ABSTRACT

Effects of DDoS are but not limited to Loss of reputation and Loss of credit which are main concerns. From an industry perspective, all small sized, medium sized and large sized industries are if not in large but get impacted by DDoS Attacks due to exploitation of their resources and not available for genuine use. From a research perspective, storage of old data is required for DDoS mitigation to learn from the patterns of the old attacks on the system.

Implementing honeypots in a business environment may benefit the business in two ways. Firstly, it can provide some form of defense to the real system. The attacker may be diverted away from the actual system into the honeypot. Secondly, the lessons learned from the attacks can be used to build even better defense mechanisms.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER-1

INTRODUCTION

In recent times, small, medium and large sized organizations are at risk of internet attacks. One common attack is the Distributed Denial of Service. It causes resource hijacking and non-accessibility to a business's service. This can cause non-availability and customer experience problems. Many methods exist to counteract such problems, one of the scalable one's is using honeypots.

Systems or networks have resources that are shared among the users who require it. A denial of service attack is a cyber-attack which prevents access or use to required resources. Such an attack might be due to malicious intent. The attack includes variations using techniques such as buffer-overflow, SQL injection, etc. Effects of the attack can be varying such as system crash, inaccessibility of resources, etc.

Distributed denial of service is an extension of denial of service, but limited to a very specific method of attack. DDoS requires the attacker to procure a large number of resources by any means to indirectly carry out the attack. After procuring the attacking resources, requests are sent to the victim. If the victim system has only a limited number of resources that can't be extended, they will run out due to the overwhelming requests from an attacker. Bottlenecks such as in the network will also cause such a problem. Ultimately, a genuine user will not be able to access his/her resources.

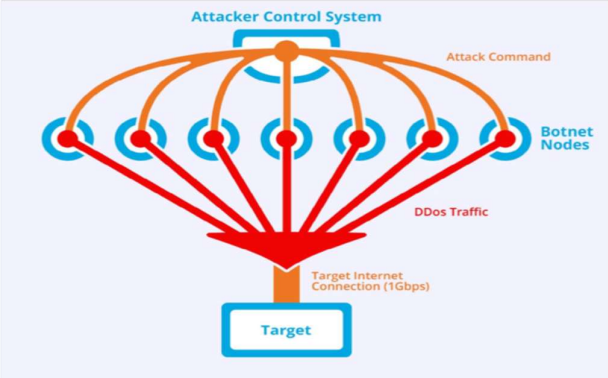

Fig 1.1 DDOS Attack

Honeypots qualify as a type of deception which allows an attacker access to the network but does not let him cause any damage. All malicious traffic is redirected to the honeypot which takes care of what to do with the incoming requests.

Honeypots are used to capture data on malicious requests and better understand easier ways to filter out malicious traffic and maintain better access control.

An entire system must be simulated that can show the working of a honeypot. This requires a tool to send packets while manipulating their parameters to simulate an attack. A firewall must be present to filter out the traffic and a honeypot behind the firewall to record all malicious traffic.

CHAPTER-2

PROBLEM DEFINITION

The aim of the project is Implementation of Honeypot to detect and analyze DDoS attacks which will be used to build even better defence mechanisms. The solution proposed to solve this problem is implementation of low-interaction honeypot with firewall and analysis of malicious traffic. The solution comprises a mechanism to generate an attack along with genuine traffic, differentiate both and record only the malicious traffic.

There are three stages of implementation involved in this problem :

- Developing a DDoS Packet Generator to generate malicious packets. The main use of this component is to simulate an attack on our firewall. Later, once the system is ready, the firewall component along with the honeypot can be deployed into the server to mitigate real DDoS attacks.
- A custom firewall that will detect malicious packets during a DDoS attack and send them over to the honeypot.
- Implementation of Honeypot, where we log the malicious packets to be analysed and improve upon our filtering mechanisms.As a part of honeypot we aim to build a traffic analyser tool to better understand the signature of the attacking traffic.

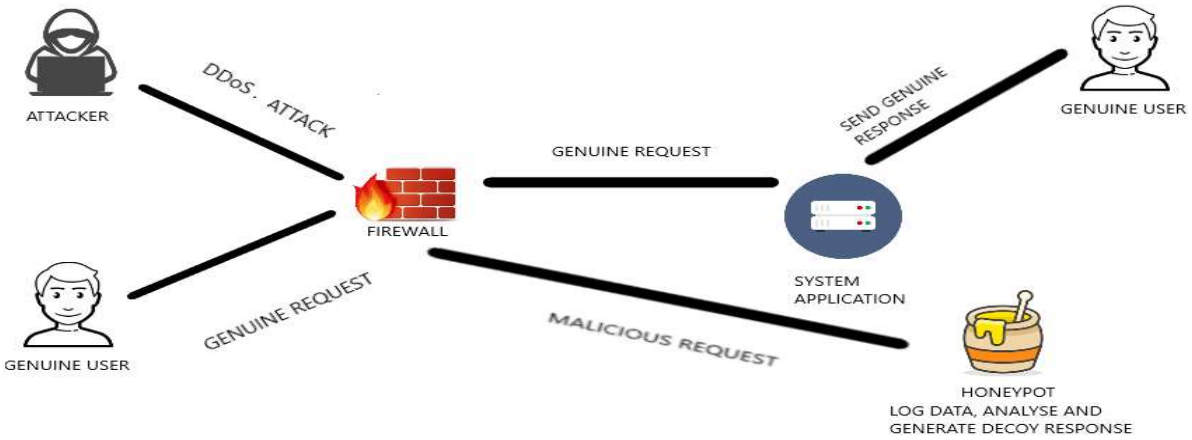

Fig 2.1 High Level Solution To The Problem

CHAPTER-3

LITERATURE SURVEY

Honeypots for Distributed Denial of Service Attacks [1] explains the distinction between DOS and DDOS attacks. It further explains how honeypot systems act as an effective methodology to mitigate such attacks. The paper further explains how honeypot systems can be scaled to cope up with the increased network traffic rate using honeynets. The paper also explains about setting up the honeypot system in an organization environment which has its own DMZ, DNS and LAN. The paper also explains the drawback of the signature based detection techniques.

A Survey of Honeypot Research: Trends and Opportunities [2] is a paper which presents a survey on the research aspect of the honeypot technology for guiding the work of honeypots in the field of research. As a part of the paper they characterize honeypot based on their interactions level, namely low, mid and high interaction honeypots. It also categorizes honeypot based on deployment modes, namely research honeypot and production environment honeypot. This paper gives a brief introduction on the latest research done across the world in the latest time in order to give readers a very good understanding about the relative work in the field on honeypot.

Detecting spoofed packets [3] is a paper which discusses various methodologies which can be employed to detect spoofed network packets. The main focus of the paper was to detect forged ip addresses in a malicious packet. The paper gives a brief introduction on the various packet spoofing attacks such as SYN-flood attack, Smuf, TCP connection spoofing, Bounce scan and Zombie control. The paper further discusses two main categories of detecting spoofed packets, namely routing based methods and non-routing based methods. Under routing methods it further explains how the spoofed packets can be detected by using the help of an internet service provider and monitoring all routers in a subnet. We felt that this technique had a drawback as this cannot be scaled to large networks. Under non-routing methods, the paper discusses techniques like traceroute, os fingerprinting, flow control, and packet retransmission. The paper further explains how these techniques can be employed in intrusion detection systems.

A practical guide to honeypots [4] is a project report submitted at Washington University in St. Louis which gives a brief introduction to honeypot systems and explains in detail about the design methodology which can be used to implement a honeypot. The report further explains design decisions like multithreaded approaches to support connection, creations of protocol filtering and packet logging. The report also explains best practices of implementation from a language prospective.

CHAPTER-4

CUSTOMER REQUIREMENTS SPECIFICATION

4.1 Product Perspective

- This product is very much independent as it will be deployed as part of an entire network. We are demonstrating what happens to malicious behaviour while accepted behaviour is not taken care of. In a real use case, accepted behaviour will be redirected to the organization's internal network.
- Software platforms used for development
  - Debian based machine
  - Pyshark
  - Scapy
  - Splunk
- Hardware used for deployment.
  - Linux machines with RAM greater than 4GB

4.1.1 User Characteristics

General users of this product will be any business which have their data, servers and other resources that have resources accessible via the internet.

Implementing honeypots in a business environment may benefit the business in two ways. Firstly, it can provide some form of defence to the real system. The attacker may be diverted away from the actual system into honeypot.

Secondly, the lessons learnt from the attacks can be used to build even better defence mechanisms.

4.1.2 General Constraints, Assumptions and Dependencies

This section of the CRS shall provide a general description of any other item that will limit the developer's option for designing the system. These can include the following:

- Software limitations
  - Will work on Debian 9 and above.
- Limitations of simulation programs
  - Quality of simulated DDoS packets and user packets is limited to those who have designed them.
- Interfaces to other applications
  - Traffic between systems might be exposed to man in the middle attacks.
- Parallel operations
  - DDoS attack detection is assumed to be happening parallelly while the honeypot's data collection is assumed to happen after an attack detection.
- Criticality of application
  - Failure of this application in a real world scenario will cause DoS outages.
- Safety and security consideration
  - Every part of the system must be secured by using authentication methods to prevent integrity loss of collected data.
  - Ideally this will be covered by the operating system that is used.

4.1.3 Risks

This section of the CRS shall provide a general description of any other item that will limit the developer's option for designing the system. These include the following:

- Filtering out requests which may be due to DDoS may lead to false positives.
- Currently, none of the open sense firewalls provide DDoS traffic redirection.
- Might have to build on top of an already available firewall

4.2 System Architecture

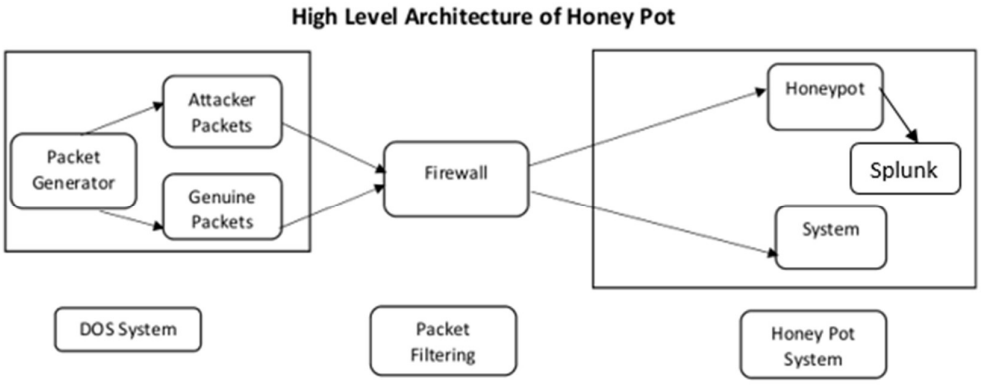

Fig 4.1: High Level Architecture of Honeypot

The packet generator will create genuine and ddos attack packets as required.

These packets will be sent to the firewall.

The firewall will redirect genuine and malicious traffic as per it rules.

All the malicious traffic which is sent to the honeypot is stored as per the required data format.

4.3 Requirements List

4.3.1 Packet Generator

Reqmt #	Requirement
CRS - 1	Functionality of this module includes generating network packets with specified parameters.
CRS - 2	Network parameters include source ip, destination ip, source port, destination port and transport layer.
CRS - 3	Additional parameters include the time interval to be between the packets to be sent, Number of packets to be generated.
CRS - 4	DOS layer built on top of packet generator using above specified parameters

Table 4.1: Packet Generator Requirements

4.3.2 Firewall

Reqmt #	Requirement
CRS - 1	Sniffing for incoming traffic
CRS - 2	Diverting malicious packets to honeypot
CRS - 3	Packet filtering based on the preset rules
CRS - 4	Triggering the packet filter process during a surge in traffic

Table 4.2: Firewall Requirements

4.3.3 Honeypot

Reqmt #	Requirement
CRS - 1	Capture the malicious packet and decode the contents
CRS - 2	Log the required parameters from the packet to a log file for further analysis
CRS - 3	Show relevant real time analysis on the captured logs

Table 4.3: Honeypot Requirements

4.4 External Interface Requirements

4.4.1 Hardware Requirements

- DDoS packet generator
  - Debian 7+ OS
  - 2GB RAM
  - 1 processing core
- Firewall
  - Debian 7+ OS
  - 8GB RAM
  - 4 processing cores
- Honeypot
  - Debian 7+ OS
  - 2GB RAM
  - 1 processing core

4.4.2 Software Requirements

Python

- Python is an interpreted, high-level, general-purpose programming language.
- Python 3.7 is used

Packet Generator - 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 the wire, capture them, match requests and replies, and much more. Scapy can easily handle most classical tasks like scanning, tracerouting, probing, unit tests, attacks or network discovery. It can replace hping, arpspoof, arp-sk, arping, p0f and even some parts of Nmap, tcpdump, and tshark.
- Version used is GPLv2 and is compatible with python2.x and python 3.x

Packet Sniffer - Pyshark

- Python wrapper for tshark, allowing python packet parsing using wireshark dissectors. There are quite a few python packet parsing modules, this one is different because it doesn't actually parse any packets, it simply uses tshark's (wireshark command-line utility) ability to export XMLs to use its parsing.This package allows parsing from a capture file or a live capture, using all wireshark dissectors you have installed. Tested on windows/linux.
- Using this we try to get the required fields from the packet and log these fields for further analysis.
- Pyshark version used is 0.4.2.9

Log Analyser - Splunk

- Splunk captures, indexes, and correlates real-time data in a searchable repository from which it can generate graphs, reports, alerts, dashboards, and visualizations.
- As mentioned above this tool is used to analyse the log files in real time and get reports and alerts.

MYSQL Database

- MySQL is an open-source relational database management system.

MYSQL Connector

- MySQL connector is a self-contained Python driver for communicating with MySQL servers

4.4.3 Communication Interfaces

- Various components of the system like DoS packet generator, firewall and packet sniffer use network ports available in the system to communicate with each other.
- Log Analyser - splunk directly takes input from log files populated by the honeypot and creates required dashboards and alerts.

4.5 User Interfaces

Terminal/CLI to interact with the DDOS Layer:

- In order to generate the packets(both malicious and genuine) the user has to configure a yaml file and run a shell script, which will be provided as a part of the software.
- The results of the ongoing attack are displayed on the terminal itself.

Splunk Dashboard to view malicious packets and reports

- A splunk dashboard will be configured as a part of the software where user can see all the logged malicious packets.
- Dashboard will also support alerts,reports and data visualization.

Terminal/CLI is required to start and end the firewall.

Honeypot is started by using the cli.

4.6 Performance Requirements

Static numerical requirements:-

- **The number of terminals to be supported:** A dashboard where one can analyse the packets.
- **The number of simultaneous users to be supported:** The application is present in the system of the user which is standalone.
- **Sizes of tables and files:** DDoS simulation generates a log file of few 100MBs currently.

Dynamic numerical requirements:-

- **Tasks and the amount of data to be processed within certain time periods for both normal and peak workload conditions:** The firewall module handles filtering of packets flowing continuously from the gateway and the honeypot system should be able to log all the packets which flow into it.
- **Compatibility between heterogeneous environments (for e.g. Open Systems, Mainframes, Mid-range systems, etc.):** Recommended to have a separate simple system to only log and analyse packets reaching this machine. Compatible with any Linux machine with Debian 7+ installed.

- **Interconnection between various networks (LAN, WAN, Internet, etc.):** The system should have good network connection which should not break.

4.7 Special Characteristics

- **Keep specific log or history data sets:** The log files have special names so that even if the system is breached, the log files are not lost.
- **Restrict communications between some areas of a program:** The system implements interfaces where the logic is abstracted.
- **Hardware / Software locks for the product / package:** The system is isolated from common places

4.8 Packaging

The entire software will be packaged, and hosted on the internet and can be downloaded easily by running a bash script and can also be easily configured by following the required steps.

CHAPTER-5

HIGH LEVEL DESIGN

5.1 Design Description

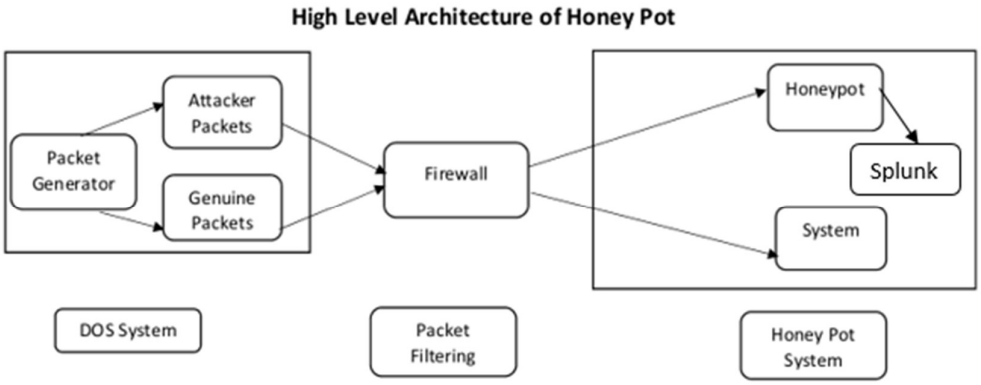

Fig 5.1: High Level Architecture

The system contains 3 Modules namely Packet Generator, Firewall and Honeypot. Each of the modules are described separately in this document.

In brief, The Packet generator module is responsible for generating genuine and malicious packets and directing these packets to a system. System in our environment consists of a firewall and a honeypot. Firewall is responsible for filtering the packets using the predefined rules and passing genuine traffic to the actual system and malicious traffic to honeypot. Honeypot module is responsible for logging the malicious packet and honeypot also has capabilities to analyze the packets.

5.2 Master Class Diagram

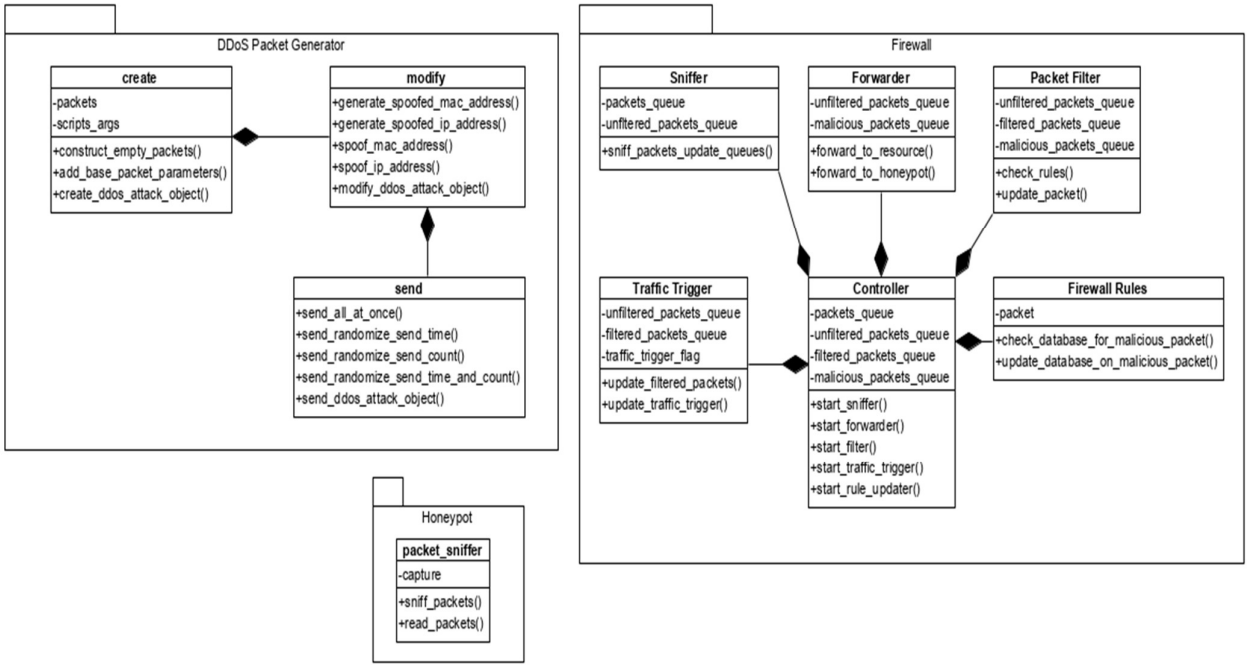

Fig 5.2: Master Class Diagram

5.3 Module 1 - Packet Generator

Generate both genuine and DDoS packets as required.

5.3.1 Description

Functionality of this module includes generating network packets with specified parameters. Network parameters include source ip, destination ip, source port, destination port and network layer protocol. Additional parameters include time interval between the packets, Number of packets to be generated. DOS layer built on top of the packet generator using above specified parameters.

5.3.2 Use Case Diagram

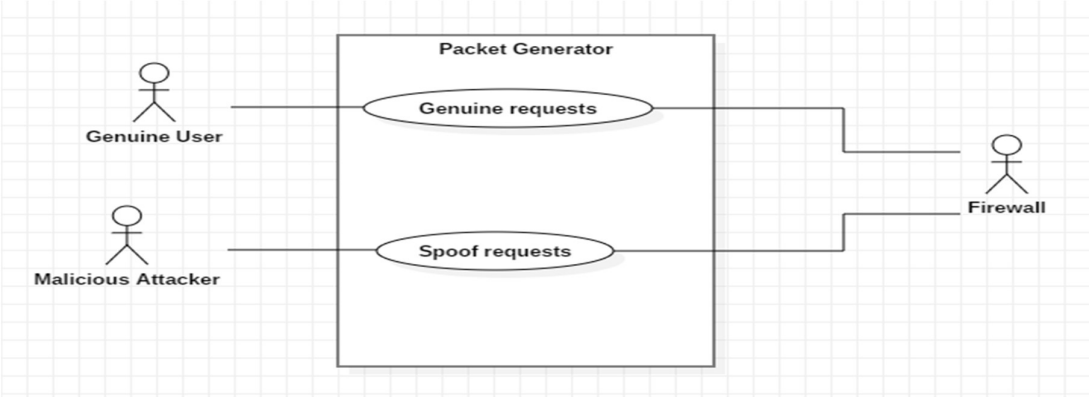

Fig 5.3: Packet Generator Use Case Diagram

Use Case Item	Description
Genuine requests	Generated by genuine users of the system
Spoofed requests	Generated by malicious attackers who want to exploit the system

Table 5.1: Packet Generator Use Case Item

5.3.3 Class Diagram

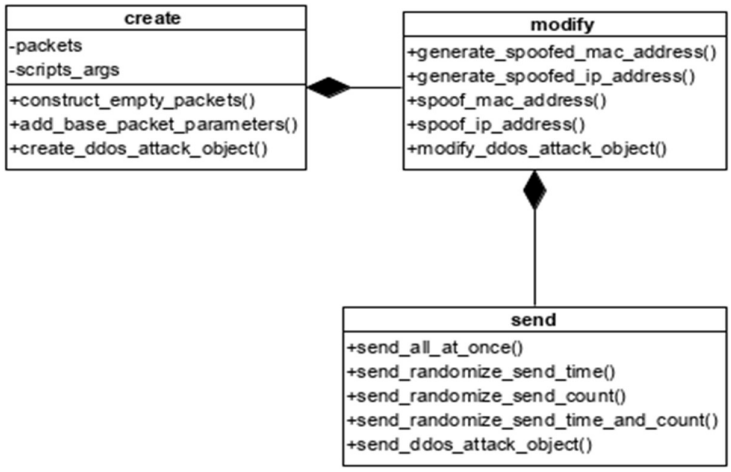

Fig 5.4: Packet Generator Class Diagram

This Module has 3 classes namely Create, Modify and Send.

The Create class creates a DDoS attack object. This class creates a simple DDoS object populated with simple parameters.

The Modify class modifies the DDoS attack object. This class modifies a newly created DDoS packet for the create class.

The Send class sends the modified malicious packet to the firewall.

5.3.4 Sequence Diagram

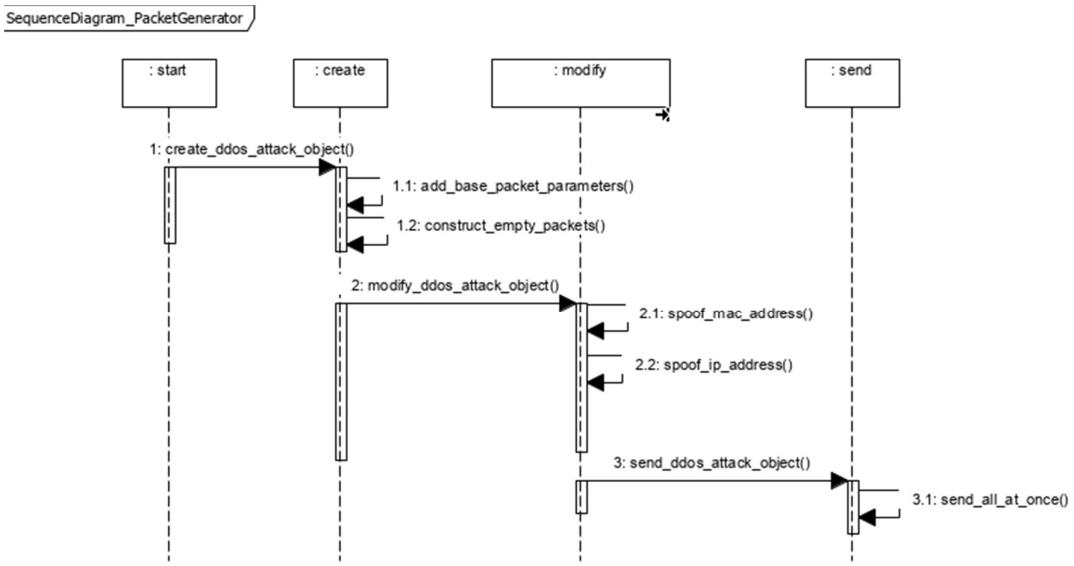

Fig 5.5: Packet Generator Sequence Diagram

5.4 Module 2 - Firewall

5.4.1 Description

The high level functionality of this module would be to determine the genuineness of the request and forward it to the respective location. It contains various processes running simultaneously. A sniffer runs continuously to store the packets that are received. A traffic trigger maintains a count of packets and takes care of initializing the packet filter during DDoS attacks. A packet filter goes through a certain amount of rules and checks for malicious packets. Filtered packets are sent to a backend resource and packets deemed malicious are sent to the honeypot.

5.4.2 Use Case Diagram

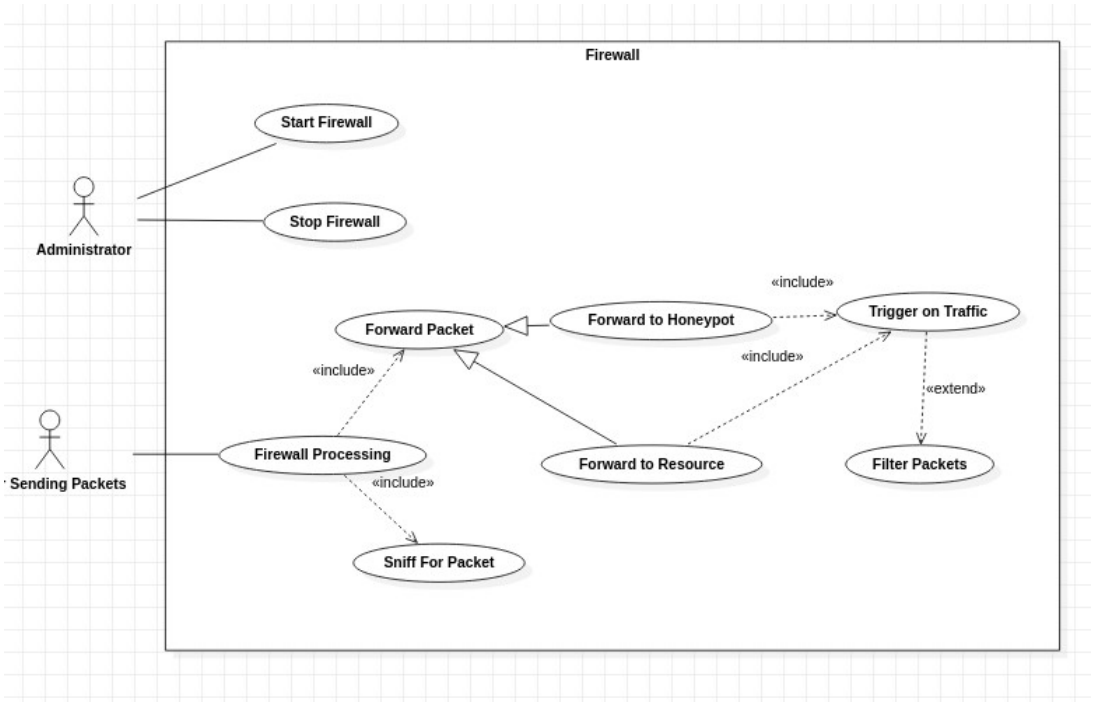

Fig 5.6: Firewall Use Case Diagram

Use Case Item	Description
Start Firewall	Administrator starts firewall
Stop Firewall	Administrator stops firewall
Firewall processing	A packet sent to the firewall is processed
Forward packet	Send the packet to honeypot or other resource
Trigger on Traffic	Traffic trigger is set when DDoS attack occurs
Sniff for packet	Firewall sniffs for packets
Filter packets	Packets is marked malicious or genuine depending on rules

Table 5.2: Firewall Use Case Item

5.4.3 Class Diagram

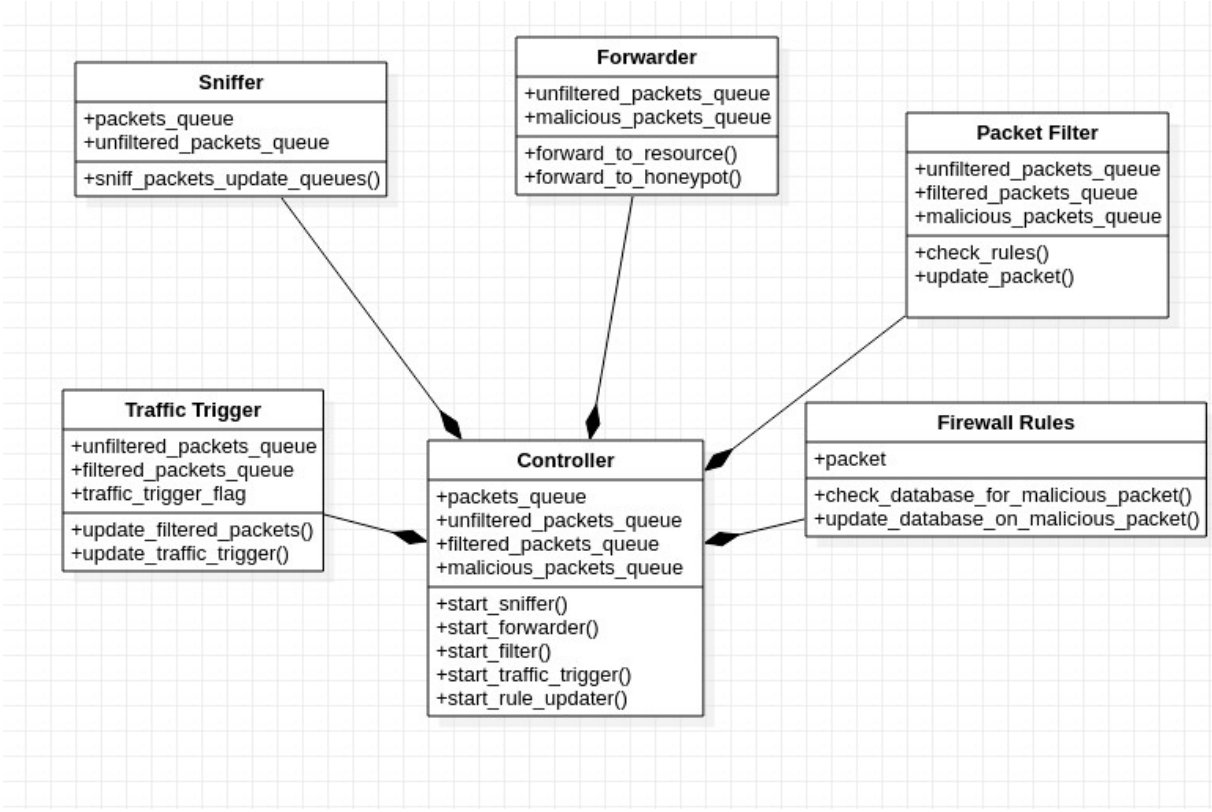

Fig 5.7: Firewall Class Diagram

This module has 6 classes

- Sniffer class's responsibility is to capture incoming packets.
- Forwarder will redirect the packet to appropriate resources.
- Packet filters will filter out malicious packets.
- Traffic trigger will start the second layer of filtering
- Firewall rules are used to determine whether the packet is malicious or not
- Controller initiates the shared memory and starts all other processes.

5.4.4 Sequence Diagram

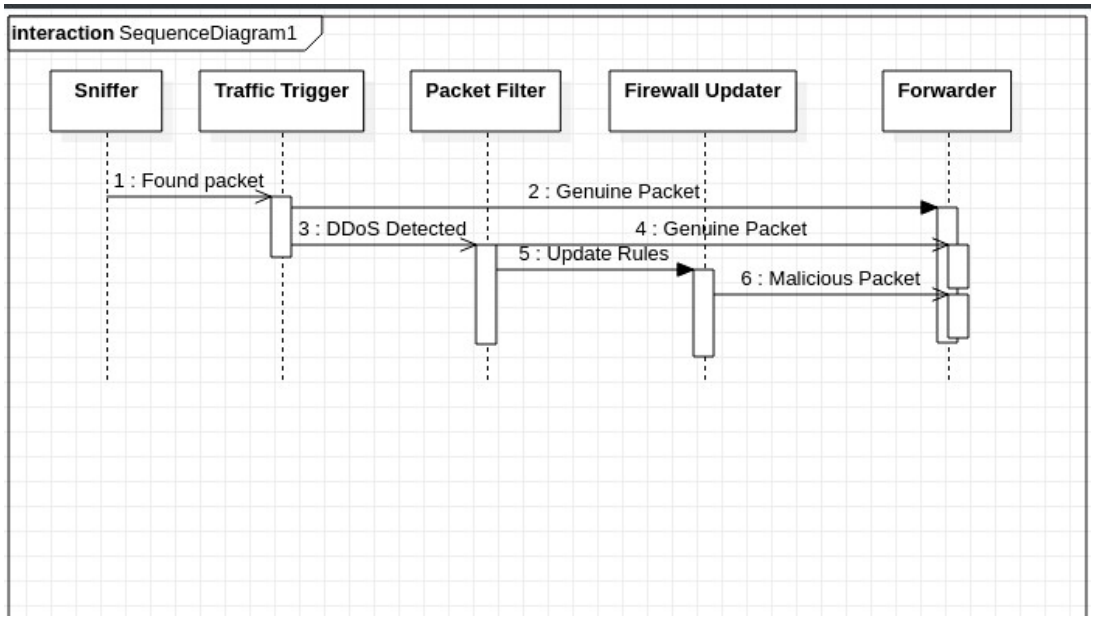

Fig 5.8: Firewall Sequence Diagram

5.5 Module 3 - Honeypot

5.5.1 Description

When high level design is considered, honey pot is made up of two components,namely packet sniffer and packet log analyzer. Packet sniffer is responsible for sniffing the packets transmitted to it by the honeypot and logging these packets appropriately to a file with preset parameters. Packet log analyzer is responsible for picking the logged packets from the file and pushing to splunk indexes for analysing network statistics like source number, IP address and MAC address.

5.5.2 Use Case Diagram

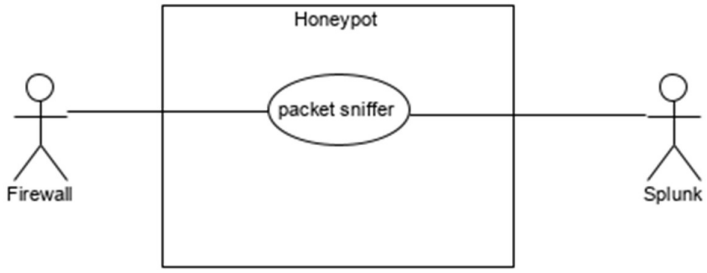

Fig 5.9: Honeypot Use Case Diagram

Use Case Item	Description
packet sniffer	Capture the malicious packets

Table 5.3: Honeypot Use Case Item

5.5.3 Class Diagram

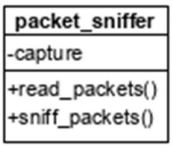

Fig 5.10: Honeypot Class Diagram

This class captures the network packets sent to it by the firewall module and further extracts the required network parameters. These network parameters are logged to a prefixed file for further analysis. Splunk which is a monitoring tool reads from this log file for further analysis.

5.5.4 Sequence Diagram

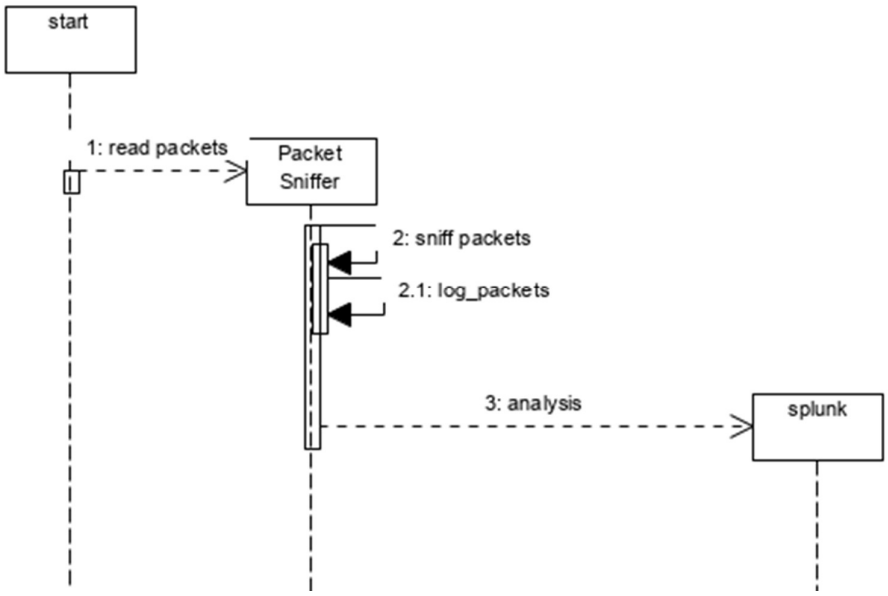

Fig 5.11: Honeypot Sequence Diagram

5.6. User Interface Diagrams

Terminal to interact with the DDOS Layer:

- In order to generate the packets(both malicious and genuine) the user has to configure a yaml file and run a shell script, which will be provided as a part of the software.
- The results of the ongoing attack are displayed on the terminal itself.

Firewall is also run through the CLI. Below is a screenshot of how it is run.

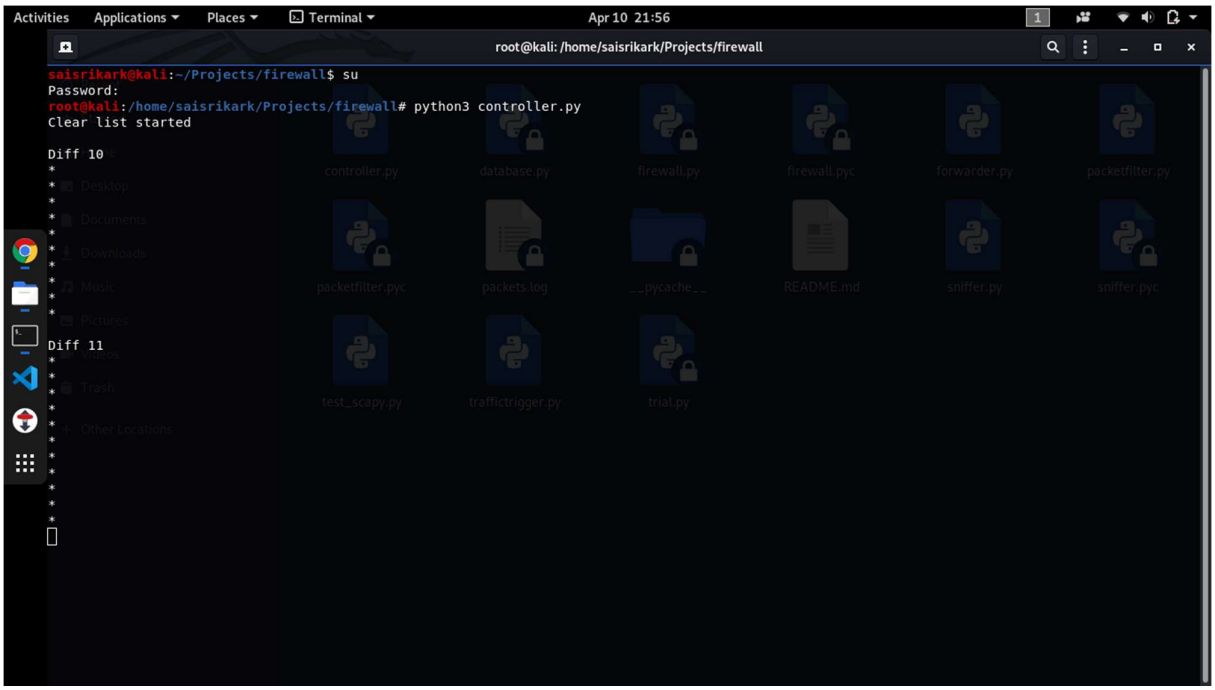

Fig 5.12: Firewall Interface

Honeypot is run through the CLI. Below is a screenshot of how it is run.

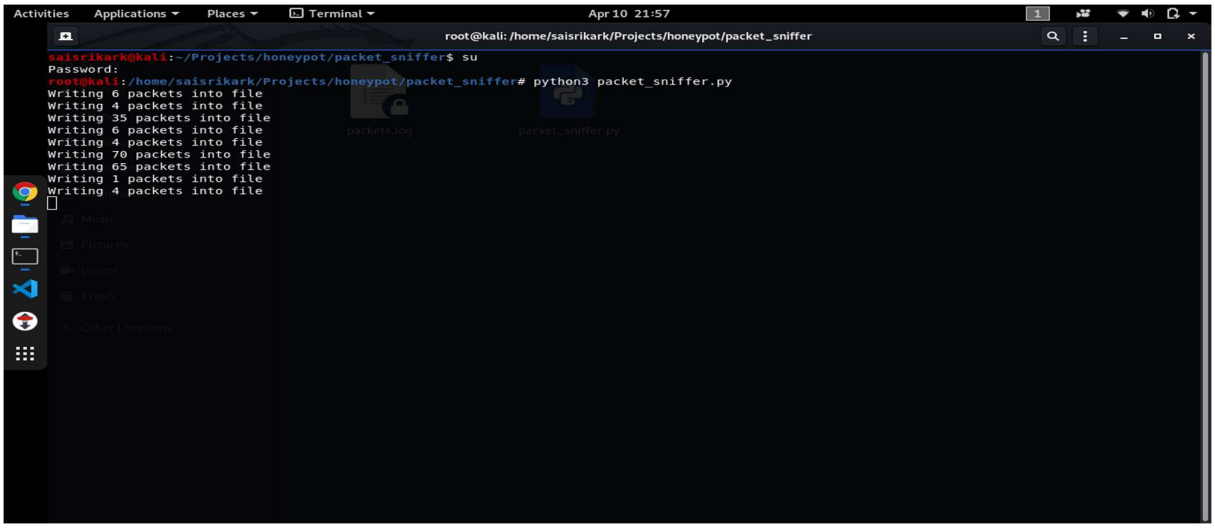

Fig 5.13: Honeypot Interface

Splunk Dashboard to view malicious packets and reports

- A splunk dashboard will be configured as a part of the software where user can see all the logged malicious packets.
- Dashboard will also support alerts,reports and data visualization.

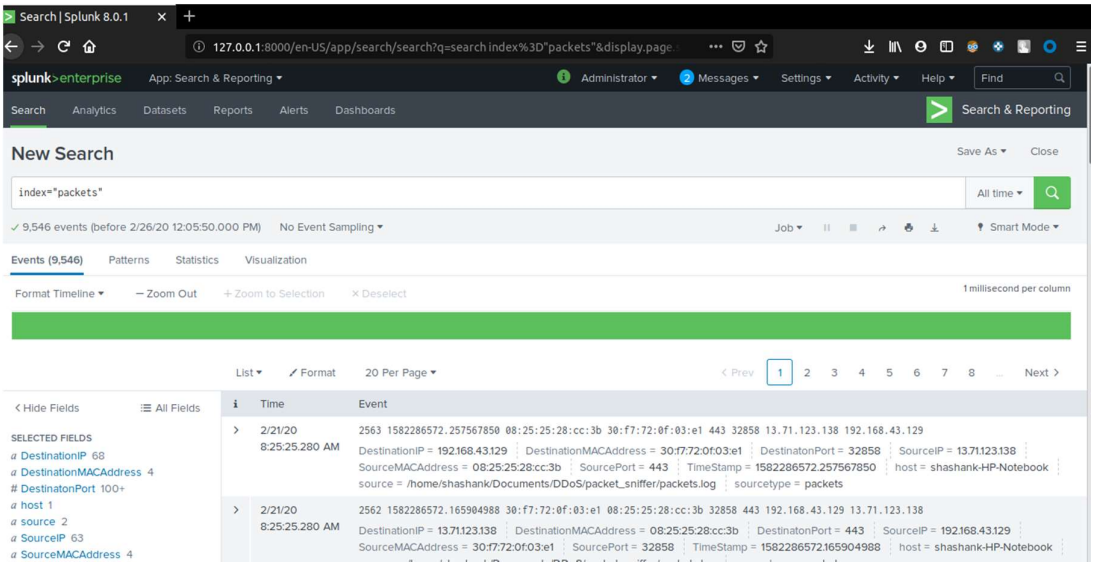

Fig 5.14: Splunk Dashboard

5.7 Report Layouts

The report was built dissecting the components of the projects. The design consists of 3 components, DDoS System, Firewall and Honeypot. The Class Diagram is designed for each component and just of the each class of all the components is mentioned in the Low Level Document of the diagram.

5.8 External Interfaces

As shown in the architecture diagram in the beginning of this document, the software is built of maily 3 modules or components and each module has specific and well defined interface as shown in the diagram to communicate with each other.

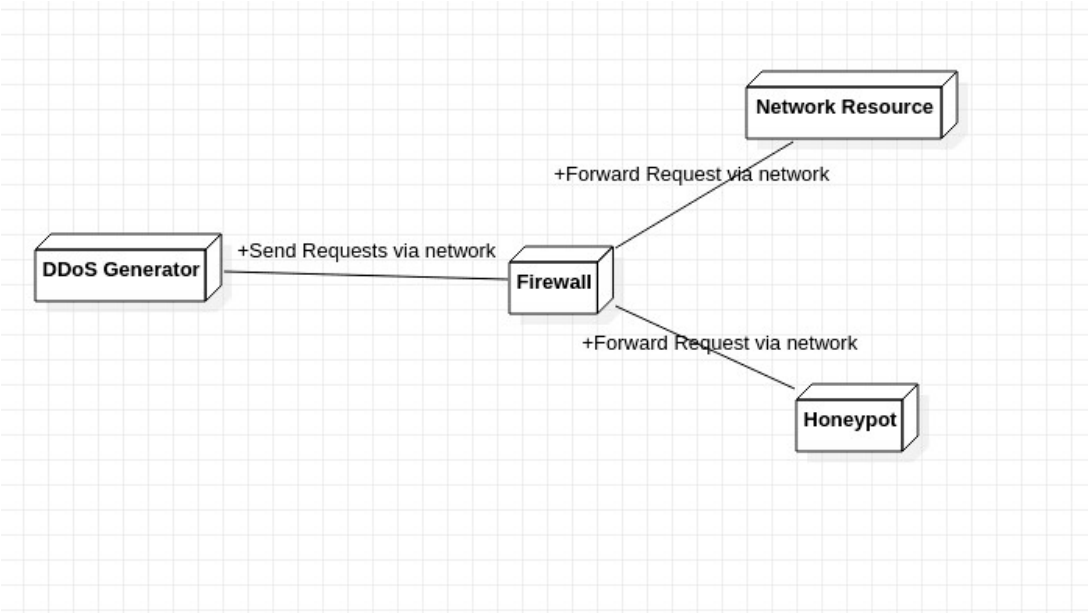

Fig 5.15: External Interfaces

5.9. Help

Documentation for individual components in the readme's of their GitHub repositories will provide enough instruction to control, use and launch them.

5.10 Design Approach

- Design approach followed through the project is a combination of "Component and Connector" and "Decomposition based on functionality".
- At high level we have followed decomposition based on functionality and at a granular level we have followed component and connector design approach
- Decomposition based on functionality helped us to create boundaries at a high level and make the project modular.
- Components and connectors helped us to decompose each boundary further and define interfaces between them for optimal communication.

5.11 Traceability Matrix

CRS Reference Section No. and Name.	HLD Reference Section No. and Name.
4.2.1 Packet Generator	5.1.2 Module 1 - Packet Generator
4.2.2 Firewall	5.1.3 Module 2 - Firewall
4.2.3 Honeypot	5.1.4 Module 3 - Honeypot

Table 5.4: CRS Vs HLD Traceability Matrix

CHAPTER-6

LOW LEVEL DESIGN

6.0 Design Description

This section describes the design with respect to functional modules.

6.1 DDoS

6.1.1 Create

- **Class Description**

This class creates a DDoS attack object. This class creates a simple DDoS object populated with simple parameters.

- **Data Members**

Data Type	Data Name	Access Modifiers	Initial Value	Description
array	packets	Private	NULL	Will hold all the packets.
array	script_args	Private	NULL	Will specify the parameters for the packets.

Table 6.1: Data members of DDOS class

- **Methods**

1. **construct_empty_packets()**

   The construct_empty_packets() method will construct packets based on the scripts_args based parameters and update to the packets array.

2. **add_base_packet_parameters()**

   The add_base_packet_parameters() method will populate the packet with the desired protocol. (Protocols covered: IP, TCP, UDP, ICMP)

3. **create_ddos_attack_object()**

   The create_ddos_attack_object() method will be invoked to create a new object as and when required.

6.1.2 Modify

- **Class Description**

This class modifies the DDoS attack object. This class modifies a newly created DDoS packet for the create class.

- **Methods**

1. **generate_spoofed_mac_address()**

   The generate_spoofed_mac_address() method will spoof the mac_address of the packet.

2. **generate_spoofed_ip_address()**

   The generate_spoofed_ip_address() method will spoof the ip_address of the packet.

3. **spoof_mac_address()**

   The spoof_mac_address() method will internally call generate_spoofed_mac_address() on the packet.

4. **spoof_ip_address()**

   The spoof_ip_address() method will internally call generate_spoofed_ip_address() on the packet.

5. **modify_ddos_attack_object()**

   The modify_ddos_attack_object() method will modify the packet as per the requirement.

6.1.3 Send

- **Class Description**

This class sends the modified malicious packet to the firewall.

- **Methods**

1. **send_all_at_once()**

   The send_all_at_once() method will send all the packets that have been generated with the required number of packets.

2. **send_randomize_send_time()**

   The send_randomize_send_time() method will send packets at every specified interval of time

3. **send_randomize_send_count()**

   The send_randomize_send_count() method will send a count number of packets.

4. **send_randomize_send_time_and_count()**

   The send_randomize_send_time_and_count() method will send a count number of packets at every specified interval of time.

5. **send_ddos_attack_object()**

   The send_ddos_attack_object() method will actually send the packets to the firewall.

6.2 Firewall

6.2.1 Sniffer

- **Class Description**

Sniffs for packets at a particular specified interface. Moves the sniffed packets to an unfiltered queue and occasionally logs received packets

- **Data Members**

Table 6.2: Data members of sniffer class\

Data Type	Data Name	Access Modifiers	Initial Value	Description
queue	packets_queue	Public	NULL	General purpose queue to store packet data
queue	unfiltered_packets_queue	Public	NULL	General purpose queue to store unfiltered packet

- **Methods**

1. **sniff_packets_update_queue()**

   Check whether any packets have been captured, if yes update the shared memory queues.

6.2.2 Forwarder

- **Class Description**

Takes the filtered and malicious packets from their respective queues in shared memory. Forwards the packets to the backend and honeypot respectively.

- **Data Members**

Data Type	Data Name	Access Modifiers	Initial Value	Description
queue	filtered_packets_queue	Public	NULL	General purpose queue to store filtered packet data
queue	malicious_packets_queue	Public	NULL	General purpose queue to store malicious packet

Table 6.3: Data member of Forwarder class

- **Methods**

1. **forward_to_resource()**

   Redirect the packet to the respective backend resource.

2. **forward_to_honeypot()**

   Redirect the packet to the configured honeypot.

6.2.3 Packet Filter

- **Class Description**

Polls continuously for a trigger flag to be set. When set begins the packet filter process for packets in the unfiltered queue. Checks the packet against rules in the firewall and updates the packet data into the logging database. Updated the firewall rules occasionally when a trigger is hit while updating.

- **Data Members**

Data Type	Data Name	Access Modifiers	Initial Value	Description
queue	filtered_packets_queue	Public	NULL	General purpose queue to store filtered packet data
queue	unfiltered_packets_queue	Public	NULL	General purpose queue to store unfiltered packet
queue	malicious_packets_queue	Public	NULL	General purpose queue to store malicious packet data

Table 6.4: Data members of Packet Filter class

- **Methods**

1. **check_rules()**

   Check a packet against all rules in the firewall.

2. **update_packets()**

   Update a particular packet as malicious or filtered.

6.2.4 Traffic Trigger

- **Class Description**

Keeps a count on the number of packets to set a trigger in case of DDoS. If there is no trigger, marks the packets as filtered.

- **Data Members**

Data Type	Data Name	Access Modifiers	Initial Value	Description
queue	filtered_packets_queue	Public	NULL	General purpose queue to store filtered packet data
queue	unfiltered_packets_queue	Public	NULL	General purpose queue to store unfiltered packet
boolean	traffic_trigger_flag	Public	False	Trigger to start second layer packet check

Table 6.5: Data members of Traffic Trigger class

- **Methods**

1. **update_filtered_packets()**

   When there is no surge in packets update them into the filtered queue.

2. **update_traffic_trigger()**

   When there is a surge in traffic, set the traffic trigger flag else unset.

6.2.5 Controller

- **Class Description**

Main class that starts all other processes

- **Data Members**

Data Type	Data Name	Access Modifiers	Initial Value	Description
queue	packets_queue	Public	NULL	General purpose queue to store packet data
queue	unfiltered_packets_queue	Public	NULL	General purpose queue to store unfiltered packet
queue	filtered_packets_queue	Public	NULL	General purpose queue to store filtered packet data
queue	malicious_packets_queue	Public	NULL	General purpose queue to store malicious packet data
boolean	traffic_trigger_flag	Public	False	Trigger to start second layer packet check

Table 6.6: Data members of Controller class

- **Methods**

1. **start_sniffer()**

   Start the sniffer process that captures all the packets.

2. **start_forwarder()**

   Start the forwarder process that sends packets to their respective destinations.

3. **start_filter()**

   Start the filter process to find malicious packets.

4. **start_traffic_trigger()**

   Start the traffic trigger process that ensures that packets are filters.

6.2.6 Firewall Rules

- **Class Description**

Has functionality to check whether packets are malicious, functionality to update firewall rules and functionality to add packet data to the logging database.

- **Data Members**

Data Type	Data Name	Access Modifiers	Initial Value	Description
Dictionary	packet	Private	NULL	Packet data represented as an object/dictionary

Table 6.7: Data members of Firewall Rules class

- **Methods**

1. **check_database_for_malicious_packets()**

   Check database containing data on malicious packets for any similarity of current packet.

2. **update_database_on_malicious_packets()**

   Update the database that keeps a log of all incoming packets with the current packet and check for any triggers indicating malicious packets on updating.

6.3 Honeypot

6.3.1 Packet Sniffer

- **Class Description**

This module captures the malicious packets filtered by the firewall component which will be later picked up by Splunk for Analysis.

- **Data Members**

Data Type	Data Name	Access Modifiers	Initial Value	Description
array	capture	Public	NULL	Sniffs the malicious packets filtered by the firewall

Table 6.8: Data members of Packet Sniffer class

- **Methods**

1. **sniff_packets()**

   The sniff_packets() method will sniff the packets filtered by a firewall and sent to the honeypot.

2. **read_packets()**

   The read_packets() method will read the packets sniffed and write it to Splunk for analysis.

6.4 Traceability Matrix

CRS Reference Section No. and Name	DESIGN / HLD Reference Section No. and Name	LLD Reference Section No. Name
4.2.1 Packet Generator	5.1.2 Module 1 - Packet Generator	6.1 DDoS
4.2.2 Firewall	5.1.3 Module 2 - Firewall	6.2 Firewall
4.2.3 Honeypot	5.1.4 Module 3 - Honeypot	6.3 Honeypot

Table 6.9: CRS Vs HLD Vs LLD Traceability Matrix

CHAPTER-7

TEST STRATEGY AND TEST PLAN

7.1 Introduction

This Test Plan specifies the strategy being followed for Testing of the project and sample test cases being written for the project.

7.2 Testing Model

Since the application built in this project consists of components, it needs component based testing. So, we emphasize on agile software testing models to focus on testing components independently and then an integration test followed by testing of the entire system.

7.3 TESTING TYPES BEING USED

7.3.1 Different types of tests which were planned to run

7.3.1.1 Unit Testing

This involved testing various individual components in our application like generating malicious packets using Scapy, running firewalls and logging packets in the Honeypot machine. All these components were tested independently to make sure that these components behave correctly by checking the algorithms, Data structures, boundary conditions and error handling. These tests were run by the developer.

7.3.1.2 Integration Testing

We opted for the Bottoms up integration approach for integrating our components to verify the

interactions between our components, this involved creating interactive interfaces where our components could easily interact with each other.

7.3.1.3 System Testing

This tests a completely integrated application to verify that it is compliant with its specified requirements. Various tests were run as a part of System testing like deploying these components on various machines and checking for its correct behavior, sanity testing to check if the new bugs were fixed, Stress test to check if the firewall was able to correct distinguish between genuine packets and malicious packets and correctly forward it accordingly.

7.3.2 Phases of the LifeCycle and the V&V done for each phase in the project

7.3.2.1 Requirement Engineering phase

Designed use-case of customer needed tests for the requirement specification. Created a detailed Customer Requirements Specification document for customers using the application. These documents went through a tracing mechanism from the requirement document to validate if we were on the right track.

7.3.2.2 Design phase

Continuously monitored the designs and architecture which enable us to meet the user defined requirements and fixed up on architecture which helped in reducing the development time and improved productivity

7.3.2.3 Coding phase

Constantly kept checking for consistency in implementation with respect to previous documents. Arranged frequent code reviews and code walkthrough to verify the proper functioning and interaction between the components. Used Software configuration management tools like GIT for validation.

7.3.2.4 Testing Phase

Execution of Unit, Integration and System testing were done and checked if the apps met the Test adequacy criteria. Logged the test results to check if the apps behaved as mentioned in the CRS.

7.3.3 Set out the Test adequacy criteria for your project

- 85% of the line of code have gone through sanity check
- 75% of the branch is covered
- 88% of condition coverage
- 90% of the path coverage

7.4 Test Cases

7.4.1 Test Case 1

- **Objective/functionality being tested**

  Generating malicious packets

- **Procedure to be followed**

  Run the attack.py file by passing the required values for the scapy parameters like no of packets to be generated, network protocol of the packet and other parameters.

- **Test Environment/Data to be used**

  Debian 9 and above linux machines.

- **Expected Result**

  Generate packets as required.

7.4.2 Test Case 2

- **Objective/functionality being tested**

  Firewall rules filter the packets and forward them correctly

- **Prerequisite**

  Must receive both genuine and malicious packets

- **Procedure to be followed**

  Run the controller.py file

- **Test Environment/Data to be used**

  Machine with high CPU capacity and good processing speed.

- **Expected Result**

  Accept both kinds of packets and separate them out

7.4.3 Test Case 3

- **Objective/functionality being tested**

  Honeypot machine accept the packets filtered by the firewall and log it for analysis

- **Prerequisites**

  Splunk configured on the machine for generating visual analysis

- **Procedure to be followed**

  Run the packet_sniffer.py file

- **Test Environment/Data to be used**

  Machine with good storage capacity and high processing speed

- **Expected Result**

  Generate visualizations in real-time

7.4.4 Test Case 4

- **Objective/functionality being tested**

  Packet Generator, Firewall and Honeypot components to work collectively.

- **Prerequisites**

  A setup of machines with necessary components deployed in them.

- **Procedure to be followed**

  Run the components in the machines

- **Test Environment/Data to be used**

  A Local Area Network of multiple workstations with high processing speed and good CPU and Storage Capacity.

- **Expected Result**

  The components perform the required behaviour.

CHAPTER-8

IMPLEMENTATION / PSEUDO CODE

8.1 Generating DDoS Packets using Scapy libraries

The first stage of implementation involves generating DDoS Packets using Scapy libraries. Following is a program that starts the **DDoS Packet Generator** Package and generates a DDoS Attack.

```
import argparse
import sys

from dependencies import *

def retrieve_arguments():
    # Retrieve all the arguments from the yaml file
    ap = argparse.ArgumentParser()
    ap.add_argument("FILE", help="Enter name of the file")
    YAML_FILE = vars(ap.parse_args())["FILE"]
    f = open(YAML_FILE, "r")
    yaml_data = yaml.load(f, Loader=yaml.FullLoader)
    args = {}
    for dictionary in yaml_data:
        for key in dictionary.keys():
            args[key] = dictionary[key]
    return args

def perform_attack():
    script_args = retrieve_arguments()
    ddos_attack_object = create_ddos_attack_object(script_args)
    modified_ddos_attack_object = modify_ddos_attack_object(
      ddos_attack_object, script_args)
```

```
    send_ddos_attack_object(modified_ddos_attack_object,
     script_args)

if __name__ == "__main__":
    perform_attack()
```

8.2 Filtering packets in the Firewall using preset rules

The next stage of implementation involves filtering packets that come to the gateway of the administrative system using the Firewall rules and separating them into genuine packets by forwarding it to the server and malicious packets to the honeypot. The following code is the program which controls the Firewall component.

```
from time import sleep
import multiprocessing
import daemon
import packetfilter
import sniffer
import traffictrigger
import forwarder

def controller():

    ipc_variables = {} # All queues are stored here

    packet_queue = multiprocessing.Queue() # All packets sniffed are stored here
    filtered_packets_queue = multiprocessing.Queue() # Queue that stores all filtered packets
    unfiltered_packets_queue = multiprocessing.Queue() # Queue that stores all unfiltered packets
    malicious_packets_queue = multiprocessing.Queue() # Queue that stores all malicious packets
    packet_check_flag = multiprocessing.Value('b', False) # Initially setting to 0 to not check for
value
```

```
    ipc_variables["packet_queue"] = packet_queue
    ipc_variables["filtered_packets_queue"] = filtered_packets_queue
    ipc_variables["unfiltered_packets_queue"] = unfiltered_packets_queue
    ipc_variables["malicious_packets_queue"] = malicious_packets_queue
    ipc_variables["packet_check_flag"] = packet_check_flag

    packetfilter_process = multiprocessing.Process(target=packetfilter.packetfilter_controller,
args=(ipc_variables,))
    sniffer_process = multiprocessing.Process(target=sniffer.sniffer_controller,
args=(ipc_variables,))
    traffictrigger_process = multiprocessing.Process(target=traffictrigger.traffictrigger_controller,
args=(ipc_variables,))
    forwarder_process1 = multiprocessing.Process(target=forwarder.forwarder_controller,
args=(ipc_variables,))
    forwarder_process2 = multiprocessing.Process(target=forwarder.forwarder_controller,
args=(ipc_variables,))

    packetfilter_process.start()
    sniffer_process.start()
    traffictrigger_process.start()
    forwarder_process1.start()
    forwarder_process2.start()

if __name__ == "__main__":
    controllerprocess = multiprocessing.Process(target=controller)
    controllerprocess.start()
```

8.3 Logging Malicious Packets in the Honeypot system for Analysis

This stage involves the logging of malicious packets that come to the honeypot machine for analysis by Splunk. The following code sniffs the packets that are sent to the machine and writes them to a log file.

```
import pyshark
import threading
import time
import signal
import sys

capture = pyshark.LiveCapture(interface = 'wlp19s0')
j = 0
turn_off_lock = threading.Lock()
turn_off_flag = False

def signal_handler(sif, frame):
    print("\n")
    print("Command Received To Terminate The Program")
    turn_off_lock.acquire()
    global turn_off_flag
    turn_off_flag = True
    turn_off_lock.release()

def write_into_file(capture_list,start_index,end_index):
    for i in range(start_index,end_index):
        try:
            global j
            packet_log = str(j) + " " + str(capture_list[i].sniff_timestamp) + " "
            j = j + 1
            try:
                packet_log = packet_log + capture_list[i]['eth'].src + " " + capture_list[i]['eth'].dst + " "
            except:
                packet_log = packet_log + "NA" + " " + "NA" + " "
            try:
```

```
            packet_log = packet_log +
capture_list[i][capture_list[i].transport_layer.lower()].srcport + " " +
capture_list[i][capture_list[i].transport_layer.lower()].dstport + " "
        except:
            packet_log = packet_log + "NA" + " " + "NA" + " "
        if("ip" in capture_list[i]):
            packet_log = packet_log + capture_list[i]['ip'].src + " " + capture_list[i]['ip'].dst + "\n"
        else:
            packet_log = packet_log + capture_list[i]['ipv6'].src + " " + capture_list[i]['ipv6'].dst +
"\n"
        fp = open("packets.log", "a")
        fp.write(packet_log)
        fp.close()
    except:
        fp = open("packets.log","a")
        fp.write("Failed To Log The Packet\n")
        fp.close()

def read_packets():

  while(len(capture) == 0):
    time.sleep(5)

  older_count = 0
  while(True):
    new_count = len(capture)
    if(new_count != older_count):
      print("Writing packets from " + str(older_count + 1) + " to " + str(new_count))
      write_into_file(capture,older_count,new_count)
      older_count = new_count
      time.sleep(10)
    turn_off_lock.acquire()
```

```
        if(turn_off_flag):
            turn_off_lock.release()
            break
        turn_off_lock.release()
    print("Exiting packet reading")

def sniff_packets():
    capture.sniff()

t1 = threading.Thread(target=read_packets)
t1.start()
t2 = threading.Thread(target=sniff_packets)
t2.start()
signal.signal(signal.SIGINT, signal_handler)
t1.join()
sys.exit()
```

CHAPTER-9

RESULTS AND DISCUSSION

The exploration and implementation of the problem statement "Mitigating DDOS attacks using honeypot" resulted in the following solutions:

- Development of firewall with predefined rules for packet filtering. Firewall implementation led to development of three modules, namely packet sniffer, packet filter and packet transmitter.

- Development of honeypot system. This led to implementation of three modules, namely packet sniffer,arguments extractor and packet logger. Honeypot implementation also comes with a packet log analyzer tool. This is developed on top of splunk software according to problem requirements.

- In order to mimic the DDOS attack environment we have also built a packet generator module which is responsible for generating both genuine and malicious packets.

CHAPTER-10

CONCLUSIONS

The outcome of the project, firewall and honeypot system can be used to mitigate DDOS attacks in real time by successfully deploying the packages on appropriate operating systems.

Firewall tool successfully filters out the malicious packets and transmits them to the honeypot system. Firewall comes with preset rules for packet filtering. Firewall can always be extended by the user by adding new rules in the appropriate package.

Honeypot successfully collects the packages transmitted to it by the firewall and logs them to appropriate files after extracting the required network arguments from the packets. Packet analyzer which comes as an additional component of honeypot helps the end user in better understanding the signature of the attacking network traffic.

Firewall can always be customized in order to extend its capabilities based on user requirements by adding additional packet filtering rules in appropriate packages.

CHAPTER-11

FUTURE ENHANCEMENTS

The project aimed at building a low interaction honeypot with a firewall with basic rules for packet filtering. The further enhancement of the projects would be to implement more robust honeypots like mid interaction or high interaction honeypots. Honeypot developed in this project log the basic network parameters. The future enhancement could be to log more advanced network parameters and use some statistical analysis methods to deduce network signature.

Firewall implemented in this project mostly includes rules which target the IP layer of the network packets. The future enhancement to this may include to come up with the rules which target the transport layer of the network packets.

Log analyser implemented in this project provide an overview about the network traffic being implemented. This can be further enhanced to provide real time alert and notification to the network administrator.

REFERENCES / BIBLIOGRAPHY

[1] Honeypots for Distributed Denial of Service Attacks, Nathalie Weiler Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology ETH Zurich, Switzerland, Proceedings of the Eleventh IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'02)

[2] A Survey of Honeypot Research: Trends and Opportunities, Ronald M. Campbell School of Computing University of South Africa Johannesburg, South Africa. Keshnee Padayachee, Institute for Science and Technology Education University of South Africa Pretoria, South Africa. Themba Masombuka School of Computing University of South Africa Johannesburg, South Africa, The 10th International Conference for Internet Technology and Secured Transactions (ICITST-2015).

[3] Detecting Spoofed Packets Steven J. Templeton, Karl E. Levitt Department of Computer Science U.C. Davis

[4] A Practical Guide to Honeypots Eric Peter, epeter@wustl.edu and Todd Schiller, tschiller@acm.org, project report present at Washington University in St. Louis.

- https://hackertarget.com/cowrie-honeypot-analysis-24hrs/
- https://www.incibe-cert.es/en/blog/honeypot-tool-know-your-enemy
- https://www.bitforestinfo.com/2017/01/how-to-write-simple-packet-sniffer.html
- https://github.com/paralax/awesome-honeypots
- https://www.cse.wustl.edu/~jain/cse571-09/ftp/honey/
- https://hackertarget.com/cowrie-honeypot-analysis-24hrs/