3.1 TCP header

In SYN flood attack, the "SYN" stands for the Synchronize flag in TCP headers. The SYN flag gets set when a system first sends a packet in a TCP connection and indicates that the receiving system should store the sequence number included in this packet. Figure 3.1 shows the TCP header, which is twenty bytes long without options. In this illustration each line represents four bytes. In this kind of flooding attack, the focus is given on the Flags, six different bits that may be sent to represent different conditions, such as initial sequence number (SYN), that

the acknowledgement field is valid (ACK), reset the connection (RST), or close the connection (FIN).

As it was mentioned before, in SYN flooding [3], the attacking system sends SYN request with spoofed source IP address to the victim host. These SYN requests appear to be legitimate. The spoofed address refers to a client system that does not exist. Hence, the final ACK message will never be sent to the victim server system. This results into increased number of half-open connections at the victim side. A backlog queue is used to store these half- open connections. These half-open connections bind the resources of the server. Hence, no new connections (legitimate) can be made, resulting in DoS or DDoS. SYN flooding attack is a DoS method affecting hosts that run TCP server processes. The TCP state transmission diagram is depicted in Figure 3.2.

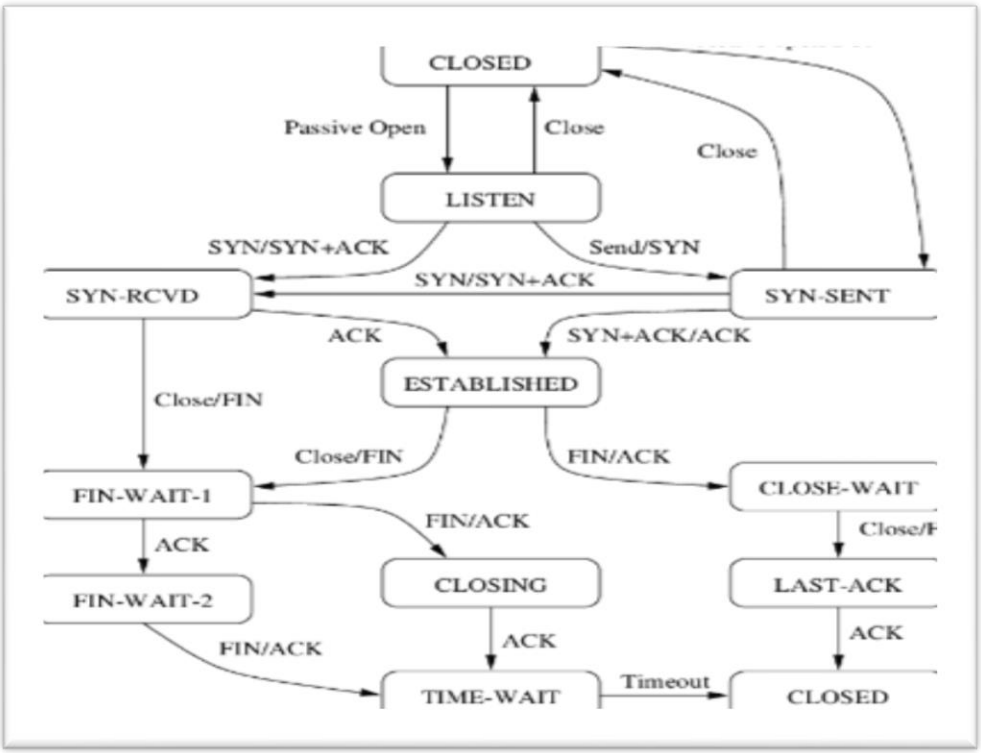


Figure 3.2 TCP state transmission diagram

3.1 Attack Mechanisms

The SYN flood attack is well-known DoS method which affects hosts that run TCP server processes (the three-way handshake mechanism of TCP connection). Nowadays, despite the original one, a lot of its variations can be seen. Although there are many effective techniques against SYN flood attack, and even RFC4987 is covering some common mitigation techniques against this attack, yet there is no single mechanism (schemes) for effective defense.

During the SYN flood attack, the attacking system sends SYN request with spoofed source IP address to the victim host. These SYN requests appear to be legitimate. The spoofed address refers to a client system that does not exist. Hence, the final ACK message will never be sent to the victim server system. This results into increased number of half-open connections at the victim side. A backlog queue is used to store these half-open connections. These half-open connections bind the resources of the server. Hence, no new connections (legitimate) can be made, resulting in DoS or DDoS. Generally in the literature, there are three types of SYN flood attacks, which are going out in the nowadays Internet networks: Direct Attack, Spoofing Attack and Distributed Direct Attack.

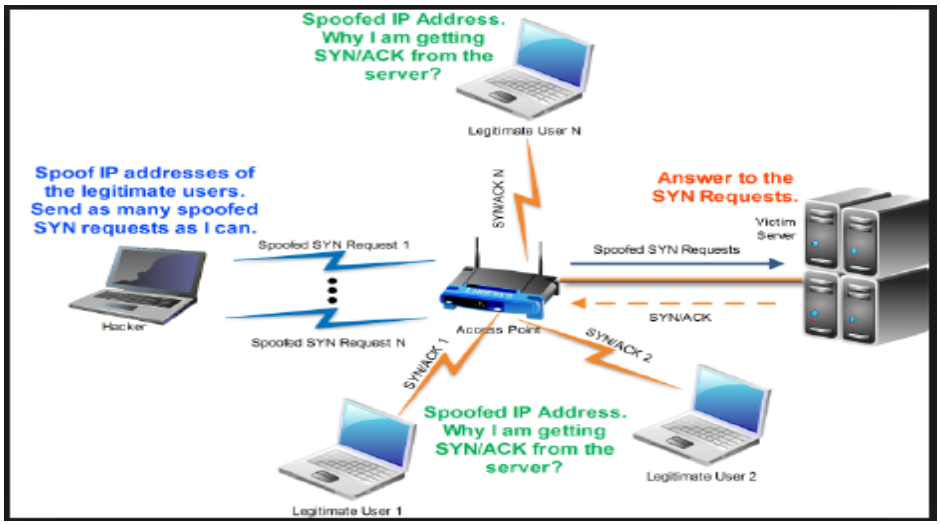


Figure 3.3 SYN flooding

If attackers rapidly send SYN segments without spoofing their IP source address, this will cause direct attack, as shown in Figure 3.3. This type of SYN flooding attack does not involve directly injecting or spoofing packets below the user level of the operating system of the attacker. One way to perform third type of attack is by simply using many TCP connect() calls. However, the attacker's operation system must not respond to the SYN-ACKs, because any ACKs, RSTs, or ICMP (Internet Control Message Protocol) messages will allow the listener to move the TCB (Transmission Control Block) out of SYN-RECEIVED. In order to prevent its operating system from responding to the SYN-ACKs, the attacker can set some firewall rules which can filter outgoing packets to the listener (allowing only SYNs out), or filter incoming packets so that any SYN-ACKs are discarded before reaching the local TCP processing code.

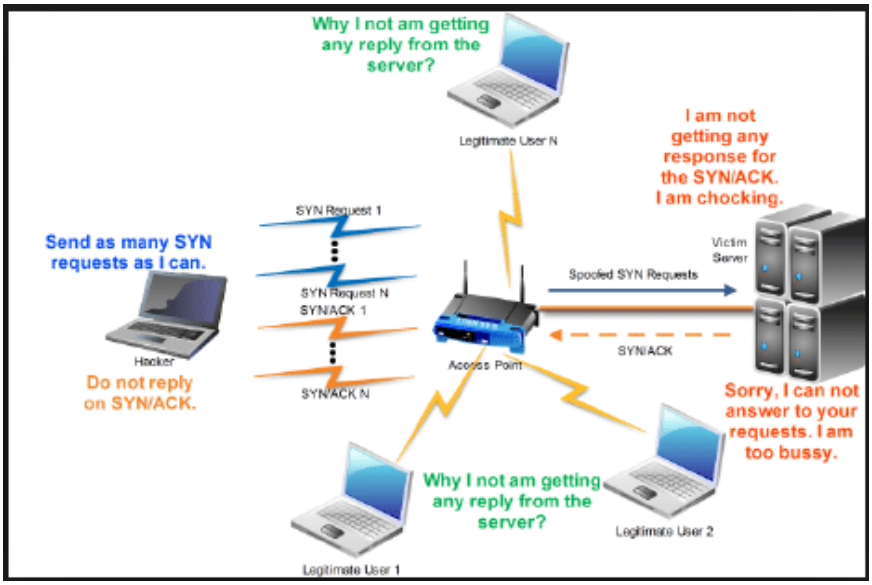


Figure 3.4 SYN flood – spoofing attack

On the other hand, the SYN spoofing attack, as shown in Figure 3.4, uses IP address spoofing, which might be considered more complex than the method used in a direct attack. During this type of SYN flooding attack instead of merely manipulating local firewall rules, the attacker also needs to be able to form and inject raw IP packets with valid IP and TCP headers. Moreover, the IP address spoofing techniques can be categorized into different types according to what spoofed source addresses are used in the attacking packets.

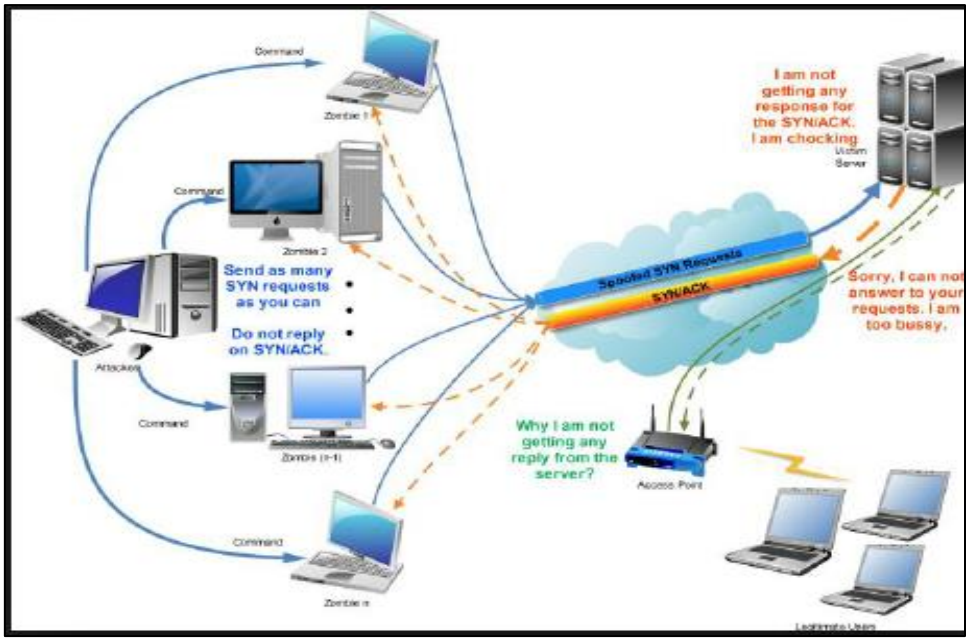


Figure 3.5 SYN flood – distributed attack

A distributed version of the SYN flooding attack shown in Figure 3.5 is the most dangerous amongst mentioned types of SYN flooding attacks. During this type of SYN flooding attack the attacker takes advantage of numerous zombie machines/processes throughout the Internet. In the case, the zombies use direct attacks, but in order to increase the effectiveness even further, each zombie could use a spoofing attack and multiple spoofed IP addresses.

Currently, distributed attacks are feasible because there are several "botnets" or "zombie armies" of thousands of compromised machines that are used by criminals for DoS attacks. Because zombie machines are constantly added or removed from the armies and can change their IP addresses or connectivity, it is quite challenging to block those types of SYN flood attacks.

3.2 Defense Mechanisms

There are several defense mechanisms to combat the SYN flood attacks. Some of these mechanisms are detailed in the following subsections.

3.2.1 SYN Cache

SYN caching allocates some state on the machine, but even with this reduced state it is possible to encounter resource exhaustion. The code must be prepared to handle state overflows and choose which items to drop in order to preserve fairness. The initial SYN request carries a collection of options which apply the TCP connection; these commonly include the desired message segment size, requested window scaling for the connection, use of timestamps, and various other items. Part of the purpose of the allocated state is to record these options, which are not retransmitted in the return ACK from the client.

3.2.2 SYN Cookies

SYN cookies do not store any state on the machine, but keeps all state regarding the initial TCP connection in the network, treating it as an infinitely deep queue.

3.2.3 Hybrid Approach

This is the mechanism to combine any two or more than two defense mechanisms. For example if we combine large backlog and SYN cookies then it will be more robust than individual mechanism.

3.2.4 Reduce the SYN-RECEIVED Time

To defend against the exhaustion of resources in the systems under attack, an obvious approach is to increase the number of resources devoted to half-open TCP connections, and to reduce the timeouts. These measures can be summarized as:

- Reduce the timeout period from the default to a short time
- Significantly increase the length of the backlog queue from the default
- Disable non-essential services, thus reducing the number of ports that can be attacked

3.2.5 Filtering

The measures proposed in the first reactions to the recent attacks attempt to make it difficult for packets with spoofed source addresses to traverse routers. The solutions proposed can be summarized as follows:

- Configure external interfaces on routers to block packets that have source addresses from the internal network.
- Configure internal router interfaces to block packets to the outside that have source addresses from outside the internal network.

3.2.6 Firewall Approach

Most of the websites today are protected by the firewalls, hence to protect against SYN flooding attack firewall can be a very useful tool. Several firewall vendors have already made products available to increase protection against the attacks, and some other solutions have been proposed. Firewall-based protection approaches are based on the idea that every packet destined to a host inside the firewall has to be examined by the firewall first, and thus decisions can be made on its authenticity and actions can be taken to protect the internal hosts. There are two approaches –

- Firewall as a Relay – in this approach, when a packet for an internal host is received the firewall answers on its behalf. Only after the three-way handshake is successfully completed does the firewall contact the host and establish a second connection.
- Firewall as a Semi-transparent Gateway – in this approach, the firewall lets SYN and ACK packets go through, but monitors the traffic and reacts to it. The firewall passes SYN packets destined to internal hosts. When the host responds with a SYN+ACK packet, the firewall forwards it, but reacts by generating and sending an ACK. packet that seems to come from the client.

3.2.7 Active Monitoring

This category of solutions consists of using software agents to continuously monitor TCP/IP traffic in a network at a given place. An agent can collect communication control information to generate a view of all connections that can be observed on a monitored network. Furthermore, it can watch for certain conditions to arise and react appropriately.

The above mentioned mechanism can be classified into two broad classes on the basis of where the defenses are implemented.

- End Host Mechanism: This involves hardening the end host TCP implementation itself, including altering algorithms and data structures used for connection lookup and establishment, as well as some solutions that diverge from the TCP state machine behavior during connection establishment.
- Network based Mechanism: This category of mechanism involves hardening the network, either to lessen the likelihood of the attack preconditions (an army of controlled hosts or the propagation of IP packets with spoofed source addresses), or to insert middle boxes that can isolate servers on the networks behind them from illegitimate SYNs.

Chapter 4

Requirement Specifications

This section describes the requirement specifications for the accomplishment of the SYN flood attack and the corresponding defense mechanisms. This includes the software and the hardware requirements, which are detailed in the following subsections.

4.1 Software Requirements

There are several software that are used in the implementation of the SYN flood attack and analysis. These are detailed in the following subsections.

4.1.1 Wireshark

Wireshark is a network protocol analyzer. This multiplatform application comes bundled with a GUI to make network troubleshooting and analysis easy to work with and view in real time. It is most often used for its packet sniffing capabilities that allow users to capture and view packets in real time across a multitude of network protocols.

Wireshark lets the user put network interface controllers into promiscuous mode (if supported by the network interface controller), so they can see all the traffic visible on that interface including unicast traffic not sent to that network interface controller's MAC address. However, when capturing with a packet analyzer in promiscuous mode on a port on a network switch, not all traffic through the switch is necessarily sent to the port where the capture is done, so capturing in promiscuous mode is not necessarily sufficient to see all network traffic. Port mirroring or various network taps extend capture to any point on the network. Simple passive taps are extremely resistant to tampering.

Features of Wireshark include:

- Data is analyzed either from the wire over the network connection or from data files that have already captured data packets.
- Supports live data reading and analysis for a wide range of networks (including Ethernet, IEEE 802.11, point-to-point Protocol (PPP) and loopback).
- With the help of GUI or other versions, users can browse captured data networks.
- For programmatically editing and converting the captured files to the editcap application, users can use command line switches.
- Display filters are used to filter and organize the data display.
- New protocols can be scrutinized by creating plug-ins.
- Captured traffic can also trace Voice over Internet (VoIP) calls over the network.
- When using Linux, it is also possible to capture raw USB traffic.

4.1.2 Scapy

Scapy is a powerful interactive packet manipulation program. It is able to forge or decode packets of a wide number of protocols, send them on a wire, capture them, match requests and replies, and much more. It can easily handle most classical tasks like scanning, trace routing, probing, unit tests, attacks on network discovery. It also performs other specific tasks that other tools cannot handle.

Scapy also performs very well on a lot of other specific tasks that most other tools can’t handle, like sending invalid frames, injecting your own 802.11 frames, combining techniques (VLAN hopping+ARP cache poisoning, VOIP decoding on WEP encrypted channel), etc.

Scapy mainly does two things: sending packets and receiving answers. You define a set of packets, it sends them, receives answers, matches requests with answers and returns a list of packet couples (request, answer) and a list of unmatched packets. This has the big advantage over tools like Nmap or hping that an answer is not reduced to (open/closed/filtered), but is the whole packet.

Scapy enables the user to describe a packet or set of packets as layers that are stacked one upon another. Fields of each layer have useful default values that can be overloaded. Scapy does not oblige the user to use predetermined methods or templates. This alleviates the requirement of writing a new tool each time a different scenario is required. In C, it may take an average of 60 lines to describe a packet. With Scapy, the packets to be sent may be described in only a single line with another line to print the result. 90% of the network probing tools can be rewritten in 2 lines of Scapy.

4.1.3 iptables

iptables is a current Linux Firewall mechanism and a successor of ipfilter and ipchains. The primary purpose is packet filtering based on header fields, e.g., IP addresses, TCP and UDP ports, and TCP flags. Originally, the most popular firewall/NAT package running on Linux was ipchains, but it had a number of shortcomings. To rectify this, the Netfilter organization decided to create a new product called iptables.

It is a user-space utility program that allows a system administrator to configure the tables provided by the Linux kernel firewall (implemented as different Netfilter modules) and the chains and rules it stores. Different kernel modules and programs are currently used for different protocols; *iptables* applies to IPv4, *ip6tables* to IPv6, *arptables* to ARP, and *ebtables* to Ethernet frames. The term *iptables* is also commonly used to inclusively refer to the kernel-level components. *x_tables* is the name of the kernel module carrying the shared code portion used by all four modules that also provides the API used for extensions;

subsequently, *Xtables* is more or less used to refer to the entire firewall (v4, v6, arp, and eb) architecture.

There are currently four independent tables (which tables are present at any time depends on the kernel configuration options and which modules are present).

-t, --table *table*

This option specifies the packet matching table which the command should operate on. If the kernel is configured with automatic module loading, an attempt will be made to load the appropriate module for that table if it is not already there.

The tables are as follows:

- **filter**: This is the default table (if no -t option is passed). It contains the built-in chains INPUT (for packets destined to local sockets), FORWARD (for packets being routed through the box), and OUTPUT (for locally-generated packets).
- **nat**: This table is consulted when a packet that creates a new connection is encountered. It consists of three built-ins: PREROUTING (for altering packets as soon as they come in), OUTPUT (for altering locally-generated packets before routing), and POSTROUTING (for altering packets as they are about to go out).
- **mangle**: This table is used for specialized packet alteration. Until kernel 2.4.17 it had two built-in chains: PREROUTING (for altering incoming packets before routing) and OUTPUT (for altering locally-generated packets before routing). Since kernel 2.4.18, three other built-in chains are also supported: INPUT (for packets coming into the box itself), FORWARD (for altering packets being routed through the box), and POSTROUTING (for altering packets as they are about to go out).
- **raw**: This table is used mainly for configuring exemptions from connection tracking in combination with the NOTRACK target. It registers at the netfilter hooks with higher priority and is thus called before ip_conntrack, or any other IP tables. It provides the following built-in chains: PREROUTING (for packets arriving via any network interface) OUTPUT (for packets generated by local processes)

The options that are recognized by *iptables* can be divided into several different groups as follows –

- Commands – these options specify the specific action to perform. Only one of them can be specified on the command line unless otherwise specified below. For all the long versions of the command and option names, you need to use only enough letters to ensure that **iptables** can differentiate it from all other options.
    - **-A, --append** *chain rule-specification*

      Append one or more rules to the end of the selected chain. When the source and/or destination names resolve to more than one address, a rule will be added for each possible address combination.
    - **-D, --delete** *chain rule-specification;* **-D, --delete** *chain rulenum*

      Delete one or more rules from the selected chain. There are two versions of this command: the rule can be specified as a number in the chain (starting at 1 for the first rule) or a rule to match.
    - **-I, --insert** *chain* [*rulenum*] *rule-specification*

      Insert one or more rules in the selected chain as the given rule number. So, if the rule number is 1, the rule or rules are inserted at the head of the chain. This is also the default if no rule number is specified.
    - **-R, --replace** *chain rulenum rule-specification*

      Replace a rule in the selected chain. If the source and/or destination names resolve to multiple addresses, the command will fail. Rules are numbered starting at 1.
    - **-L, --list** [*chain*]

      List all rules in the selected chain. If no chain is selected, all chains are listed. As every other iptables command, it applies to the specified table (filter is the default), so NAT rules get listed by

      iptables -t nat -n –L

      iptables -L –v

- Parameters – the following parameters make up a rule specification (as used in the add, delete, insert, replace and append commands).
    - **-p, --protocol** [*protocol*]

      The protocol of the rule or of the packet to check. The specified protocol can be one of *tcp*, *udp*, *icmp*, or *all*, or it can be a numeric value, representing one of these protocols or a different one.
    - **-s, --source** [*address*]

      Source specification. *Address* can be either a network name, a hostname (please note that specifying any name to be resolved with a remote query such as DNS is a really bad idea), a network IP address (with /mask), or a plain IP address.
    - **-d, --destination** [*address*]

      Destination specification. See the description of the -s (source) flag for a detailed description of the syntax. The flag --dst is an alias for this option.
    - **-j, --jump** *target*

      This specifies the target of the rule; i.e., what to do if the packet matches it**.**

- Other options – the following additional options can be specified:
    - **-v, --verbose**

      Verbose output. This option makes the list command show the interface name, the rule options (if any), and the TOS masks
    - **-n, --numeric**

      Numeric output. IP addresses and port numbers will be printed in numeric format. By default, the program will try to display them as host names, network names, or services (whenever applicable).
    - **--line-numbers**

      When listing rules, add line numbers to the beginning of each rule, corresponding to that rule's position in the chain.

4.1.4 Python

Python is an interpreted, high-level, general purpose programming language. Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library. Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of its features support functional programming and aspect-oriented programming (including by meta-programming and meta-objects (magic methods)). Many other paradigms are supported via extensions, including design by contract and logic programming.

Python uses dynamic typing, and a combination of reference counting and a cycle-detecting garbage collector for memory management. It also features dynamic name resolution (late binding), which binds method and variable names during program execution. Python's design offers some support for functional programming in the Lisp tradition. It has filter, map, and reduce functions; list comprehensions, dictionaries, sets and generator expressions. The standard library has two modules (itertools and functools) that implement functional tools borrowed from Haskell and Standard ML.

4.1.5 Shell Script

A shell script is a computer program designed to be run by the Unix shell, a command-line interpreter. The various dialects of shell scripts are considered to be scripting languages. Typical operations performed by shell scripts include file manipulation, program execution, and printing text. A script which sets up the environment, runs the program, and does any necessary cleanup, logging, etc. is called a wrapper.

The term is also used more generally to mean the automated mode of running an operating system shell; in specific operating systems they are called other things such as batch files (MSDos-Win95 stream, OS/2), command procedures (VMS), and shell scripts (Windows NT stream and third-party derivatives like 4NT—article is at cmd.exe), and mainframe operating systems are associated with a number of terms.

The typical Unix/Linux/POSIX-compliant installation includes the KornShell (ksh) in several possible versions such as ksh88, Korn Shell '93 and others. The oldest shell still in common use is the Bourne shell (sh); Unix systems invariably also include the C shell (csh), Bash (bash), a Remote Shell (rsh), a Secure Shell (ssh) for SSL telnet connections, and a shell which is a main component of the Tcl/Tk installation usually called tclsh; which is a GUI-based Tcl/Tk shell. The C and Tcl shells have syntax quite similar to that of said programming languages, and the Korn shells and Bash are developments of the Bourne shell, which is based on the ALGOL language.

4.2 Hardware Requirements

A hardware configuration with the capability to run multiple virtual machines is required. It should also be capable of running the operating system and the necessary software for the SYN flood attack and defense mechanisms.

Some specifications for the hardware is given by –

- CPU – 64-bit quad-core processor running at least 2GHz, to run the virtual machines, each on at least 1 processor core.
- RAM – minimum of 8GB, to support multiple instances of the virtual machines.
- Display unit and other peripherals.

Chapter 5

Design

This chapter describes the design of the project. It consists of the structure chart of the system and the functional descriptions of the modules present in it.

5.1 Structure Chart

A structure chart is used to represent hierarchical structure of modules in a particular system. The system is broken down into its lowest functional modules and then, these modules are described in detail, along with the functions present in them.

The structure chart partitions the system into *black boxes* wherein the functionality of the system is known to the users, but the inner working details are unknown. Inputs are given to these black boxes and outputs are generated.

Modules at the top level of the system call the modules at the lower levels. Components are read from top to bottom and left to right in the structure chart. When a module is called by another module, the called module is viewed as a black box, thereby passing required parameters and receiving results.

The implementation of defense mechanism for SYN flood attack requires the use of these modules –

- Main control module of implementation of a defense mechanism.
- Sub-modules for setting up the working environment, execution of SYN flood attack and execution of the defense mechanism.

Figure 5.1 describes the structure chart for the implementation of a defense mechanism for SYN flood attack. This chart specifies the various interactions between the modules for the complete implementation of the defense mechanism. The

main control module shows the overall output that is expected. The sub-modules represent the setting up of working environment, execution of SYN flood attack and finally, the execution of the defense mechanism.

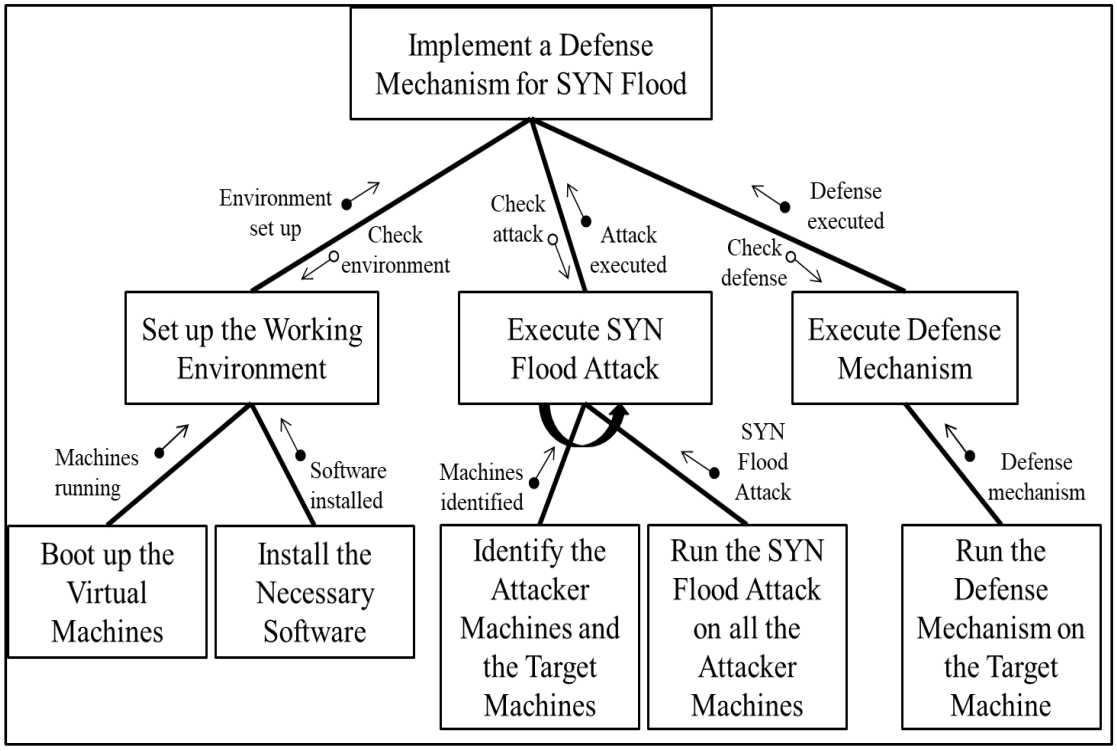


Figure 5.1 Structure chart for implementing SYN flood defense mechanism

5.2 Functional Description of the Modules

The modules of the structure chart are used to represent certain functionalities, with respect to the particular system. The implementation of a defense mechanism for SYN flood attack involves the use of one main module, three sub-modules and these sub-modules further have sub-modules.

5.2.1 Implement a Defense Mechanism for SYN Flood

This is the main control module of the system. It represents the overall system and is used to check for the execution of its sub-modules. Data flow occurs between the main module and its sub-modules.

5.2.2 Set up the Working Environment

The set up of the working environment is a sub-module in the system under the main control module. It is used to determine if the necessary environment has been set up for the execution of the SYN flood attack and for the implementation of the defense mechanism.

It is further divided into two sub-modules which are used to complete the main objective of setting up of the working environment. These sub-modules are –

- Boot up the virtual machines – this particular sub-module is used to depict the booting up of the virtual machines upon which the SYN flood attack is supposed to take place.
- Install the necessary software – this sub-module is used to depict the installation of vital software that are needed for the attack to take place.

5.2.3 Execute SYN Flood Attack

Execution of the SYN flood attack on the attacker machines is a sub-module under the main control module. It relies on the set up of the environment as a pre-requisite for its successful deployment.

It is further divided into two sub-modules which are used to execute the SYN flood attack step-by-step. These sub-modules are –

- Identify the attacker machines and the target machine – It is used to find out the attackers and the target. This gives a clear idea of where the attack is going to originate and where it is expected to be received.
- Run the SYN flood on all the attacker machines – The next step in the execution of the attack is to run the SYN flood itself. All the attacker machines run the script that generates the flooding of the SYN packets at the target.

5.2.4 Execute Defense Mechanism

The defense mechanism is implemented in this module, once the attack flourishes. This is used to specify that the defense mechanism is executed only after the attack occurs in this case. However, the defense mechanism can also be used as a prevention measure, which allows for the attack to be detected and defended against, as and when it begins at the origin.

The defense mechanism is executed on the target machine by using the root privileges and taking control of the incoming packets. The observations are made with respect to the necessary parameters and metrics, and the results are reported back to the main control module.

Chapter 6

Implementation

This chapter focuses on the implementation part of the SYN flood attack and its corresponding defense mechanism. It deals with the selection of programming language and platform, followed by the coding conventions used. It also describes the difficulties encountered and strategies used to tackle these difficulties.

6.1 Programming Language Selection

This is one of the most important decisions to be made whenever there is any execution of a code. Thus, some research was done before selecting a particular programming language.

A programming language that would be capable of handling network programming had to be chosen. The chosen programming language would need to have several functions or libraries and also support custom functions or libraries, in order to carry out the execution of the SYN flood attack and incorporate its defense mechanism.

C/C++ is a structured programming language that supports network programming, but is strenuous to work with, due to its over-dependence on syntax and semantic rules rather than the functionality itself. Similarly, Java offers a variety of object-oriented network programming utilities and also supports the portability of code extensively, but the run-time and memory overhead consumed is much larger in Java due to its own Java Virtual Machine (JVM) implementation.

Python overcomes these difficulties by providing an open-ended approach to writing code for network programming. It consists of various in-built functions and libraries that support socket programming for basic client-server communication and advanced network communication. The syntax used in Python is comparatively much

simpler and can be learnt and adapted easily for a development environment. Emphasis is given more on developing efficient code rather than depending on tons of lines of code and thereby increasing complexity.

Python offers the use of Scapy, a Python framework that enables the end user to send, sniff, dissect and forge packets in a network. This capability of Scapy allows for the construction of a wide variety of tools that can probe, scan or attack networks. It acts as an efficient and powerful packet manipulation tool that is able to forge or decode packets of a wide number of protocols, send them over a wired interface, capture them, match requests and replies, among many other functions. Thus, Python along with Scapy is used in the implementation of a SYN flood attack.

Implementation of the defense mechanism in particular uses commands that can be run on the terminal. These commands are those pertaining to the use of *iptables,* which are used to determine the policies for accepting, forwarding, rejecting or dropping any incoming packets into the network. These iptables rules are written in a shell script that scrapes for the incoming packets and rejects them based on their TCP packet signature.

6.2 Platform Selection

The platform upon which the implementation of the SYN flood attack and its defense mechanism is selected in this particular section. There are several operating systems that can be used to carry out this task.

The operating system Windows by Microsoft offers several functionalities and utilities to implement network programming. But it is limited by its restrictive network administration set up, which does not allow for pure packet transmission to take place either to a system in its local environment or to any system over the Internet.

Linux-based systems such as Ubuntu, Fedora, Arch Linux, Linux Mint, openSUSE, Debian, among many others, offer various network functionalities and features. Kali Linux is a very powerful distribution of Linux that consists of various networking tools and features. However, it is extremely secure due to its prevalence in ethical hacking and penetration testing. Thus, it is not considered for this implementation. Instead, Ubuntu is chosen as the main operating system platform upon which the SYN flood attack and its defense mechanism are implemented.

6.3 Code Conventions

The implementation of the code for SYN flood attack needs to be well-documented and has to follow certain code conventions. These code conventions are listed in this section.

6.3.1 Naming Conventions

The variables in the code and the functions always follow a straight-forward naming convention, bearing in mind the keywords supported by the Python programming language. This allows for easier understanding of the code and its variables. For example, the variable used for generation of an infinite number of packets is called as "packet_gen."

6.3.2 File Organization

The files are first edited in a suitable text editor such as Notepad++ or gedit. These files are then saved according to their necessary extensions. The attack script is saved as a Python file with a ".py" extension, the defense script is saved as a Shell script with a ".sh" extension and the logged output is stored as a text file with a ".txt" extension.

6.3.3 Comments

Each file and script has comments written in them in order to help the reader to easily understand the code and the underlying logic.

Chapter 7

Software Testing

This chapter focuses on the testing done during the implementation of the SYN flood attack and its defense mechanism. It describes the unit testing, integration testing and system testing for the modules implemented.

7.1 Test Environment

The test environment consists of the system on which the attack and the defense scripts are implemented and tested. The environment is first set up as per the software and hardware requirements and then, the individual modules are tested.

7.2 Unit Testing

Unit testing is a software testing method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures, are tested to determine whether they are fit for use.

In this implementation, the three modules of setting up the environment, execution of SYN flood attack and execution of the defense mechanism are tested unit-wise.

7.2.1 Unit Testing of Setting Up the Environment

The implementation environment of setting up the Virtual Machines (VM) and the installation of the necessary software requires unit testing. Each of the VMs is booted up several times to check for fatal errors, if any. The VMs are tested with various levels of RAM and assigned number of processors to ensure the smooth setup of the working environment.

7.2.2 Unit Testing of SYN Flood Attack

The SYN flood attack is tested multiple times with varying levels of time-to-live (TTL) values, payload, interval of flooding the packets and the timeout delay. The attack is carried out at different levels of CPU load in order to check if there is any effect on the performance of the attack.

7.2.3 Unit Testing of the Defense Mechanism

The defense mechanism for the SYN flood attack is tested against different incoming packets from the source attackers. The iptables rules are set with different parameters and tested for stability in the performance.

7.3 Integration Testing

Integration testing is the phase in software testing in which individual software modules are combined and tested as a group. Integration testing is conducted to evaluate the compliance of a system or component with specified functional requirements. Integration testing takes as its input modules that have been unit tested, groups them in larger aggregates, applies tests defined in an integration test plan to those aggregates, and delivers as its output the integrated system ready for system testing.

In this implementation, the individual modules are integrated together and tested for any issues in the software and hardware. The environment is set up first and then, the attack is carried out and the defense mechanism is implemented. The different performance tests for the attack and the analyzer tools are made with respect to the environment that is set up for the implementation of the individual modules.

7.4 System Testing

System testing is testing conducted on a complete integrated system to evaluate the system's compliance with its specified requirements. System testing takes, as its input, all of the integrated components that have passed integration testing. The purpose of integration testing is to detect any inconsistencies between the units that are integrated together (called assemblages). System testing seeks to detect defects both within the "inter-assemblages" and also within the system as a whole. The actual result is the behavior produced or observed when a component or system is tested.

System testing is performed on the entire system in the context of either functional requirement specifications (FRS) or system requirement specification (SRS), or both. System testing tests not only the design, but also the behaviour and even the believed expectations of the customer. It is also intended to test up to and beyond the bounds defined in the software or hardware requirements specification(s).

The setting up of the environment, execution of the SYN flood attack and the defense mechanism are considered as a whole and tested. The operating system running the entire set up, the network adapters and the necessary software and hardware requirements are all tested in the system testing.

Chapter 8

Experimental Analysis and Results

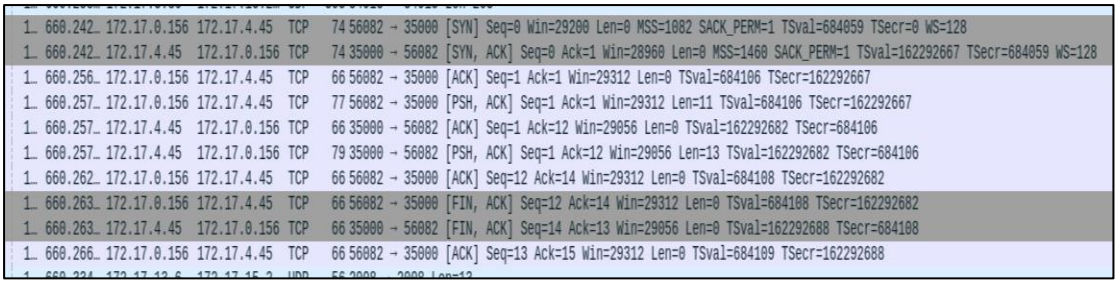
This section describes the analysis done on the TCP SYN flood attack and the subsequent results obtained. A comparative analysis is also done with the ICMP flood attack in terms of overall attack time and the average round trip time, with varying the number of packets.

8.1 Observations and Results

The SYN flood attack is carried out on the target system by the attacker system, and later the DDoS variant is also tested. Some of the main observations and results are given as follows –

- SYN flood attack occurs extremely fast due to less intervals and time-to-live (TTL) values.
- The outputs observed at Wireshark indicate that the continuous RST+ACK control signals that are sent without the defense mechanism in place are caused due to the buffer overflow occurring in the system.
- When the defense mechanism is put in place, the continuous RST+ACK control signals that are generated are due to the iptables rules that reject the incoming packets.
- It can also be observed that using iptables allows for easier filtering of the incoming packets in to the network. Only the necessary packets are allowed and the remaining ones are discarded.

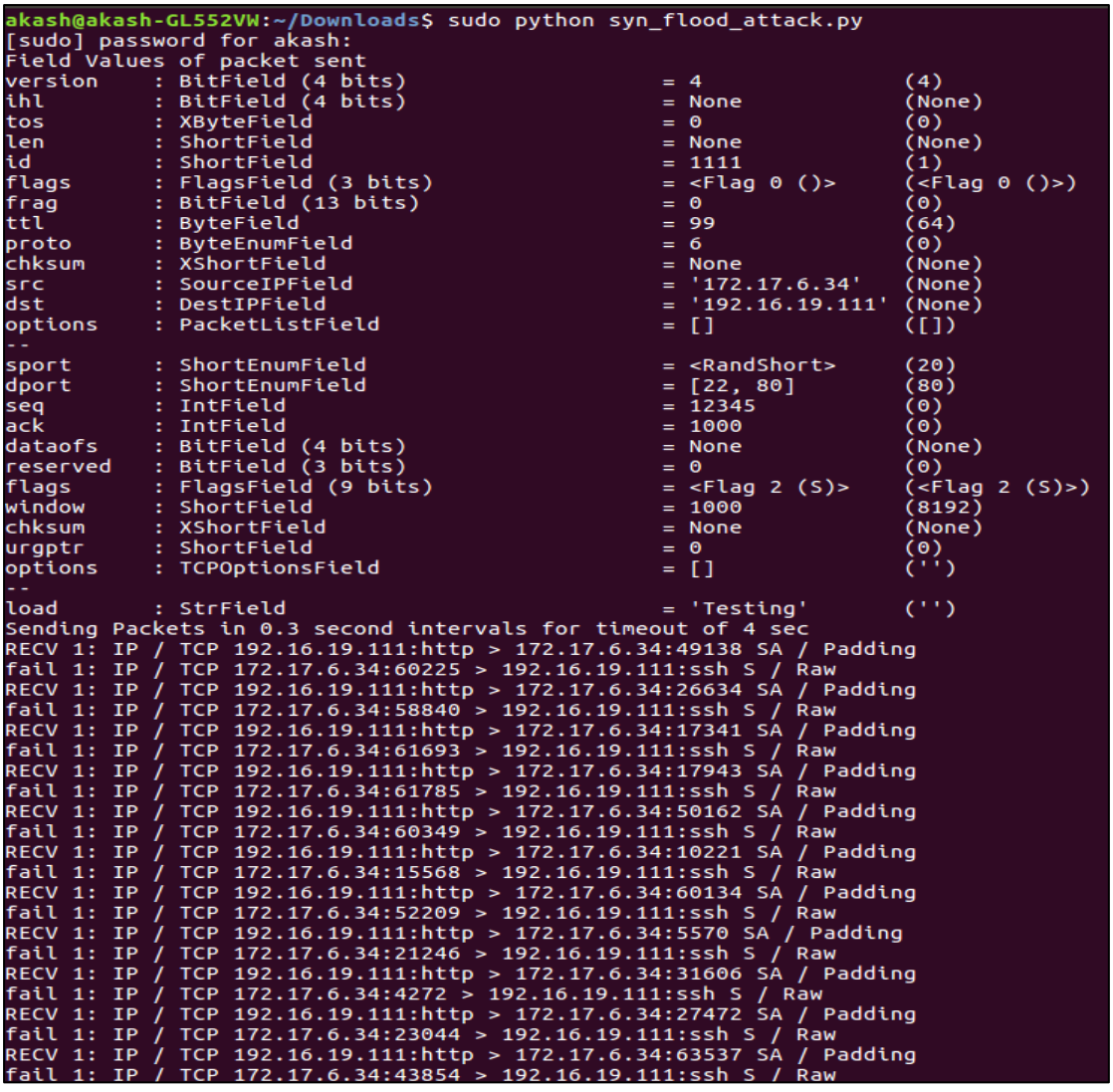
Figure 8.1 shows the output of a normal TCP 3-way handshake. It can be observed that the SYN packet is sent first to the target, which then responds with an ACK. This is followed by the data transmission that takes place between the attacker and the target. Finally, the connection is ended with a FIN packet.



1…	660.242…	172.17.0.156	172.17.4.45	TCP	74 56082 → 35000 [SYN] Seq=0 Win=29200 Len=0 MSS=1082 SACK_PERM=1 TSval=684059 TSecr=0 WS=128
1…	660.242…	172.17.4.45	172.17.0.156	TCP	74 35000 → 56082 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=162292667 TSecr=684059 WS=128
1…	660.256…	172.17.0.156	172.17.4.45	TCP	66 56082 → 35000 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=684106 TSecr=162292667
1…	660.257…	172.17.0.156	172.17.4.45	TCP	77 56082 → 35000 [PSH, ACK] Seq=1 Ack=1 Win=29312 Len=11 TSval=684106 TSecr=162292667
1…	660.257…	172.17.4.45	172.17.0.156	TCP	66 35000 → 56082 [ACK] Seq=1 Ack=12 Win=29056 Len=0 TSval=162292682 TSecr=684106
1…	660.257…	172.17.4.45	172.17.0.156	TCP	79 35000 → 56082 [PSH, ACK] Seq=1 Ack=12 Win=29056 Len=13 TSval=162292682 TSecr=684106
1…	660.262…	172.17.0.156	172.17.4.45	TCP	66 56082 → 35000 [ACK] Seq=12 Ack=14 Win=29312 Len=0 TSval=684108 TSecr=162292682
1…	660.263…	172.17.0.156	172.17.4.45	TCP	66 56082 → 35000 [FIN, ACK] Seq=12 Ack=14 Win=29312 Len=0 TSval=684108 TSecr=162292682
1…	660.263…	172.17.4.45	172.17.0.156	TCP	66 35000 → 56082 [FIN, ACK] Seq=14 Ack=13 Win=29056 Len=0 TSval=162292688 TSecr=684108
1…	660.266…	172.17.0.156	172.17.4.45	TCP	66 56082 → 35000 [ACK] Seq=13 Ack=15 Win=29312 Len=0 TSval=684109 TSecr=162292688

Figure 8.1 Output of normal TCP 3-way handshake

Figure 8.2 shows the logged outputs for the normal TCP 3-way handshake.



```
akash@akash-GL552VW:~/Downloads$ sudo python syn_flood_attack.py
[sudo] password for akash:
Field Values of packet sent
version     : BitField (4 bits)                 = 4                 (4)
ihl         : BitField (4 bits)                 = None              (None)
tos         : XByteField                        = 0                 (0)
len         : ShortField                        = None              (None)
id          : ShortField                        = 1111              (1)
flags       : FlagsField (3 bits)               = <Flag 0 ()>       (<Flag 0 ()>)
frag        : BitField (13 bits)                = 0                 (0)
ttl         : ByteField                         = 99                (64)
proto       : ByteEnumField                     = 6                 (0)
chksum      : XShortField                       = None              (None)
src         : SourceIPField                     = '172.17.6.34'     (None)
dst         : DestIPField                       = '192.16.19.111'   (None)
options     : PacketListField                   = []                ([])
--
sport       : ShortEnumField                    = <RandShort>       (20)
dport       : ShortEnumField                    = [22, 80]          (80)
seq         : IntField                          = 12345             (0)
ack         : IntField                          = 1000              (0)
dataofs     : BitField (4 bits)                 = None              (None)
reserved    : BitField (3 bits)                 = 0                 (0)
flags       : FlagsField (9 bits)               = <Flag 2 (S)>      (<Flag 2 (S)>)
window      : ShortField                        = 1000              (8192)
chksum      : XShortField                       = None              (None)
urgptr      : ShortField                        = 0                 (0)
options     : TCPOptionsField                   = []                ('')
--
load        : StrField                          = 'Testing'         ('')
Sending Packets in 0.3 second intervals for timeout of 4 sec
RECV 1: IP / TCP 192.16.19.111:http > 172.17.6.34:49138 SA / Padding
fail 1: IP / TCP 172.17.6.34:60225 > 192.16.19.111:ssh S / Raw
RECV 1: IP / TCP 192.16.19.111:http > 172.17.6.34:26634 SA / Padding
fail 1: IP / TCP 172.17.6.34:58840 > 192.16.19.111:ssh S / Raw
RECV 1: IP / TCP 192.16.19.111:http > 172.17.6.34:17341 SA / Padding
fail 1: IP / TCP 172.17.6.34:61693 > 192.16.19.111:ssh S / Raw
RECV 1: IP / TCP 192.16.19.111:http > 172.17.6.34:17943 SA / Padding
fail 1: IP / TCP 172.17.6.34:61785 > 192.16.19.111:ssh S / Raw
RECV 1: IP / TCP 192.16.19.111:http > 172.17.6.34:50162 SA / Padding
fail 1: IP / TCP 172.17.6.34:60349 > 192.16.19.111:ssh S / Raw
RECV 1: IP / TCP 192.16.19.111:http > 172.17.6.34:10221 SA / Padding
fail 1: IP / TCP 172.17.6.34:15568 > 192.16.19.111:ssh S / Raw
RECV 1: IP / TCP 192.16.19.111:http > 172.17.6.34:60134 SA / Padding
fail 1: IP / TCP 172.17.6.34:52209 > 192.16.19.111:ssh S / Raw
RECV 1: IP / TCP 192.16.19.111:http > 172.17.6.34:5570 SA / Padding
fail 1: IP / TCP 172.17.6.34:21246 > 192.16.19.111:ssh S / Raw
RECV 1: IP / TCP 192.16.19.111:http > 172.17.6.34:31606 SA / Padding
fail 1: IP / TCP 172.17.6.34:4272 > 192.16.19.111:ssh S / Raw
RECV 1: IP / TCP 192.16.19.111:http > 172.17.6.34:27472 SA / Padding
fail 1: IP / TCP 172.17.6.34:23044 > 192.16.19.111:ssh S / Raw
RECV 1: IP / TCP 192.16.19.111:http > 172.17.6.34:63537 SA / Padding
fail 1: IP / TCP 172.17.6.34:43854 > 192.16.19.111:ssh S / Raw
```

Figure 8.2 Logged output for normal TCP 3-way handshake

All the field values of the packets that are sent can be observed in Figure 8.2. This includes the source and the destination IP addresses, sequence number, packet ID, source and destination ports, flags and the payload that is being sent. Figure 8.3 shows the detailed summary of the packets that were sent and the received acknowledgements.

```
Sent 46 packets, received 23 packets. 50.0% hits.
Summary of answered & unanswered packets
IP / TCP 172.17.6.34:49138 > 192.16.19.111:http S / Raw ==> IP / TCP 192.16.19.111:http > 172.17.6.34:49138 SA / Padding
IP / TCP 172.17.6.34:26634 > 192.16.19.111:http S / Raw ==> IP / TCP 192.16.19.111:http > 172.17.6.34:26634 SA / Padding
IP / TCP 172.17.6.34:17341 > 192.16.19.111:http S / Raw ==> IP / TCP 192.16.19.111:http > 172.17.6.34:17341 SA / Padding
IP / TCP 172.17.6.34:17943 > 192.16.19.111:http S / Raw ==> IP / TCP 192.16.19.111:http > 172.17.6.34:17943 SA / Padding
IP / TCP 172.17.6.34:50162 > 192.16.19.111:http S / Raw ==> IP / TCP 192.16.19.111:http > 172.17.6.34:50162 SA / Padding
IP / TCP 172.17.6.34:10221 > 192.16.19.111:http S / Raw ==> IP / TCP 192.16.19.111:http > 172.17.6.34:10221 SA / Padding
IP / TCP 172.17.6.34:60134 > 192.16.19.111:http S / Raw ==> IP / TCP 192.16.19.111:http > 172.17.6.34:60134 SA / Padding
IP / TCP 172.17.6.34:5570 > 192.16.19.111:http S / Raw ==> IP / TCP 192.16.19.111:http > 172.17.6.34:5570 SA / Padding
IP / TCP 172.17.6.34:31606 > 192.16.19.111:http S / Raw ==> IP / TCP 192.16.19.111:http > 172.17.6.34:31606 SA / Padding
IP / TCP 172.17.6.34:27472 > 192.16.19.111:http S / Raw ==> IP / TCP 192.16.19.111:http > 172.17.6.34:27472 SA / Padding
IP / TCP 172.17.6.34:63537 > 192.16.19.111:http S / Raw ==> IP / TCP 192.16.19.111:http > 172.17.6.34:63537 SA / Padding
IP / TCP 172.17.6.34:62868 > 192.16.19.111:http S / Raw ==> IP / TCP 192.16.19.111:http > 172.17.6.34:62868 SA / Padding
IP / TCP 172.17.6.34:40643 > 192.16.19.111:http S / Raw ==> IP / TCP 192.16.19.111:http > 172.17.6.34:40643 SA / Padding
IP / TCP 172.17.6.34:21010 > 192.16.19.111:http S / Raw ==> IP / TCP 192.16.19.111:http > 172.17.6.34:21010 SA / Padding
IP / TCP 172.17.6.34:35304 > 192.16.19.111:http S / Raw ==> IP / TCP 192.16.19.111:http > 172.17.6.34:35304 SA / Padding
IP / TCP 172.17.6.34:41798 > 192.16.19.111:http S / Raw ==> IP / TCP 192.16.19.111:http > 172.17.6.34:41798 SA / Padding
IP / TCP 172.17.6.34:8797 > 192.16.19.111:http S / Raw ==> IP / TCP 192.16.19.111:http > 172.17.6.34:8797 SA / Padding
IP / TCP 172.17.6.34:1344 > 192.16.19.111:http S / Raw ==> IP / TCP 192.16.19.111:http > 172.17.6.34:1344 SA / Padding
IP / TCP 172.17.6.34:40667 > 192.16.19.111:http S / Raw ==> IP / TCP 192.16.19.111:http > 172.17.6.34:40667 SA / Padding
IP / TCP 172.17.6.34:5641 > 192.16.19.111:http S / Raw ==> IP / TCP 192.16.19.111:http > 172.17.6.34:5641 SA / Padding
IP / TCP 172.17.6.34:22214 > 192.16.19.111:http S / Raw ==> IP / TCP 192.16.19.111:http > 172.17.6.34:22214 SA / Padding
IP / TCP 172.17.6.34:36230 > 192.16.19.111:http S / Raw ==> IP / TCP 192.16.19.111:http > 172.17.6.34:36230 SA / Padding
IP / TCP 172.17.6.34:58619 > 192.16.19.111:http S / Raw ==> IP / TCP 192.16.19.111:http > 172.17.6.34:58619 SA / Padding
IP / TCP 172.17.6.34:60225 > 192.16.19.111:ssh S / Raw
IP / TCP 172.17.6.34:58840 > 192.16.19.111:ssh S / Raw
IP / TCP 172.17.6.34:61693 > 192.16.19.111:ssh S / Raw
IP / TCP 172.17.6.34:61785 > 192.16.19.111:ssh S / Raw
IP / TCP 172.17.6.34:60349 > 192.16.19.111:ssh S / Raw
IP / TCP 172.17.6.34:15568 > 192.16.19.111:ssh S / Raw
IP / TCP 172.17.6.34:52209 > 192.16.19.111:ssh S / Raw
IP / TCP 172.17.6.34:21246 > 192.16.19.111:ssh S / Raw
IP / TCP 172.17.6.34:4272 > 192.16.19.111:ssh S / Raw
IP / TCP 172.17.6.34:23044 > 192.16.19.111:ssh S / Raw
IP / TCP 172.17.6.34:43854 > 192.16.19.111:ssh S / Raw
IP / TCP 172.17.6.34:61465 > 192.16.19.111:ssh S / Raw
IP / TCP 172.17.6.34:713 > 192.16.19.111:ssh S / Raw
IP / TCP 172.17.6.34:13162 > 192.16.19.111:ssh S / Raw
IP / TCP 172.17.6.34:7116 > 192.16.19.111:ssh S / Raw
IP / TCP 172.17.6.34:32272 > 192.16.19.111:ssh S / Raw
IP / TCP 172.17.6.34:54611 > 192.16.19.111:ssh S / Raw
IP / TCP 172.17.6.34:23496 > 192.16.19.111:ssh S / Raw
IP / TCP 172.17.6.34:43498 > 192.16.19.111:ssh S / Raw
IP / TCP 172.17.6.34:8993 > 192.16.19.111:ssh S / Raw
IP / TCP 172.17.6.34:38970 > 192.16.19.111:ssh S / Raw
IP / TCP 172.17.6.34:44184 > 192.16.19.111:ssh S / Raw
IP / TCP 172.17.6.34:46164 > 192.16.19.111:ssh S / Raw
```

Figure 8.3 Summary of the transmitted packets

This is followed by the implementation of our SYN flood attack along with the defense mechanism, which yields the packets as shown in Figure 8.4. The SYN+ACK is replaced by the RST+ACK instead, and it can be observed that the packets are being rejected and the TCP connection is being closed immediately.

1…	422.026…	172.17.0.156	172.17.4.45	SSH	61	Client: Encrypted packet (len=7)
1…	422.026…	172.17.4.45	172.17.0.156	TCP	54	22 → 7926 [RST, ACK] Seq=1 Ack=8 Win=0 Len=0
1…	422.026…	172.17.0.156	172.17.4.45	TCP	61	63476 → 80 [SYN] Seq=0 Win=1000 Len=7 [TCP segment of a reassembled PDU]
1…	422.026…	172.17.4.45	172.17.0.156	TCP	54	80 → 63476 [RST, ACK] Seq=1 Ack=8 Win=0 Len=0
1…	422.129…	172.17.13.6	172.17.15.2…	UDP	56	2007 → 2007 Len=13
1…	422.132…	172.17.0.243	172.17.15.2…	UDP	305	54915 → 54915 Len=263
1…	422.135…	172.17.3.80	172.17.15.2…	UDP	305	54915 → 54915 Len=263
1…	422.231…	172.17.13.6	172.17.15.2…	UDP	56	2007 → 2007 Len=13
1…	422.231…	172.17.2.29	172.17.15.2…	NBNS	92	Name query NB WORKGROUP<1c>
1…	422.333…	172.17.0.156	172.17.4.45	SSH	61	Client: Encrypted packet (len=7)
1…	422.333…	172.17.4.45	172.17.0.156	TCP	54	22 → 51443 [RST, ACK] Seq=1 Ack=8 Win=0 Len=0
1…	422.333…	172.17.0.156	172.17.4.45	TCP	61	26865 → 80 [SYN] Seq=0 Win=1000 Len=7 [TCP segment of a reassembled PDU]
1…	422.333…	172.17.4.45	172.17.0.156	TCP	54	80 → 26865 [RST, ACK] Seq=1 Ack=8 Win=0 Len=0
1…	422.436…	172.17.13.6	172.17.15.2…	UDP	56	2008 → 2008 Len=13
1…	422.436…	172.17.13.6	172.17.15.2…	UDP	56	2008 → 2008 Len=13
1…	422.437…	172.17.13.6	172.17.15.2…	UDP	56	2007 → 2007 Len=13
1…	422.537…	172.17.13.6	172.17.15.2…	UDP	56	2007 → 2007 Len=13
1…	422.642…	172.17.13.6	172.17.15.2…	UDP	56	2008 → 2008 Len=13
1…	422.642…	172.17.13.6	172.17.15.2…	UDP	56	2007 → 2007 Len=13
1…	422.642…	172.17.0.156	172.17.4.45	SSH	61	Client: Encrypted packet (len=7)
1…	422.642…	172.17.4.45	172.17.0.156	TCP	54	22 → 30784 [RST, ACK] Seq=1 Ack=8 Win=0 Len=0
1…	422.642…	172.17.0.156	172.17.4.45	TCP	61	37677 → 80 [SYN] Seq=0 Win=1000 Len=7 [TCP segment of a reassembled PDU]
1…	422.642…	172.17.4.45	172.17.0.156	TCP	54	80 → 37677 [RST, ACK] Seq=1 Ack=8 Win=0 Len=0
1…	422.727…	172.17.13.6	172.17.15.2…	UDP	56	2008 → 2008 Len=13
1…	422.727…	172.17.13.6	172.17.15.2…	UDP	56	2007 → 2007 Len=13
1…	422.844…	172.17.13.6	172.17.15.2…	UDP	56	2008 → 2008 Len=13
1…	422.845…	172.17.13.6	172.17.15.2…	UDP	56	2007 → 2007 Len=13
1…	422.947…	172.17.13.6	172.17.15.2…	UDP	56	2008 → 2008 Len=13
1…	422.951…	172.17.0.156	172.17.4.45	SSH	61	Client: Encrypted packet (len=7)
1…	422.951…	172.17.4.45	172.17.0.156	TCP	54	22 → 21257 [RST, ACK] Seq=1 Ack=8 Win=0 Len=0
1…	422.951…	172.17.0.156	172.17.4.45	TCP	61	49753 → 80 [SYN] Seq=0 Win=1000 Len=7 [TCP segment of a reassembled PDU]
1…	422.951…	172.17.4.45	172.17.0.156	TCP	54	80 → 49753 [RST, ACK] Seq=1 Ack=8 Win=0 Len=0
1…	423.023…	172.17.13.6	172.17.15.2…	UDP	56	2008 → 2008 Len=13
1…	423.023…	172.17.13.6	172.17.15.2…	UDP	56	2007 → 2007 Len=13
1…	423.028…	172.17.0.243	172.17.15.2…	UDP	305	54915 → 54915 Len=263

Figure 8.4 RST+ACK received upon flooding

This implementation is further expanded to include a DDoS variant, where there are multiple attackers and a single target. This is shown in Figure 8.5.

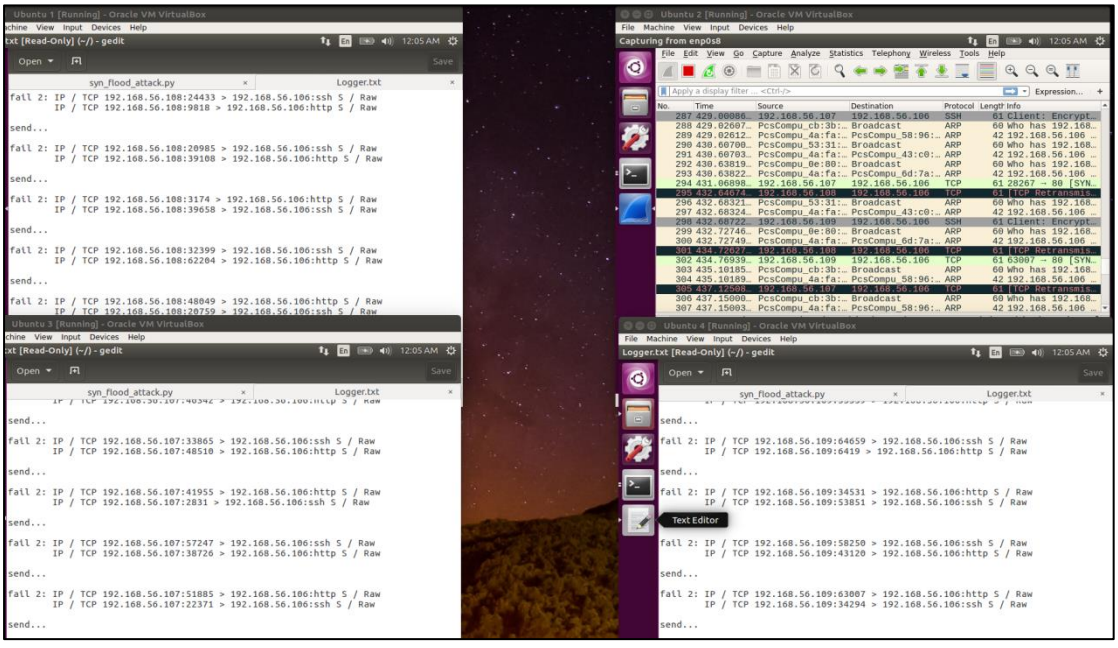


Figure 8.5 DDoS implementation of SYN flood

8.2 Comparative Analysis with ICMP Flood

A comparative analysis of the SYN flood attack with that of ICMP flood attack is done, with respect to the total attack time and the average round trip time. The number of packets are varied with each trial, ranging from 100 to 300. The total number of trials for a fixed number of packets are 5. These are then compared and an analysis is made. Figure 8.6 depicts the a distributed variant of the ICMP flood taking place in a virtual machine setup.

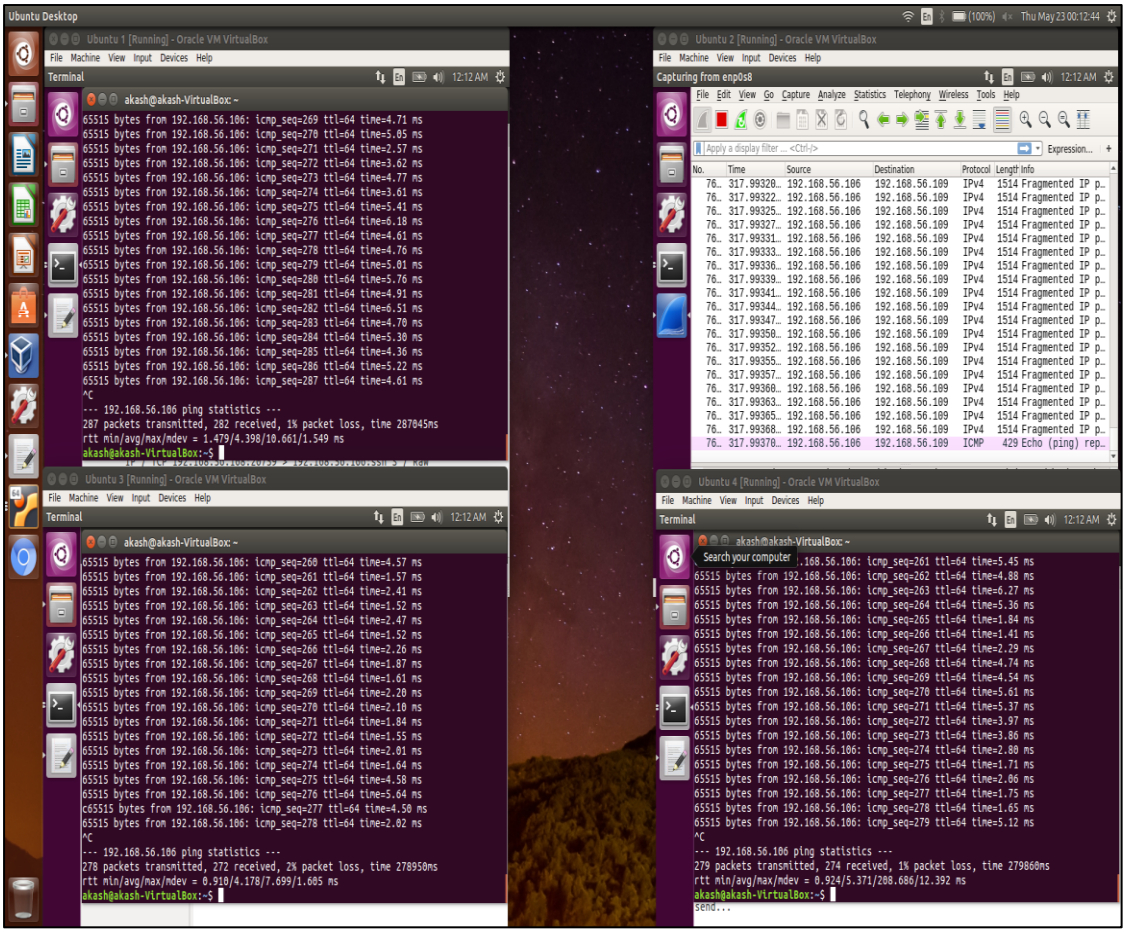


Figure 8.6 DDoS implementation of ICMP flood

Tables 8.1, 8.2 and 8.3 show the total attack time taken by both SYN flood and ICMP flood for three different numbers of packets, namely 100, 200 and 300. Figures 8.7, 8.8 and 8.9 depict the total attack times in terms of bar graphs. It can be observed that the total attack time taken by the SYN flood is significantly lesser than that of ICMP flood.

Table 8.1 Total attack time for flooding of 100 packets

Trials (Number of packets = 100)	Total Attack Time for SYN Flood	Total Attack Time for ICMP Flood
Trial 1	31.56 s	92.74 s
Trial 2	30.78 s	93.98 s
Trial 3	31.73 s	91.54 s
Trial 4	29.43 s	92.09 s
Trial 5	30.21 s	93.17 s

Table 8.2 Total attack time for flooding of 200 packets

Trials (Number of packets = 200)	Total Attack Time for SYN Flood	Total Attack Time for ICMP Flood
Trial 1	62.64 s	189.23 s
Trial 2	61.12 s	188.07 s
Trial 3	59.93 s	190.35 s
Trial 4	60.76 s	191.54 s
Trial 5	61.33 s	189.88 s

Table 8.3 Total attack time for flooding of 300 packets

Trials (Number of packets = 300)	Total Attack Time for SYN Flood	Total Attack Time for ICMP Flood
Trial 1	89.64 s	278.95 s
Trial 2	91.82 s	287.04 s
Trial 3	90.73 s	279.86 s
Trial 4	90.01 s	280.45 s
Trial 5	88.62 s	279.08 s

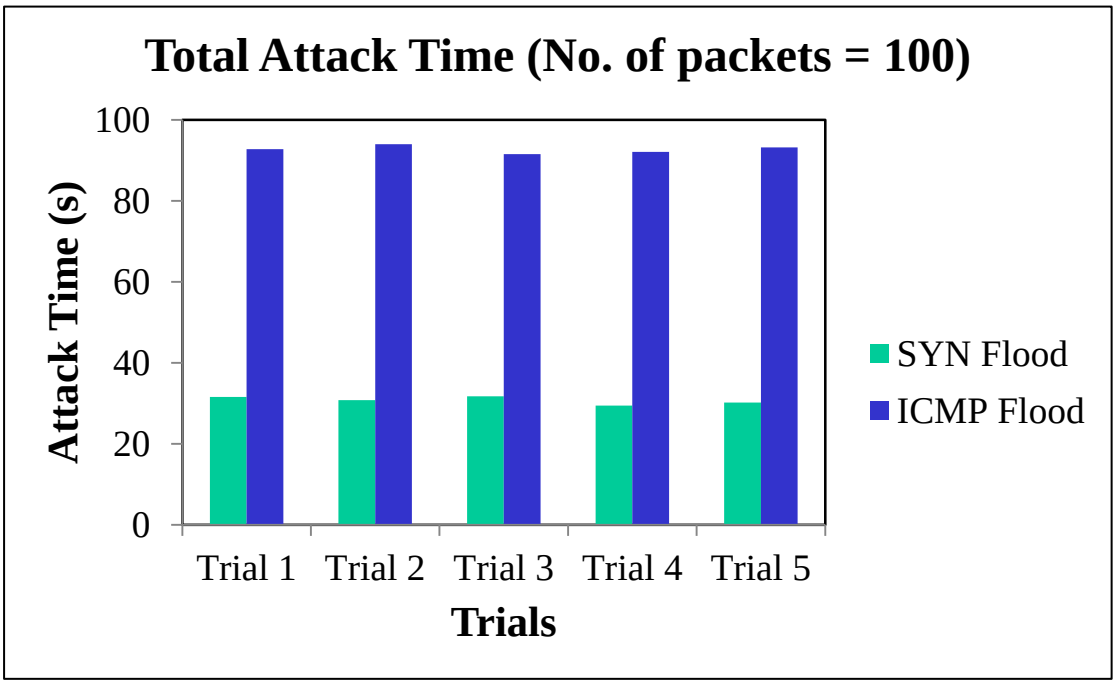


Figure 8.7 Graph of total attack time for flooding of 100 packets

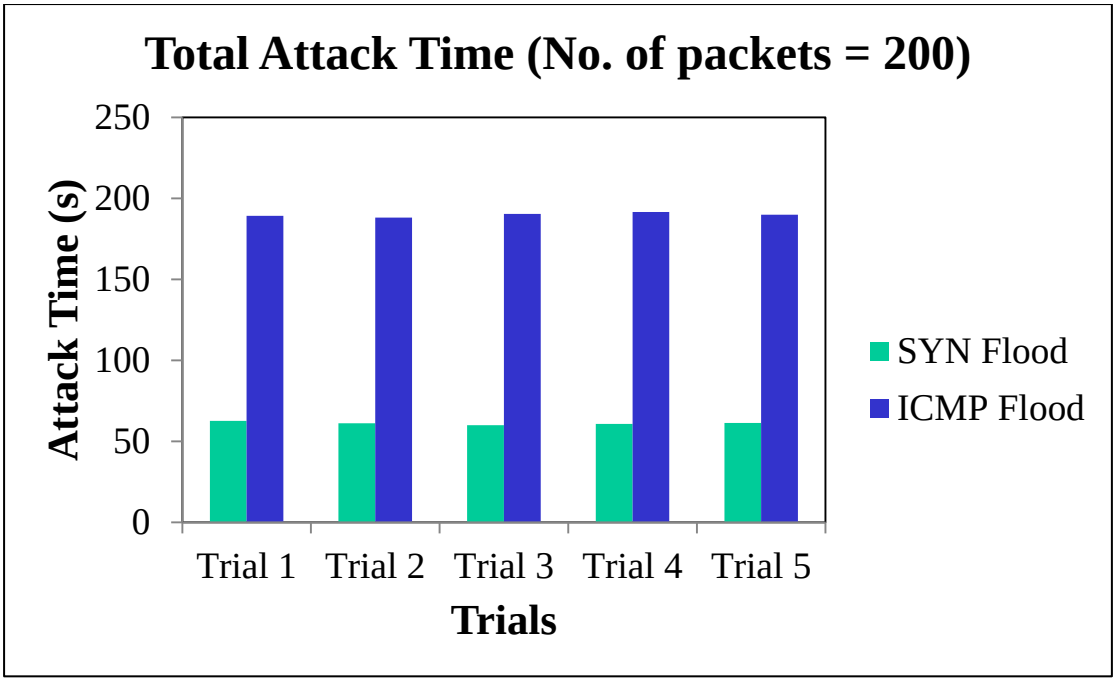


Figure 8.8 Graph of total attack time for flooding of 200 packets

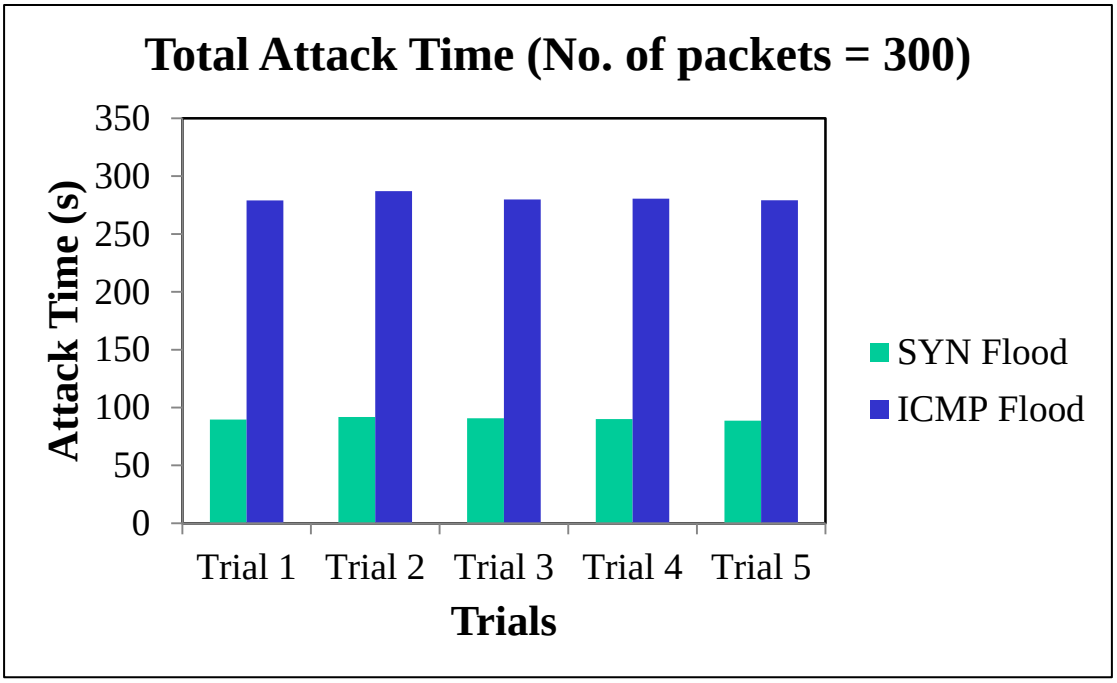


Figure 8.9 Graph of total attack time for flooding of 300 packets

The average round trip times (RTT) for the SYN flood and the ICMP flood attacks are measured for the varying number of packets. The average round trip time is the time taken for a packet to travel to the destination and then reach back to the source. This is done for 5 trials each, for the number of packets ranging from 100 to 300.

Tables 8.4, 8.5, 8.6 represent the average round trip times for the SYN flood and the ICMP flood attacks, for the number of packets being 100, 200 and 300 respectively.

This is followed by Figures 8.10, 8.11 and 8.12, where a bar graph is used to depict the average round trip times. It can be observed that the average round trip time taken for SYN flood is lesser than that of ICMP flood. This can be attributed to the fact that there is significantly higher reply packets to be sent in ICMP flood than in SYN flood.

Table 8.4 Average round trip time for flooding of 100 packets

Trials (Number of packets = 100)	**Average RTT for SYN Flood**	**Average RTT for ICMP Flood**
Trial 1	0.66 ms	2.43 ms
Trial 2	0.68 ms	2.54 ms
Trial 3	0.65 ms	2.48 ms
Trial 4	0.70 ms	2.69 ms
Trial 5	0.62 ms	2.77 ms

Table 8.5 Average round trip time for flooding of 200 packets

Trials (Number of packets = 200)	Average RTT for SYN Flood	Average RTT for ICMP Flood
Trial 1	0.76 ms	3.46 ms
Trial 2	0.78 ms	3.39 ms
Trial 3	0.71 ms	3.51 ms
Trial 4	0.72 ms	3.66 ms
Trial 5	0.81 ms	3.25 ms

Table 8.6 Average round trip time for flooding of 300 packets

Trials (Number of packets = 300)	Average RTT for SYN Flood	Average RTT for ICMP Flood
Trial 1	0.89 ms	4.39 ms
Trial 2	0.98 ms	4.17 ms
Trial 3	0.93 ms	5.37 ms
Trial 4	0.86 ms	5.24 ms
Trial 5	0.89 ms	4.77 ms

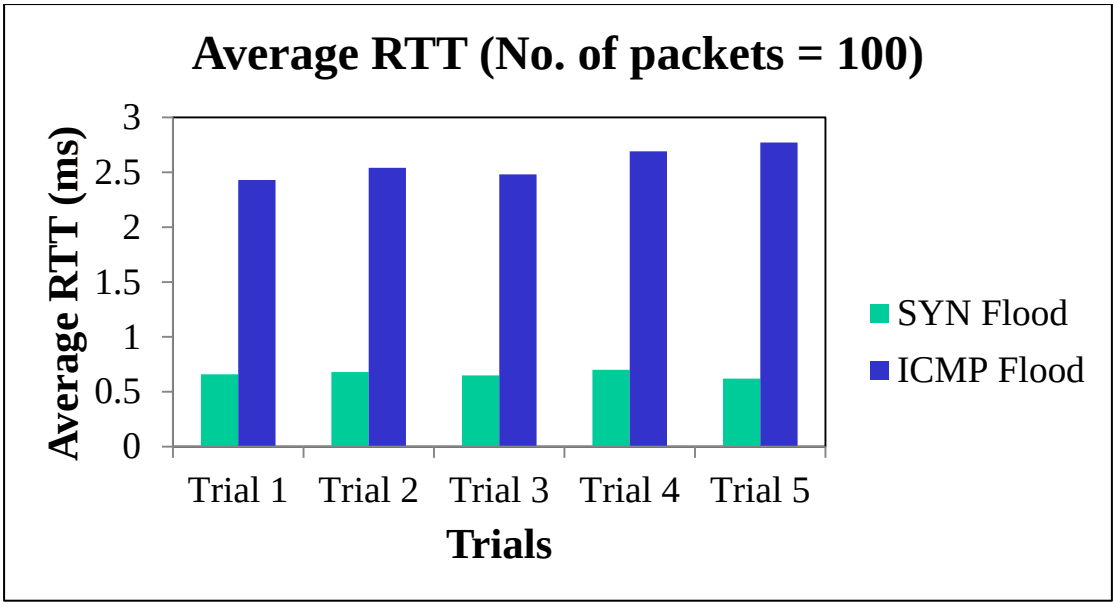


Figure 8.10 Graph of average round trip time for flooding of 100 packets

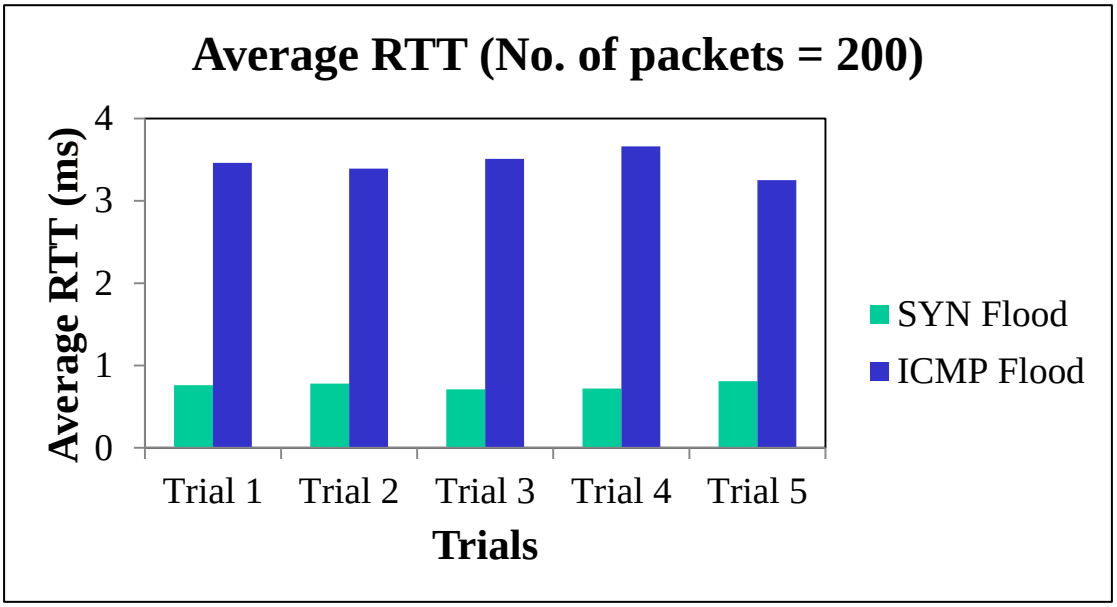


Figure 8.11 Graph of average round trip time for flooding of 200 packets

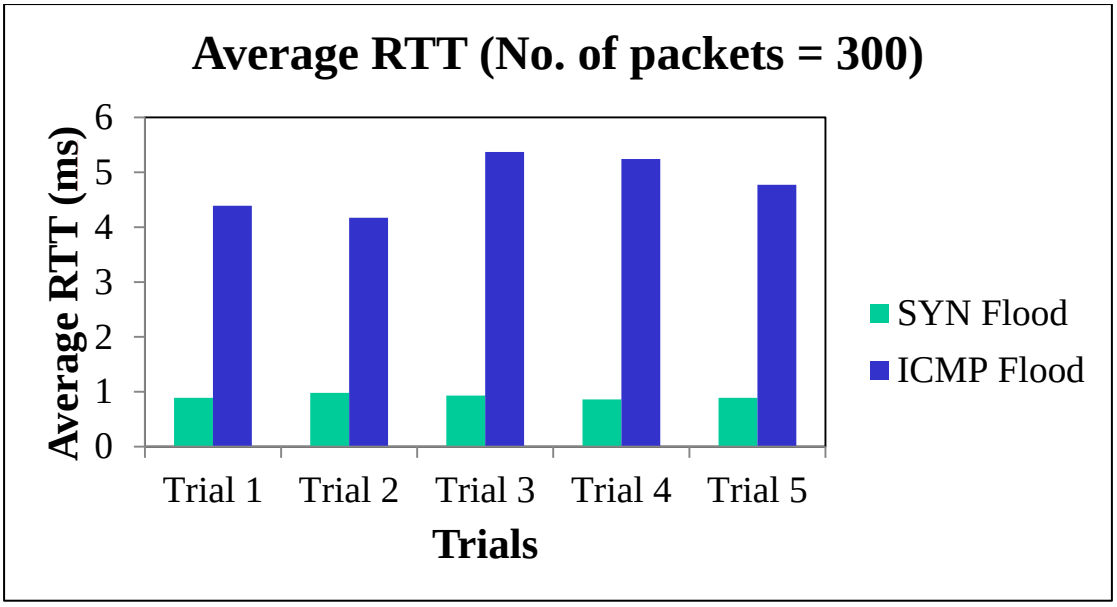


Figure 8.12 Graph of average round trip time for flooding of 300 packets

8.3 Inference from the Results

The results show that the amount of total attack time taken is significantly lesser for the SYN flood attack than the ICMP flood attack. This could be attributed to the fact that the ICMP flood relies on the ping command, which can carry heavy payloads and can unnecessarily take extra time for its completion.

The average round trip times (RTT) for the SYN flood attack is also much lesser when compared to that of ICMP flood attack. This can be attributed to the fact that there is a delay experienced by the ICMP flood attack due to the nature of the heavy payload ping, which will take some time for the destination to be able to respond to the original packet. This will incur higher round trip times.

Overall, it can be inferred from the results that the SYN flood attack occurs much faster than that of ICMP flood attack. The defense mechanism hence employed to block SYN flood attacks is much better due to the nature of TCP responding immediately to the iptables rules that are set in order to block the incoming packets and close the TCP connection.

Chapter 9

Conclusion

The TCP SYN flood attack occurs when there is a continuous flow of SYN packets from a source to a destination, which renders the destination node unable to serve any other clients. A detailed study of the TCP SYN flood attack is made and its distributed variant DDoS, where in there exist multiple attackers and a single target.

A brief study is made on iptables rules and the procedure to set up these iptables rules and configure them. A multi-machine set up is made for the demonstration of the SYN flood attack, where in the source attacker is used to flood the destination target. It is observed that the iptables rules help in effective rejection of the SYN packets from the flooding. The TCP connection is closed immediately and a RST+ACK response is sent to the attacker machine. It is also observed that iptables rules help in easier filtering of the packets, by checking the source IP addresses.

A comparative analysis is made between the SYN flood attack and the ICMP flood attack, with respect to the total attack time and the average round trip time for varying number of flooding packets. It is observed that the SYN flood attack occurs faster than the ICMP flood attack and also has significantly lesser average round trip time than that of the ICMP flood attack.

The SYN flood attack may be enhanced in order to easily attack the target systems. Thus, there can be automated setting of iptables rules to detect the abnormal incoming packets into the network. Furthermore, frequently attacking IP addresses can be blacklisted, which would improve the response time of the target to such attacks.

References

[1] Nagai R., Kurihara W., Higuchi S. and Hirots T., "Design and Implementation of an OpenFlow-based TCP SYN Flood Mitigation", *IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*, Bamberg, Germany, 2018, pp. 37-42.

[2] Thang T. M., Nguyen C. Q. and Nguyen V. K., "Synflood Spoofed Source DDoS Attack Defense Based on Packet ID Anomaly Detection with Bloom Filter", *Asian Conference on Defense Technology (ACDT)*, Hanoi, Vietnam, 2018, pp. 75-80.

[3] Kavisankar L., Chellappan C., Venkatesan S. and Sivasankar P., "Efficient SYN spoofing Detection and Mitigation Scheme for DDoS attack", *International Conference on Recent Trends and Challenges in Computational Models (ICRTCCM)*, Tindivanam, India, 2017, pp. 269-274.

[4] Rani D. D., Krishna T. V. S., Dayanandam G. and Rao T. V., "TCP SYN Flood Attack Detection And Prevention", *International Journal of Computer Trends and Technology (IJCTT)*, vol. 4, issue10, 2013, pp. 3412-3417.

[5] Al-Musawi B. Q. M., "Mitigating DoS/DDoS Attacks using iptables", *International Journal of Engineering & Technology IJET-IJENS*, vol. 12 issue 03, 2012, pp. 101-111.

[6] Wang H., Zhang D., and Shin K. G., “Detecting SYN flooding attacks”, *Proceedings of Annual Joint Conference of the IEEE Computer and Communications Societies(INFOCOM)*, volume 3, pages 1530-1539, June 23-27 2002.

[7] G. Wei, Y. Gu and Y. Ling, "An Early Stage Detecting Method against SYN Flooding Attack," csa, pp.263-268, *International Symposium on Computer Science and its Applications*, 2008.

[8] D. M. Divakaran, H. A. Murthy and T. A. Gonsalves, ``Detection of SYN Flooding Attacks Using Linear Prediction Analysis," *14th IEEE International Conference on Networks, ICON 2006*, pp. 218-223, Sep. 2006.

[9] V. A. Siris and P. Fotini, “Application of Anomaly Detect Algorithms for Detecting SYN Flooding Attack” *Elsevier Computer Communications*, 29: 1433-1442, 2006.

[10] S.Gavaskar, R.Surendiran and Dr.E.Ramaraj, "Three Counter Defense Mechanism for TCP-SYN Flooding Attacks", *International Journal of Computer Applications*, Volume 6–No.6, pp.12-15, September 2010.

[11] T. Nakashima and S. Oshima, “A detective method for SYN flood attacks”, *First International Conference on Innovative Computing, Information and Control*, 2006.

[12] D. Nashat,X. Jiang and S. Horiguchi, “Detecting SYN Flooding Agents under Any Type of IP Spoofing”, *IEEE International Conference on eBusiness Engineering*, 2008.

[13] W. Chen and D.-Y. Yeung, "Defending Against TCP-SYN Flooding Attacks Under Different Types of IP Spoofing", *ICN/ICONS/MCL ’06*, IEEE Computer Society, pp. 38-44, April 2006.