

# Identification of Malicious SSH Network Activities using HoneyNet

**THESIS**

Submitted in partial fulfillment  
of the requirements for the degree of  
**DOCTOR OF PHILOSOPHY**

by

**GOKUL KANNAN SADASIVAM**

**ID. No. 2013PHXF0201H**

Under the Supervision of  
**Prof. Chittaranjan Hota**

&

Under the Co-supervision of  
**Dr. Anand Bhojan**



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI**

**2020**

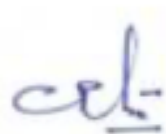
# BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

## CERTIFICATE

This is to certify that the thesis entitled \_\_\_\_\_

\_\_\_\_\_ Identification of Malicious SSH Network Activities using HoneyNet \_\_\_\_\_

and submitted by Gokul Kannan Sadasivam ID No 2013PHXF0201H for award of Ph.D. of the Institute embodies original work done by him under my supervision.



**PROF. CHITTARANJAN HOTA**

Professor,  
Department of Computer Science  
and Information Systems,  
BITS Pilani, Hyderabad Campus  
Hyderabad, Telangana - 500078



**DR. ANAND BHOJAN**

Senior Lecturer,  
School of Computing,  
National University of Singapore,  
Singapore

Date: 12-December-2020

Date: 12-December-2020

Dedicated to my wife, my son, and my parents.

## Acknowledgements

There are several personalities who guided me during my PhD programme. I would like to mention their contributions and take this opportunity to thank them.

I am extremely grateful to my supervisor Prof. Chittaranjan Hota for providing his valuable guidance, encouragement, and support. He guided me not only in research but in building my character to be an independent, vigorous, and aggressive researcher. He encouraged me in several occasions pointing out the mistakes in my research methodology. His support in providing the hardware resources helped me to collect the data, without which my research would have been impossible.

My co-supervisor Dr Anand Bhojan also played a pivotal role in my career. Without his advice, I would not have been able to endure the hard times in my PhD. I also thank him for his profound research inputs and his discussions over the social media (Skype, Google Duo, WhatsApp). His insights have polished my work significantly.

Nevertheless, the questions and suggestions made by my Doctoral Advisory Committee (DAC) members, Prof. N. L. Bhanumurthy and Prof. Tathagata Ray, have immensely directed me in the right research path. I am always grateful to them for their valuable time spend in the PhD seminars. Prof. G Geethakumari (ex-DAC member) also provided some research inputs for a short period of time.

I thank the network administrators for configuring the university routers (in my experimental setup). I also thank my colleagues in helping me with the registration of PhD courses and other PhD activities that involve the university.

Even though several people have directly and indirectly contributed to my thesis, I am solely responsible for its content, contributions, results, or omissions in it.

# Abstract

The objective of this thesis is to identify malicious SSH (Secure Shell) activities using honeypot traffic. The malicious activities are identified using Machine Learning algorithms (supervised learning and unsupervised learning).

The malicious network traffic is obtained from a honeynet. The honeynet captured malicious activities for nearly nine months, and the quantity of network traffic in the pcap file format is 5.5 GigaBytes. Kippo honeypot log files also provided the necessary information for building the models. For convenience, the data set obtained using this experiment is called “HP-Dataset”.

An attack participant refers to the list of client machines (or IP addresses) that have involved in a particular attack (for example, a password guessing attack by a botnet on a particular server). The attack participants are from a particular botnet (forming an instance of an attack). In any period, there would be one or more instances of stealthy distributed SSH password guessing attacks. The problem of identifying the attack participants (i.e., finding the group of attackers who performed a particular attack) in stealthy SSH password guessing attacks is explored. An approach based on merging the TCP flows using the application layer content of packets is discussed. The model provided promising results for the data set “HP-Dataset”. The downside of this model is its inability to differentiate stealthy distributed SSH password guessing attacks from stealthy single-source SSH password guessing attacks.

Next, the problem of identifying stealthy single-source SSH password guessing attacks in honeypot traffic is explored deeply. A clustering mechanism (using DBSCAN algorithm) is used to segregate stealthy single-source SSH password guessing attacks from stealthy distributed SSH password guessing attacks. The

features of the clustering model were obtained through domain knowledge. The model is tested with “HP-Dataset” and a public data set. The validation of the model is done using the silhouette coefficient ( $> 0.7$ ).

The packets of SSH Authentication protocol and SSH Connection protocol are encrypted. The SSH authentication protocol is used for providing login credentials (where password guessing attack occurs). In SSH Connection protocol, commands are executed by an authenticated user. In the peripheral devices (firewall, IDS), it is hard to distinguish these two protocols. In honeynet traffic (pcap files), it is hard to find those attacks that have succeeded in cracking the password and executing commands. Finally, the problem of detecting SSH compromises (SSH traffic using SSH Connection protocol) is studied. An approach based on a machine learning algorithm (J48 decision tree) is devised to detect the compromises. The data set used was “HP-Dataset”. Suitable features were extracted from the network traffic, and performance analysis was done using sensitivity, precision, and F2-score. All three performance metrics were nearly 0.9 for “HP-Dataset”.

# Contents

<b>List of Figures</b>	<b>viii</b>
<b>List of Tables</b>	<b>x</b>
<b>List of Abbreviations</b>	<b>xii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Aims and Objectives . . . . .	1
1.2 Motivation . . . . .	2
1.3 Capturing Network Attacks . . . . .	3
1.4 Networking Background . . . . .	4
1.4.1 Secure Shell (SSH) Protocol . . . . .	4
1.4.2 Static and Dynamic IP Addresses . . . . .	6
1.4.3 Private IP Addresses . . . . .	7
1.4.4 Network Address Translation . . . . .	7
1.4.5 Session . . . . .	8
1.5 SSH Password Guessing Attacks and SSH Compromises . . . . .	9
1.5.1 Security Tools to Detect SSH Password Guessing Attacks . . . . .	9
1.5.2 Other Detection and Prevention Measures . . . . .	10

1.5.3	Non-stealthy SSH Password Guessing Attacks . . . . .	11
1.5.4	Stealthy SSH Password Guessing Attacks . . . . .	11
	Stealthy Single-Source SSH Password Guessing Attacks . . . . .	12
	Stealthy Distributed SSH Password Guessing Attacks . . . . .	12
1.5.5	SSH Compromise . . . . .	15
1.6	Research Problem Statement . . . . .	15
1.7	Methodology . . . . .	16
1.7.1	Supervised Learning . . . . .	17
	Algorithms . . . . .	17
	Evaluation of the Classifier Model . . . . .	20
	Feature Selection in Classification . . . . .	21
	Evaluation Metrics for Classification . . . . .	21
1.7.2	Unsupervised Learning . . . . .	23
	DBSCAN . . . . .	23
	Selection of DBSCAN parameters . . . . .	24
	Evaluation Metric for Clustering: Silhouette Coefficient . . . . .	24
1.8	Scope and Organization of the Thesis . . . . .	25
<b>2</b>	<b>Literature Survey</b>	<b>29</b>
2.1	Introduction . . . . .	29
2.2	Honeynet Architecture . . . . .	30
2.3	Detection of SSH Password Guessing Attacks . . . . .	33
	2.3.1 Non-Stealthy Attacks . . . . .	33
	2.3.2 Stealthy Attacks . . . . .	34
2.4	Segregation of Network Attacks . . . . .	36
2.5	Determining the Attack Participants . . . . .	39



2.6	Detection of SSH Compromises . . . . .	39
2.7	Tabular Summary of Malicious Network Activities and their Countermeasures . . . . .	41
<b>3</b>	<b>Honeynet Architecture</b>	<b>43</b>
3.1	Introduction . . . . .	43
3.1.1	Issues in Deploying a Honeypot . . . . .	44
3.2	Honeypot Tools . . . . .	45
3.2.1	Dionaea . . . . .	45
3.2.2	Glastopf and Snare . . . . .	45
3.2.3	Kippo . . . . .	46
3.2.4	HoneyD . . . . .	47
3.2.5	J-Honeypot . . . . .	47
3.3	Architecture to Collect Malicious Traffic . . . . .	48
3.3.1	Honeynet Architecture . . . . .	48
	Virtual Network . . . . .	50
	Tools to Collect Malicious Traffic . . . . .	51
3.3.2	Network Traffic Collection . . . . .	54
3.3.3	Advantages . . . . .	54
3.4	Pre-processing of Network Traffic and Logs . . . . .	55
3.5	Results and Discussion . . . . .	57
<b>4</b>	<b>Identifying each Stealthy SSH Password Guessing Attack Instances</b>	<b>59</b>
4.1	Introduction . . . . .	59
4.2	Data Pre-processing . . . . .	60
4.2.1	Processing of TCP flow pcap files . . . . .	60

4.2.2	Processing of Kippo log files . . . . .	62
4.3	Source of the Attacks . . . . .	63
4.3.1	Source IP Address . . . . .	63
4.3.2	Source Port Number . . . . .	65
4.4	Secure Shell (SSH) Traffic Analysis . . . . .	66
4.4.1	Login Credentials . . . . .	66
4.4.2	SSH Client Library . . . . .	67
4.4.3	Login Attempts in a TCP flow . . . . .	68
4.5	Model to Segregate Attack Instances . . . . .	68
4.5.1	Extraction of Stealthy SSH Password Guessing Attacks . . . . .	70
4.5.2	Store Specific TCP Flow Information . . . . .	70
4.5.3	Segregation of Attack Instances . . . . .	71
4.6	Results and Discussion . . . . .	72
4.6.1	Results . . . . .	72
4.6.2	Limitations . . . . .	75
4.7	Conclusion . . . . .	75
<b>5</b>	<b>Detection of Stealthy Single-Source SSH Password Guessing At-</b>	
	<b>tacks</b>	<b>77</b>
5.1	Introduction . . . . .	77
5.2	Data Extraction . . . . .	79
5.2.1	Session . . . . .	79
5.2.2	Extraction of Stealthy SSH Password Guessing Attacks . . . . .	82
5.2.3	Stealthy SSH Password Guessing Attacks . . . . .	83
5.3	Data Collection and Feature Analysis . . . . .	83
5.3.1	Number of Login Attempts . . . . .	85

5.3.2	Repetition of Login Attempts . . . . .	87
5.3.3	Port Scanning and Password Guessing . . . . .	90
5.3.4	Nature of TCP Connections . . . . .	91
5.4	Model Fitting of the Data . . . . .	94
5.4.1	Reason for Clustering Model . . . . .	94
5.4.2	Feature Distribution . . . . .	94
5.4.3	Feature Selection . . . . .	97
5.4.4	Min-Max Normalization . . . . .	97
5.4.5	Distance Function . . . . .	98
5.4.6	Clustering . . . . .	99
5.5	Results and Discussion . . . . .	101
5.5.1	Data Model without Feature 3 . . . . .	102
5.5.2	Data Model with Hailmary Dataset . . . . .	104
5.5.3	Online Detection . . . . .	104
5.5.4	Comparison with Existing Works . . . . .	106
5.6	Conclusion . . . . .	109
<b>6</b>	<b>Detection of SSH Compromises</b>	<b>111</b>
6.1	Introduction . . . . .	111
6.2	Data Pre-processing . . . . .	112
6.2.1	Network Traffic . . . . .	112
6.2.2	Class Labels . . . . .	113
6.2.3	Tagging the Pcap Files . . . . .	115
6.3	Design Choices for the Model . . . . .	116
6.3.1	Machine Learning Features . . . . .	116
6.3.2	Choice of Machine Learning Algorithm . . . . .	117

OneR . . . . .	118
Decision Tree (J48) . . . . .	118
PART . . . . .	119
Naive Bayes Algorithm . . . . .	119
Logistic Regression . . . . .	119
Support Vector Machines . . . . .	119
k-Nearest Neighbour . . . . .	120
6.3.3 Performance Analysis . . . . .	120
6.3.4 Feature Selection . . . . .	124
6.4 Machine Learning Model . . . . .	126
6.4.1 Normalize the Data Set . . . . .	126
6.4.2 Chosen Machine Learning Algorithm . . . . .	128
6.5 Real-Time Detection of Severe Attacks . . . . .	129
6.5.1 Network Packet Capture . . . . .	129
6.5.2 Segregation of Packets . . . . .	129
6.5.3 Creation of TCP Flows . . . . .	132
6.5.4 Classification . . . . .	134
6.5.5 Performance of the Classifier . . . . .	134
6.5.6 Limitations . . . . .	134
6.6 Conclusion . . . . .	135
<b>7 Conclusions and Future Scope of Work</b>	<b>137</b>
7.1 Conclusions and Summary of Research Contributions . . . . .	138
7.2 Future Scope of Work . . . . .	141
<b>Bibliography</b>	<b>143</b>

<b>Publications</b>	<b>158</b>
<b>Biographies</b>	<b>160</b>

# List of Figures

1.1	Simple Network (Source: wikipedia)	7
1.2	TCP flows forming a session	8
1.3	Botnet Architecture	12
3.1	Honeynet Architecture	49
3.2	FIN/FIN-ACK packets with a private IP address	56
3.3	Most wanted open ports	57
4.1	Geo-location of IP Addresses	64
4.2	Source port distribution	66
4.3	Flow chart for segregating attack instances	69
5.1	Variation in normalized session count with change in session gap	81
5.2	Persistent Stealthy Attack by the IP address $x_1.x_2.115.55$	84
5.3	Effect of multithreading on the TCP flows	92
5.4	Identification numbers of two TCP flows of $x_1.x_2.70.26$	93
5.5	Clustering Model	95
5.6	Distribution of the features	95
5.7	K-dist plot for normalized data	100
5.8	Silhouette Coefficients for various clusters	103

5.9	Parallel Coordinates Plot from Choi <i>et al.</i> work . . . . .	106
5.10	Parallel Coordinates Plot from “Conti and Abdullah” work . . . . .	107
6.1	Best Feature Set Performance . . . . .	125
6.2	Machine Learning Model . . . . .	127
6.3	J48 Decision Tree . . . . .	128
6.4	Flowchart for Real-Time Detection of SSH Severe Attacks - Thread 1	130
6.5	Flowchart for Real-Time Detection of SSH Severe Attacks - Thread 2	131
6.6	Flowchart for Real-Time Detection of SSH Severe Attacks - Thread 3	133

# List of Tables

1.1	Confusion Matrix . . . . .	22
2.1	Summary of Malicious Network Activities and Countermeasures Techniques . . . . .	41
3.1	Server machine configuration . . . . .	48
3.2	Honeynet Virtual Machines . . . . .	52
3.3	Honeynet Time Periods . . . . .	54
4.1	TCP flow types . . . . .	61
4.2	Busy IP blocks . . . . .	65
4.3	Most used SSH client library . . . . .	67
4.4	Number of login attempts in a TCP flow . . . . .	68
4.5	Segregation of attack instances - data set 1 . . . . .	72
4.6	Segregation of attack instances - data set 2 and data set 3 . . . . .	73
5.1	Parameters of the Distribution . . . . .	96
5.2	Correlation between the features . . . . .	96
5.3	Average value of the features . . . . .	101
5.4	Summary of Comparisons with Existing Works . . . . .	109



6.1	Traffic Classes . . . . .	115
6.2	Confusion Matrix . . . . .	121
6.3	Confusion Matrix for ML algorithms . . . . .	122
6.4	Performance Evaluation . . . . .	123
6.5	Performance with Best Features . . . . .	125

# List of Abbreviations

<b>Term</b>	<b>Definition</b>
ACL	Access Control Lists
CMS	Content Management System
DDoS	Distributed Denial-of-Service attack
DoS	Denial-of-Service attack
DHCP	Dynamic Host Configuration Protocol
DMZ	DeMilitarized Zone
DNS	Domain Name System
ERP	Enterprise Resource Planning
HIDS	Host-based Intrusion Detection Systems
HIHP	High-Interaction HoneyPot
HTTP	HyperText Transfer Protocol
IP	Internet Protocol
ISP	Internet Service Provider
LIHP	Low-Interaction HoneyPot
MIHP	Medium-Interaction HoneyPot
NAT	Network Address Translation

NIDS	Network-based Intrusion Detection Systems
POP3	Post Office Protocol 3
RFI	Remote File Inclusion attacks
RST packet	A TCP packet with the header flag RST set to 1.
SIEM	Security Information and Event Management
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SSH	Secure SHell
SYN packet	A TCP packet with TCP header SYN flag set to 1.
SYN-ACK packet	A TCP packet with the header flag SYN and ACK set to 1.
TLS	Transport Layer Security
XSS	Cross-site scripting

# Chapter 1

## Introduction

### 1.1 Aims and Objectives

The main research aims and objectives of this thesis are given below.

1. Detailed literature survey on the following:
  - (a) The different types of emerging network attacks
  - (b) Server honeypots that help detect intrusions
2. To build a network of honeypots working together to monitor, detect, and collect network attacks.
3. The captured attacks are analyzed using machine learning algorithms for automated classification. Classification outputs are used to generate signatures, which are further utilized to improve a peripheral security system's performance.

## 1.2 Motivation

There are several network attacks on system services [1]. Some of them are Denial-of-Service (DoS) attack, Distributed Denial-of-Service (DDoS) attack, packet forging attack, fingerprinting attack, and password guessing attack.

In a password guessing attack, the attacker tries different username and password combinations to crack the system. In an online password guessing attack, the attacker guesses the server username and password from a remote client machine. Whereas, an offline password guessing is not a network attack and is done on a local computer using the stolen encrypted password database. Hereafter, a “password guessing attack” refers to an “online password guessing attack”.

Most of the authentication systems are password-based. For example, all email systems and bank websites provide password-based authentication as the primary mode of authenticating a user. Hence, a password guessing attack has been in business for attackers for a very long time.

In 2010, SANS Internet Storm Center [2] cautioned the appearance of a password guessing attack and mentioned methods to prevent the system from it. In 2013, GitHub [3] was faced with a severe password guessing attack leading to a leak of several confidential information. The attackers targeted weak passwords of the GitHub account users. McAfee Labs Threat Report [4] published in 2017, states that a brute force attack (i.e., password guessing attack) is one of the top network attacks.

The systems that are compromised by a password guessing attack is used for several purposes. A study conducted by Daniel [5] inferred that compromised servers were used for executing DDoS attacks. Important files stored in the computer can be stolen after the compromise. The system could also be used to host

phishing websites or to spread malware.

### 1.3 Capturing Network Attacks

Honeypot [6] is a software tool that collects network attacks. The value of a honeypot lies with the hackers interacting with it. Honeypot provides an environment in which the hackers perform attacks. Moreover, honeypots are equipped with enough logging capabilities to capture the attacker's sessions. Hence, all traffic that is going in and out of a honeypot is considered malicious. There is a very less chance that the network traffic belongs to a normal user. Based on the level of interaction, honeypots can be classified into various types.

In a low-interaction honeypot (LIHP), the level of interaction is minimal. LIHP only provides limited features and does not allow the download or execution of malware on its system. Attacks cannot damage the operating system or change any of the valuable contents of the hard disk. Medium-interaction honeypots (MIHP) provide relatively more features and replies to most of the attack queries.

LIHP is an emulation of real network services. They are simple to design and develop. It comes as an executable package or can be installed using the source code. As the name says, the amount of interaction with a malicious entity is limited. Some of the well-known low-interaction honeypots are Dionaea [7], Glastopf [8], and HoneyD [9]. MIHP interacts more with an attacker compared to LIHP. Kippo [10] is a MIHP providing a fake SSH service. The capability of the network service is different in all these honeypots (both LIHP and MIHP).

High-Interaction Honeypots (HIHP) are real operating systems with complete services. The level of interaction is usually high because the services will have all the features. Since the honeypot provides full control of the network services,

an attacker can exploit a known/unknown vulnerability in the system. If these honeypots are set up in an enterprise environment, along with production servers, then there is the danger of production systems going down.

A virtual honeypot [11] is a honeypot running on a virtual machine. The advantage of a virtual honeypot is the containment provided by a virtual machine. If an attacker compromises a virtual honeypot, it will not affect the host operating system.

## 1.4 Networking Background

The following networking topics play a key role in the research work done. Hence, they are explained below.

The thesis explores the network attacks on the SSH server. The conversation between a client and an SSH server uses the SSH protocol. The SSH packet information is analyzed and used in this thesis. Hence, an introduction to the SSH protocol is briefly mentioned.

In Chapter 5, an assumption is made to handle the dynamic nature of IP addresses on the Internet. The types of IP addresses (public IP addresses, private IP addresses) and the formation of “Session” is explained below to have a better understanding of the work.

### 1.4.1 Secure Shell (SSH) Protocol

SSH is a network protocol that is used to remotely connect to a server machine from a client machine. The client and the server can be in the same or different systems. The client can access the file system and execute programs on the server

machine [12].

SSH uses cryptographic algorithms to secure the connection. The cryptographic algorithms used are key-exchange algorithms, authentication algorithms, and MAC algorithms.

The SSH protocol is made up of three sub-protocols: SSH Transport Layer protocol, SSH Authentication protocol, SSH Connection protocol. The initial packet exchange starts with Transport Layer protocol, followed by the Authentication protocol, and Connection protocol.

SSH Transport Layer protocol [13] helps to exchange handshaking packets, exchange client capability and server capability (support of encryption or decryption algorithms, MAC algorithms, and compression algorithms) and runs a key-exchange algorithm. The content of handshaking packets is used to generate symmetric keys. Mostly, the Diffie-Hellman key exchange algorithm is used. All packets in this protocol are in plaintext. Server host authentication also occurs in this protocol.

In SSH Authentication protocol [14], all packets are encrypted using a strong cryptographic algorithm and decrypted using the symmetric key generated in the previous protocol. The purpose of this protocol is to provide client authentication. Usually, the client sends a pair of username and a password to the server. The server replies with a success or a failure message. Depending on the configuration, the client can make several login attempts before a successful connection.

SSH Connection protocol [15] executes only after the completion of Authentication protocol (i.e., after successful authentication). As in the previous protocol, all packets in this protocol are encrypted. In the case of a shell application, the client sends commands and the server replies with the command outputs.



## 1.4.2 Static and Dynamic IP Addresses

An Internet Service Provider (ISP) assigns IP addresses to their customers (residents, companies). The IP address could be a static IP address or a dynamic IP address. Static IP address remains the same for an extended period (until ISP/procurer terminates the contract). All publically accessible SSH servers have a static IP address. Dynamic IP address remains the same only for a short duration (called a DHCP lease time). Mostly, when a residential customer purchases an internet connection from an ISP, he/she is usually given a dynamic IP address. It is done by the ISP to increase the number of customers beyond the ISP's capacity of available IP addresses.

When an IP address is statically assigned to a client, in all the password guessing made by an attacker, the source IP address captured in the honeypot remains the same.

When a client computer is assigned a dynamic IP address, a finite amount of time (DHCP lease time) is allocated for using that IP address by that machine. After the lease time, the computer requests the ISP to provide the same IP address for another lease period. Most of the time, the same IP address is assigned to the computer. If the computer does not make a request (on various scenarios like computer shutdown, network failure, and computer network software problems) after the lease time, the IP address is expired and can be used by the ISP to assign to another customer's computer. When the former computer reconnects to the ISP, the computer is assigned a new IP address. Hence, in the password guessing attacks made by the same client (attacker), there is one or more source IP addresses captured by the honeypot.

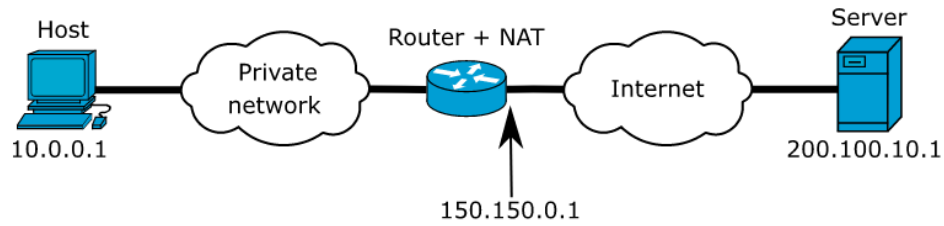


Figure 1.1: Simple Network (Source: wikipedia)

### 1.4.3 Private IP Addresses

The IP addresses discussed in the previous section are universally called a public IP address. Private IP addresses [16] are a block of IP addresses used only for private purposes. Private IP addresses are used within a local network and are not recognized globally.

### 1.4.4 Network Address Translation

Network Address Translation (NAT) [17] is a technology that provides the mapping between private IP addresses and public IP addresses.

In Figure 1.1, the host IP address 10.0.0.1 is a private IP address, the router IP address 150.150.0.1 is a public IP address, and the server IP address 200.100.10.1 is a public IP address.

Assume packets are exchanged between the host and the server. The outgoing packets originating from the host have the source IP address as 10.0.0.1. The router changes the source IP address to 150.150.0.1. The outgoing packets from the server have the destination IP address as 150.150.0.1. The router changes the destination IP address from 150.150.0.1 to 10.0.0.1.

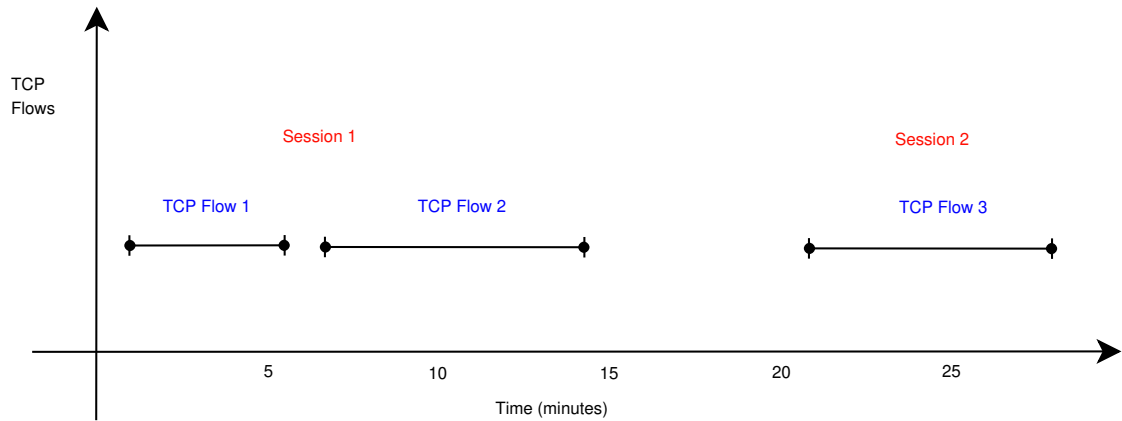


Figure 1.2: TCP flows forming a session

### 1.4.5 Session

A TCP flow is a group of TCP packets between two applications (referred using source port number and destination port number) executing on two different hosts (referred by source IP address and destination IP address).

A session is a set of TCP flows originating from the same client IP address occurring in a finite amount of time. A session is needed because the IP address (public IP provided by the ISP) of a host may not always be the same. It might change if the IP is a dynamic IP address. Hence, the set of TCP flows associated with a particular source IP address occurring in a short duration of time is grouped to form a session. In Figure 1.2, the TCP flow 1 and TCP flow 2 belong to a single session, and TCP flow 3 belongs to another session (assuming the gap between the sessions as 300 seconds or 5 minutes).

The challenging part in forming a session is in estimating the time gap (hereafter called the session gap) between two consecutive TCP flows with the same client IP address and belonging to two different machines. In the literature, Alata [18] has proposed a method to determine the session gap. Also, Nicomette et al.

[19] used Alata’s method on their data set and got a session gap of 16 seconds.

## 1.5 SSH Password Guessing Attacks and SSH Compromises

### 1.5.1 Security Tools to Detect SSH Password Guessing Attacks

There are several security tools to detect and prevent a password guessing attack. The algorithm used in these tools is simplistic and widely known. The algorithm checks for the presence of  $x$  failure login attempts in  $y$  duration of time. If the condition returns true, then the client IP address is blocked for  $z$  duration of time.

Host-based Intrusion Detection Systems (HIDS) are security tools that use log files to detect and prevent attacks. There are several HIDS that implement the above algorithm: fail2ban [20], DenyHosts [21], SSHblock [22], SSHGuard [23], sshdfilter [24], bruteforceblocker [25], and blockhosts [26]. Each tool does the implementation with minor variations. For example, in fail2ban, the values of  $x, y, z$  are 3, 600 seconds, 600 seconds, respectively. In DenyHosts, three different sets of  $\{x, y, z\}$  are provided: one for the “root” account, another one for an existing account (the account details are stored in the system configuration files), and the last one for a non-existent account. In the SSHBlock tool, the value of  $y$  is increased exponentially every time a client IP address is banned by the server. In all the above security tools, the values of  $x, y, z$  are configurable.

Network-based Intrusion Detection Systems (NIDS) analyze only the network traffic to detect and prevent attacks. A NIDS also implements the above algo-

rithm. But, in the case of an SSH protocol, the packets are encrypted (packets of SSH Authentication protocol and SSH Connection protocol), and a NIDS cannot differentiate between a login packet and a non-login packet. So, NIDS (Snort, Suricata, Zeek) cannot decrypt SSH packets to detect password guessing attacks. It can detect the presence of  $x$  TCP connections (made by an SSH client) in  $y$  duration of time (It is assumed that in each TCP connection at least one login attempt is made).

SIEM (Security Information and Event Management) tools are advanced security tools that collect log information from many servers and make a unified decision. SIEM tools are also installed to monitor the internal network of an organization. Password guessing attacks originating from an internal network can be detected using these tools. SIEM is also programmed with the above algorithm to detect password guessing attacks.

### **1.5.2 Other Detection and Prevention Measures**

The following configurations can be done on an SSH server (for example, OpenSSH server) to prevent password guessing attacks.

1. Restrict SSH access by IP address. Only configured IP addresses are allowed to connect to the server.
2. Provide SSH access to certain users only.
3. Deploy SSH server on a different network port number (say 2222).
4. Lock accounts after a pre-defined number of login failures.
5. Disable password-based authentication and enable key-based authentication.

6. Disable “root” access.

### 1.5.3 Non-stealthy SSH Password Guessing Attacks

As per the literature review, all password guessing attacks that can be detected by the security tools is called non-stealthy password guessing attacks. IDS (HIDS and NIDS) and SIEM can detect and prevent non-stealthy password guessing attacks.

Consider a scenario where the client IP address  $x_1.x_2.x_3.x_4$  makes 21 login attempts on an SSH server in a duration of 23 seconds. On averaging, it is one login attempt per second. The security tool (fail2ban with  $x=3$ ,  $y=600$  seconds,  $z=600$  seconds) logs the first three login attempts and immediately detects the presence of a password guessing attack. All login attempts starting from the fourth one will be blocked by the security tool.

### 1.5.4 Stealthy SSH Password Guessing Attacks

The attacks that cannot be detected (using the algorithm in Section 1.5.1) by the security tools are called as stealthy SSH password guessing attacks. Assume an attacker makes  $x_a$  login attempts in  $y$  duration of time. Let the value  $x_a$  be lesser than  $x$ . In this case, the rate of the login attempts is lower than the rate threshold ( $x$  login attempts in  $y$  duration) set by the security tool. This attack cannot be detected by security tools.

There are two types of stealthy SSH password guessing attacks. In the first category, the hacker manually installs a malicious tool in his computer and executes the attack. In the second category, a set of computers which were automatically installed with malware attacks the server.

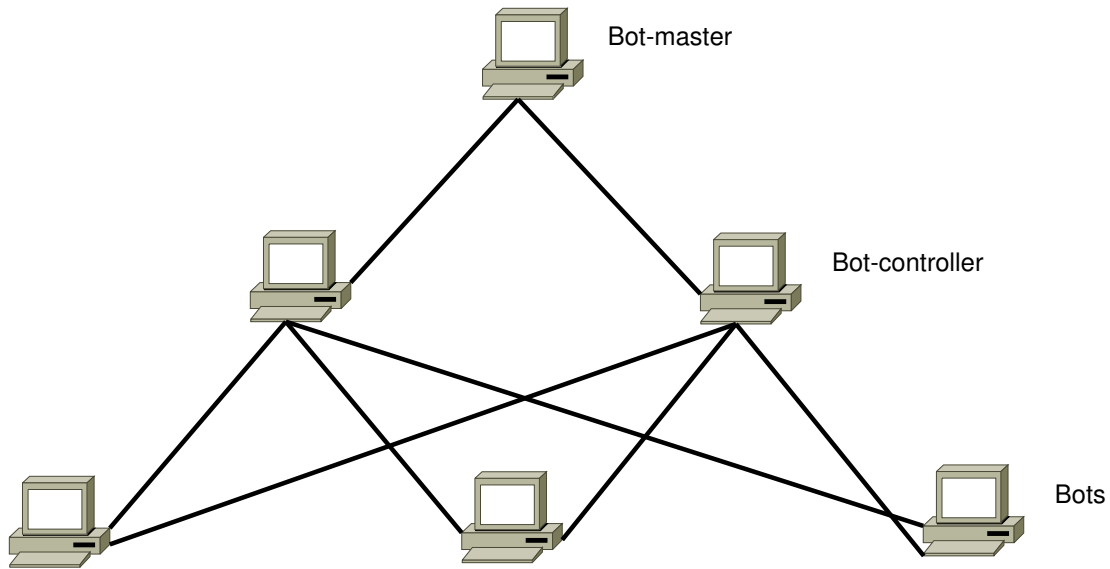


Figure 1.3: Botnet Architecture

### Stealthy Single-Source SSH Password Guessing Attacks

The hacker installs a password cracking tool on a computer and executes the attack. Since the attack is from a single system, it is referred as “Stealthy Single-Source Password Guessing Attack”. There are several attack tools that help in launching this password guessing attack. Some of them are Medusa [27], Patator [28], THC Hydra [29], Ncrack [30], and phrasendrescher [31].

### Stealthy Distributed SSH Password Guessing Attacks

A botnet [32] (as shown in Figure 1.3) is a network of bots (victim computers) which are controlled by a bot-master. Sometimes there might be an intermediary (“bot-controller”) between the bot-master and the bots. The attacks that are launched from a botnet are called “Stealthy Distributed SSH Password Guessing Attacks”.

The stealthy distributed SSH password guessing can occur in the infection

phase or in the attack phase of a botnet. These two phases are explained below.

The infection phase [33] represents the propagation of bot program on the Internet. Once a victim machine is infected with malware, it starts scanning for vulnerable machines. Once a vulnerable machine is found, a password guessing attack is executed. After cracking the correct password, a fresh malware is injected into the new victim. The new victim will also propagate the malware.

In the infection phase, the password guessing of the bots is hardly controlled by the bot-master. As an example, in the Mirai botnet [33] the login credentials are hard-coded in the bot program. Once a bot finds a victim machine through scanning, it starts guessing the login information. Since the password guessing is done independently by each bot, the password guessing is “uncoordinated”.

There are several instances of distributed (both stealthy and non-stealthy) uncoordinated SSH password guessing attacks.

1. Michael Rash reported dd\_ssh [34] botnet that does password guessing on SSH servers. The injection starts by exploiting “phpMyAdmin” vulnerability on a web server. Once a web server is compromised, the botnet starts scanning for SSH servers and ultimately carries out a password guessing attack.
2. Peter Kalnai and Michal Malik reported Linux/Rakos [35] bot malware. The bot runs as a process executed from a temporary directory and disguises its name as “.javaxxx” or “.swap” or “.kworker”. The target of the password guessing is embedded devices and servers running SSH service. Most of the time these embedded devices are IoT devices configured with a default weak password. The list of IP addresses (of SSH servers) is obtained from a web server located at “https://bot-master server IP/scan”. The usernames and



the passwords are hardcoded in the bot program.

3. Goldberg and Ziv reported bread and butter attacks [36]. The password guessing is executed in a stealthy manner. Strangely the number of attack machines was only a limited quantity (four IP addresses).
4. ESET Research and Michal Malik discovered Linux Shishiga malware [37] that does password guessing on various protocols: SSH, Telnet, HTTP, BitTorrent. The password list is hardcoded in the bot program. The credential list consists of weak passwords mostly belonging to IoT devices.
5. Mirai botnet variants [38] also do password guessing on SSH protocol.
6. John E. Dunn reported Chalubo botnet [39] which does password guessing on SSH servers.

In the attack phase, all the bots in unison do password guessing on a single SSH server. All bots coordinate the attack at a particular SSH server. The botmaster (C & C server) issues the victim IP address and the set of login credentials to each bot. The bots together attack the victim server. These attacks are called “distributed and coordinated password guessing attacks”.

The appearance of distributed and coordinated password guessing attacks was first reported by Peter Hansteen [40]. One particular instance of this attack is as follows:

1. At the time  $t_1$ , several bots try the same username “alias” with different passwords.
2. At the time  $t_2$ , several bots try the same username “amanda” with different passwords.

3. At the time  $t_n$  ( $n > 2$ ), several bots try the same username with different passwords.

The attack mentioned by GitHub [3] is a distributed and coordinated SSH password guessing attack executed in a stealthy manner.

The stealthy distributed SSH password guessing is more severe than stealthy single-source SSH password guessing attacks.

### 1.5.5 SSH Compromise

When authentication is successful in an SSH password guessing attack, the client is provided access to the server system. After the system compromise, the malicious entity can steal confidential information, use the server to send spam email, use it to launch phishing websites, execute DDoS attacks, spread malware, and so on.

## 1.6 Research Problem Statement

After an extensive literature survey, the following research problems satisfy the considered research objectives.

1. While carrying out network forensics, it has always been hard to identify the attacker (or a set of attackers) involved in a particular attack. For example, consider two TCP flows or sessions from a malicious origin.
  - (a) Assume that the origin belongs to different IP addresses. It is hard to say if both the IP address belongs to the same botnet.
  - (b) Assume that the origin belongs to the same IP address. If the two sessions appear at different times, then it is hard to say if they belong to the same botnet.

2. A stealthy SSH password guessing attack can be generated either from a single-source or a distributed network. Both these types make login attempts at a low rate. Besides, both the single-source and the distributed attacker use the same standard SSH protocol. It is hard to differentiate them by analyzing their server logs and inspecting their packet details.
3. In an SSH session (network traffic), the packets in the authentication protocol and the connection protocol are encrypted. A failed password guessing attack will have authentication protocol packets and does not contain connection protocol. A successful attack (SSH compromise) would have a connection protocol (in addition to authentication protocol). By inspecting the network packets, it is difficult to determine the presence of connection protocol in a session.

## 1.7 Methodology

This thesis utilizes machine learning approaches particularly supervised learning and unsupervised learning techniques for classification and clustering of attacks, respectively.

Machine Learning is the field of study that involves algorithms that are capable of learning from experience. As per Tom M. Mitchell [41], “A computer program is said to **learn** from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.”

Machine Learning algorithms are used in a wide range of areas: network security, email filtering, social media, IoT security, and so on.

## 1.7.1 Supervised Learning

Supervised learning [42] is a branch of machine learning that build a mathematical model of the training data (a set of input features and output values), in order to make predictions or classification decisions. The training data is labelled (the output value is known). When a testing data arrives, the model (function) predicts the output value.

Given a set of  $N$  data points in a data set of the form  $(X_1, Y_1), (X_2, Y_2), \dots, (X_N, Y_N)$  such that  $X_i$  is the feature vector of the  $i^{th}$  sample and  $Y_i$  its corresponding label (i.e., class label), the supervised learning seeks a function  $g : X \rightarrow Y$ , where  $X$  is the input space and  $Y$  is the output space.

Let  $M$  be the number of labels in a given data set. Let  $M$  be much lesser than  $N$  ( $M \ll N$ ). The supervised learning technique will assign a test sample to a given set of output classes  $(Y_1, Y_2, \dots, Y_M)$ . This type of learning is called “classification”. For a 2-class training data,  $M = 2$  and the output classes are  $Y_1, Y_2$ .

There are several widely used classification algorithms: ZeroR, OneR, Decision Trees, PART, Naive Bayes, Logistic Regression, Support Vector Machines, k-Nearest Neighbour. These algorithms are used in this work and are briefly described in the below paragraphs.

### Algorithms

**ZeroR** ZeroR [43] is a machine learning algorithm that do not use the features to do the prediction. The algorithm counts the number of class labels and chooses the label with the highest count as the “target label”. If there appears a new test data, the class label “target label” is immediately assigned.

ZeroR is mostly used as a baseline classifier. The baseline classifier is needed to compare the performance of different machine learning algorithms.

**OneR** OneR [44] is the most simplistic rule-based classifier that creates a single rule to classify all the data. The rule is based only on a single attribute (best attribute based on a performance measure) from the set of attributes.

**Decision Tree (J48)** A decision tree [45] classifies training data by dividing them down a tree (data structure) from the root node to a leaf node. Each non-leaf node specifies a test of an attribute, and each child of this node corresponds to one of the different values for this attribute. In other words, a decision tree creates a set of “if-then” rules based on the attributes.

J48 is the decision tree algorithm C4.5 (revision 8) written in Java language.

**PART** PART [46] is a rule-based classifier based on C4.5 and RIPPER algorithms. It uses the separate-and-conquer strategy to build the rules. The rules are partial decision trees formed from the C4.5 algorithm.

**Naive Bayes Classifier** The Naive Bayes classifier [47] uses Bayes Theorem to do the classification of items.

For a given data set, let  $X_i$  be the  $i^{th}$  sample (data point) without the class label and  $Y_i$  be the binary class label. Let  $X_{ij}$  be the  $j^{th}$  attribute value of sample  $X_i$ . Assuming that the attributes are independent of each other, the posterior probability  $P(Y_i|X_i)$  is given by the Equation 1.1.

$$P(Y_i|X_i) = \frac{P(Y_i) \prod_{j=1}^d P(X_{ij}|Y_i)}{P(X_i)} \quad (1.1)$$

In Equation 1.1,  $P(Y_i)$  is the prior probability of the classes,  $P(X_i)$  is the probability of occurrence of  $X_i$ ,  $d$  is the number of attributes, and  $P(X_{ij}|Y_i)$  is the probability of  $X_{ij}$  given  $Y_i$ .

The Equation 1.1 is applicable only for categorical attributes. For continuous attributes, the probability distribution is determined. For example, Gaussian distribution is characterized by two parameters, its mean,  $\mu$ , and variance,  $\sigma^2$ .

**Logistic Regression** Logistic Regression uses a sigmoid function to do the classification of items. An hyperplane  $y = \theta^T x$  is found that segregates the “positive” and “negative” classes.

$$h_{\theta}(X_i) = \frac{1}{1 + e^{-(\theta^T X_i)}} \quad (1.2)$$

For a given test sample  $X_i$ , the hypothesis function (sigmoid function) is given by Equation 1.2. Here,  $\theta$  is the vector representing the parameters of the hyperplane.

$$L = - \sum_{i=1}^m (Y_i \log h_{\theta}(X_i) + (1 - Y_i) \log(1 - h_{\theta}(X_i))) \quad (1.3)$$

The cost function (Equation 1.3)  $L$  is used to determine the parameters of the hyperplane.

**Support Vector Machines** Support Vector Machine (SVM) [48] is a classifier that finds the hyperplane that has the maximum margin (gap) between two different classes. If the hyperplane misclassifies a few data points, then to avoid overfitting, a soft margin can be introduced with the help of a slack variable. If a suitable hyperplane does not exist, then the data set can be transformed into a different dimension using several transformative functions.

**k-Nearest Neighbour Algorithm** k-Nearest Neighbour (kNN) algorithm [49] is quite different from other algorithms like Naive Bayes algorithm, SVM, and Logistic Regression. This algorithm does not train a dataset to create a model. When a test sample appears, a decision is taken based on the class label of the nearest neighbors. The nearest neighbors are found using a proximity measure like Euclidean distance.

### **Evaluation of the Classifier Model**

**Validation** Cross-Validation is a resampling technique to evaluate the performance of classifier model on a limited data set. The technique is described using the following steps.

1. Randomly shuffle the data set.
2. Divide the data set into  $k$  groups.
3. For each group (say  $D_1$ ):
  - (a) Take this group  $D_1$  as test data set.
  - (b) Take the remaining  $k - 1$  groups as training data set.
  - (c) Apply the classifier model on the training data set.
  - (d) Evaluate the model using the test data set.
  - (e) Store the evaluation score.
  - (f) Discard the model.
4. Summarize the evaluation scores.

Stratification is a procedure in which every fold of the cross-validation is taken with equal proportion of different class labels.

## Feature Selection in Classification

Feature Selection [50] is a technique to determine the best set of features for the classification of samples. In other words, it helps to remove the redundant and irrelevant features from the set of original features. Feature selection helps in reducing the computational cost involved in training the data set. It helps to increase the accuracy as well.

Exhaustive search is a wrapper method in which all possible combinations of the features are tried out to get the best set. The computation time of this method increases exponentially ( $O(2^F)$ ) with respect to the number of features ( $F$ ).

## Evaluation Metrics for Classification

There are several metrics to check the performance of a classifier. The metrics that are used in this work are given below.

All the metrics used in this work make use of a confusion matrix.

A confusion matrix describes the number of items that are correctly or incorrectly classified by the algorithm. Assume a data set consisting of two classes. The first class is “positive” and the second class is “negative”.

- True Positive (TP): It is the number of “positive” class items that are correctly classified by the model.
- True Negative (TN): It is the number of “negative” class items that are correctly classified by the model.
- False Positive (FP): It is the number of “negative” class items that are classified as “positive” by the model.



Table 1.1: Confusion Matrix

TP	FN
FP	TN

- False Negative (FN): It is the number of “positive” class items that are classified as “negative” by the model.

A confusion matrix is a 2 x 2 matrix consisting of the above four quantities (as shown in Table 1.1).

**Accuracy** Accuracy is the ratio between the number of correctly classified items (both “positive” and “negative” classes) to the overall items (as given in Equation 1.4).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (1.4)$$

**Sensitivity** Sensitivity is the fraction of “positive” items that are correctly classified by the algorithm. It is given by the Equation 1.5.

$$Sensitivity = \frac{TP}{TP + FN} \quad (1.5)$$

**Precision** Precision measures the ability of the classifier to predicts correctly. It is the ratio of the True Positive to that of all items that are classified as “True Positive” by the classifier. It is given by the Equation 1.6.

$$Precision = \frac{TP}{TP + FP} \quad (1.6)$$

**F-score** F-score is the weighted average of Sensitivity and Precision. It ranges between 0 and 1. The value of  $\beta$  determine the priority between sensitivity and

precision. If  $\beta > 1$ , then F-score gives more preference to sensitivity than precision. The mathematical expression for F-score is given in Equation 1.7.

$$F - score = \frac{(1 + \beta^2)(Precision \times Sensitivity)}{(\beta^2 Precision) + Sensitivity} \quad (1.7)$$

## 1.7.2 Unsupervised Learning

Unsupervised learning is the branch of machine learning algorithms that works on a data set that does not have labels or classes. “Cluster Analysis” is the part of unsupervised learning technique that takes a set of data and form groups (clusters) from it. The techniques are also called as “clustering algorithms”.

There are several clustering algorithms: k-means clustering, hierarchical clustering, DBSCAN. DBSCAN is used in this work and explained in the following section.

### DBSCAN

DBSCAN uses density of the data points to form clusters [51]. This algorithm initially assigns labels (core points, border points, noise points) to each data point in a sample. Thereafter, the core points are merged to form a cluster. Each border point gets assigned to a cluster. The noise points are not part of any clusters and based on the application it is also called as “outliers”.

The core function of a DBSCAN algorithm is to determine the density of each data point in a data set. The density is obtained by counting the number of neighbours ( $nPts$ ) within a particular radius ( $Eps$ ). If  $nPts$  is beyond a threshold ( $MinPts$ ), then the data point is called as a “core” point. If the  $nPts$  of a data point is less than the threshold ( $MinPts$ ), but one of the neighbours is a “core” point, then the data point is termed a “border” point. If the  $nPts$  of a data point

is less than the threshold ( $MinPts$ ) and there are no neighbours that are core points, then the data point is called as a “noise” point.

Finally, a core point and its neighbours which are core points are connected to form a cluster. Each border point is assigned to the nearest core point’s cluster.

### **Selection of DBSCAN parameters**

For each data point, the distance to the  $k^{th}$  nearest neighbour is calculated. This distance is called “k-dist”. The value of  $k$  depends on the application domain. In this approach,  $k$  functions as the  $MinPts$  in the DBSCAN algorithm.

All the distances of the data points are sorted in increasing order and plotted in a line graph. If the density of the clusters do not change much, then there will be a sharp change in the value of “k-dist”. The value corresponding to this sharp change is taken as  $Eps$ .

### **Evaluation Metric for Clustering: Silhouette Coefficient**

The Silhouette coefficient [52] is an unsupervised technique to evaluate the performance of a clustering algorithm. This method combines both cohesion and separation.

- Cohesion: determines how close the objects in a cluster are.
- Separation: determines how separated a cluster is from other clusters.

The Silhouette coefficient for a data point is determined as follows:

1. Calculate  $a_i$ : The average distance of the data points  $i$  to all the other points in the same cluster.

2. Calculate  $b_i$ : The average distance of the data point  $i$  to all the points in a different and nearest cluster. This is usually done by finding the average distance between  $i$  and all other points in each cluster. Finally, the minimum average distance is considered as  $b_i$ .
3. Silhouette coefficient  $s_i$

$$s_i = \frac{(b_i - a_i)}{\max(a_i, b_i)} \quad (1.8)$$

The average of all the  $s_i$  is the final silhouette coefficient ( $s$ ) of the clustering algorithm on the data set.

$$s = \frac{\sum_{i=1}^n s_i}{n} \quad (1.9)$$

The value of  $s_i$  or  $s$  ranges between -1 and +1. A negative value indicates that the average distance between the clusters is considerably low compared to the average distance of the points within a cluster. A positive value is desirable and indicates a compact cluster well separated from other clusters.

The silhouette coefficient of a cluster (i.e., an average of silhouette coefficients ( $s_i$ ) of all the data points in a cluster) reaches 1 (the maximal value) only when  $a_i$  reaches 0 for all the data points in the cluster. This indicates all the data points in the cluster are very close (or duplicates).

## 1.8 Scope and Organization of the Thesis

The subject area covered in this thesis work is “Computer Networks” and “Machine Learning”. In computer networks, only the traditional wired networking (not

wireless networking, not personal area networks, and not optical networking) involving TCP/IP stack is considered. In machine learning, only supervised learning and unsupervised learning are explored.

All the proposed methods in this thesis made use of network traffic (pcap files and log files). The traffic is obtained from a honeynet system. Honeynet system captures only malicious packets (no legitimate packets). The duration of capture was nearly nine months and the quantity of network traffic in pcap format is 5.5 GigaBytes. The network protocols in the traffic were limited by the services ran in the honeynet system.

The scope of the thesis covers only the malicious activities on the SSH server. The SSH server is affected by the following methods.

1. Scanning of the server.
2. Password guessing attacks.
3. Malicious activities (DoS attacks, Spread phishing email, Spread malware) carried after the server compromise.
4. Exploiting the server software vulnerabilities.

The first three methods (scanning, password guessing, system compromise) mostly involve network traffic and are analysed in this work. The last method (exploiting vulnerabilities) is in the field of “software security” and not part of this work.

There are methods ([Section 2.3](#)) to detect SSH password guessing attacks. Also, several proposals are made to detect stealthy SSH password guessing attacks. In the literature, there is no method to detect stealthy single-source SSH password guessing attacks.

The stealthy single-source SSH password guessing attacks and stealthy distributed SSH password guessing attacks look very similar and hard to segregate them even manually [53]. There is no automated tool to classify these two types of attacks. Given a honeynet traffic it is hard to distinguish between these two attacks.

Detecting the presence of stealthy single-source SSH password guessing attacks would directly improve the method to detect stealthy SSH password guessing attacks. Since there are several methods to detect non-stealthy SSH password guessing attacks, research on stealthy SSH password guessing attacks is essential.

Consider a honeypot log consisting of stealthy SSH password guessing attacks. All the attacks would be from different sources. It is hard to group them based on the attacker. For example, an attacker (say *Attacker 1*) with a botnet (containing bots of IP addresses:  $IP_1, IP_2, IP_3, \dots, IP_n$ ) would have attacked the SSH server. Another attacker (say *Attacker 2*) with a botnet (containing bots of IP addresses:  $IP_{n+1}, IP_{n+2}, IP_{n+3}, \dots, IP_{n+m}$ ) would have attacked the SSH server. Another attacker (say *Attacker 3*) would have used only one IP address (single-source attacker) to attack the SSH server. All the above attacks are grouped together in the honeypot log files. In the literature, no method is proposed to segregate individual attack instances (separate *Attacker 1*, *Attacker 2*, and *Attacker 3*).

Segregating individual attackers (single-source or distributed) would help in network forensic analysis. From the perspective of law and justice, when an attack happens on a server(s), it is important to know the source(s) behind the attack.

The password guessing (packets with username and password) is done using the SSH Authentication protocol. When an SSH server is compromised, the malicious

activities use SSH Connection protocol. As previously mentioned, all packets in both these protocols are encrypted. Hence, by analysing the network traffic, it is hard to check the presence of SSH connection protocol packets in it.

A method to detect SSH compromises using the network traffic would help improve the detection mechanism of peripheral network security devices (Firewall, NIDS).

Chapter 2 discusses the related works on honeynet architectures, detection of SSH password guessing attacks, and detection of SSH compromises. Some of the literature works discuss the detection of non-stealthy SSH password guessing attacks and some other works discuss the detection of stealthy SSH password guessing attacks. Also, the research papers that classify network attacks are explained.

Chapter 3 explains the honeynet architecture to capture malicious network activities. Each and every component in the architecture is explained in detail. The collected traffic is pre-processed before usage.

In Chapter 4, a methodology is proposed to segregate stealthy SSH password guessing attack instances. In Chapter 5, a clustering technique to segregate stealthy single-source SSH password-guessing attacks from stealthy distributed SSH password guessing attacks are explained. Both chapters discuss the results and limitations.

Finally, in Chapter 6, the detection of SSH compromises is explained in detail. A set of machine learning algorithms were used to check the performance of the classification of the data set. The machine learning model for detecting these SSH compromises with the support for real-time detection is also explained. Also, the results are explained.

# Chapter 2

## Literature Survey

### 2.1 Introduction

Password guessing attacks are captured using honeypots. Hence, a detailed study on honeypots was done. In Section 2.2 the honeynet architectures that capture password guessing attacks are described. There are several research problems in honeypots. These are not considered in this thesis. The purpose of literature survey on honeypots is to have an experimental setup to collect password guessing attacks.

In the literature, there are different approaches to detect non-stealthy SSH password guessing attacks and stealthy SSH password guessing attacks. In Section 2.3, there are two sub-sections. In the first sub-section, the various methods to detect non-stealthy SSH password guessing attacks are described. In the second sub-section, methods to detect stealthy SSH password guessing attacks and their limitations are discussed.

One of the research problem is to detect the stealthy single-source SSH password guessing attacks in honeypot traffic. Honeypot collects only malicious traffic.



Hence, the problem is to segregate the attacks. There are many research works that does segregation of network attacks. These research works are described in Section 2.4.

The papers that mentioned about finding the attack participants are described in Section 2.5. Finally, in Section 2.6 the various works that describe detecting SSH compromises in honeypot traffic are explained.

## 2.2 Honeynet Architecture

Owens and Matthews [54] modified and installed three SSH servers in three different locations: a small business unit, a residential unit, a campus network. The SSH servers behaved as honeypots collecting password guessing attacks. The modifications in the server software were:

1. Addition of code to store the passwords in all login attempts.
2. To always return a failed login attempt (even though the login is successful)
3. Storing of collected data in a remote database.

The architecture is not novel but they did detailed analysis on SSH attacks. They observed a stealthy password guessing attack carried out by a single attacker (same IP address) on an SSH server (honeypot). There were 21 different attack sessions over a duration of eight days. In each attack session, there were less than ten login attempts. The “root” account was targeted with passwords that were progressively increasing in strength (initial passwords were simple like “newuser”, “youareok”; followed by leets like “c4bl3m0d3m” (cablemodem), “c4l3nd4r” (calendar); followed by strong passwords like “U50s8AdF”, “OxZBA4xOMd”).

Adachi [55] proposed “Bit Saucer” that dynamically creates virtual machines running honeypot processes. The purpose of creating dynamic virtual machines is to allow the propagation of malware within a network system. Moreover, virtual machines use fewer resources.

John et al. [56] developed a honeypot design (called as a “heat-seeking honeypot”) that attracts malicious web requests. The honeypot is developed by placing vulnerable website code. More attacks can be collected by indexing the webpages in search engines (Google, Yahoo). Attackers usually find vulnerable web servers by querying the search engines. One such query is “phpiz-abi v0.848b c1 hfp1”, which is run for finding PHP vulnerabilities in a web server. There are a lot of other queries provided in SearchAudit [57].

In the architecture designed and deployed by Romain Bezut and Vivien Bernet-Rollande [58], only SSH service was installed in three virtual machines. The firewall configuration did not allow a compromised server to attack outside Internet users. The focus was to analyse SSH password guessing attacks and SSH intrusions.

The work carried out by Saurabh Chamotra et al. [59] involves deployment of low-interaction honeypot (honeyD tool) in the private subnet of an organisation. HoneyD was configured to emulate FTP, telnet, HTTP, and SMTP services. Tcpdump tool was used to capture the network traffic. They analysed the propagation of worms like Blaster and Conflicker. There is a high possibility for the occurrence of password guessing attacks on FTP/telnet/HTTP/SMTP services.

Nicomette et al. [19] describes the sharing of dictionaries among different attackers. Clustering of different password guessing attacks show that more than 20% of the attacks have high-correlation among the set of login credentials. In other words, for some of the attackers, the set of login credentials (i.e., both

username and the password) were the same.

Sanjeev Kumar et al. [60] propose a virtual network comprising of low-interaction honeypots and high-interaction honeypots. Such a network is called a hybrid honeypot architecture. Their focus was to propose a unique method to capture attacks. Their method captured botnet traffic, and they mentioned the different commands in the botnet IRC channel communication. As per their paper, the architecture is configurable for any application server.

Abdou et al. [61] discuss the reattempting of the same username-password combination by an attacker. For example, the username “root” and the password “root” was tried several times by the same attacker on the same SSH server. As per their findings, there are many variations in the time interval between the login attempts; some of the repeated logins were in seconds, and some were in days. In the data set collected by them, one-third of the IP addresses (i.e., 25% of the total login attempts) manifested this behaviour.

Sentanoe et al. [62] proposed a virtual machine-based SSH honeypot system that captures attacks on SSH server. The system consists of three Virtual Machines: Monitored VM, Introspection VM, Database VM. In “Monitored VM” the SSH server listens and reacts to the commands sent by the attackers. The “Introspection VM” is involved in the data analysis of the SSH commands. “Database VM” stores all the information about the attackers. The purpose of the architecture is to hide the presence of a honeypot to the attackers and provide a mechanism to effectively analyse the SSH attacks.

## 2.3 Detection of SSH Password Guessing Attacks

### 2.3.1 Non-Stealthy Attacks

Kumagai et al. [63] proposed a method to detect password guessing attacks on SSH server. When an SSH client connects to the SSH server, the server makes a DNS query (DNS request message) to the local DNS server. It is done to store the domain name of the client in the server logs. During a non-stealthy password guessing attack, there would be a high rate of DNS queries on the DNS server. By observing the rate of DNS queries, password guessing attacks is detected. Their work always require the presence of DNS server in the organization. Moreover, their scope of work do not include stealthy password guessing attacks. The number of DNS queries created because of stealthy password guessing is low and cannot be detected using their detection algorithm.

Maryam M. Najafabadi et al. [64] proposed a method to detect SSH password guessing attacks using Machine Learning algorithms. Several algorithms were chosen by them: Naive Bayes algorithm, Decision Tree algorithms (C4.5D and C4.5N), and k-Nearest Neighbour algorithm (where k is 5). The performance of all the algorithms were satisfactory. k-nearest neighbour algorithm with k=5 outperformed all the other algorithms. The performance was compared with and without source port features. On both occasions, k-nearest neighbour performed well. Their work does not include the detection of stealthy password attacks.

SSH password guessing attacks was modelled using a Hidden Markov Model (HMM) by Sperotto et al. [65]. The HMM has seven hidden states: active scanning, inactive scanning, active brute-force attack, inactive brute-force attack, active compromise, inactive compromise, end state. The end state is the final state,

after which there is no transition. Flow metrics were used to compute the transition probabilities. Flow metrics considered by them are flows per second, packets per flow, and bytes per packet. Their model was used to build a detection tool called as SSHCure [66].

Most of the routers and network devices are integrated with tools to generate NetFlow data [67]. Every few intervals, these network devices create NetFlow data and transmit to the NetFlow sensor. The authors of SSHCure created a plugin to the NetFlow sensor. The plugin displays attack details, target details, attacker details in a Graphical User Interface (GUI) frontend. The scope of their work does not include stealthy password guessing attacks. Furthermore, the rate detection used in their tool would produce a false negative for stealthy password guessing attacks.

SSHCure 3.0 [68] is an advancement to SSHCure tool that explores the flat nature of SSH password guessing attack traffic. Jonker et al. [69] discuss the effect of TCP retransmission and control packets on SSHCure.

### **2.3.2 Stealthy Attacks**

Malecot et al. [70] proposed a technique to detect stealthy SSH password guessing attacks visually. The technique compared quadtree mappings of several servers. In each quadtree mapping, the source IP address of every malicious entity is plotted and linked. The similarity between any two quadtree mapping visually helps in determining the presence of a distributed password guessing attack. They have not commented on the distributed and uncoordinated SSH password guessing attacks, and the technique will not detect these attacks. Also, they do not mention anything about stealthy single-source password guessing attacks on the SSH server.

Javed and Paxson [53] proposed a method to detect distributed and coordinated SSH password guessing attacks. They used the cumulative sum (CUSUM) algorithm to find the time region where a distributed attack occurs. Within the time region, the individual attackers (instances of distributed brute-forcers) are determined by partitioning a bipartite graph (where the nodes on one side represent the attackers and the nodes on the other side represents the server machines). The single-source high-rate password guessing attacks (i.e., non-stealthy password guessing attacks) were removed during the partitioning step. The partitioning was done manually. The primary limitation of their technique is its inability to differentiate a distributed and coordinated SSH password guessing attack from a single-source SSH password guessing attack (first paragraph of page 6 of [53]). There are a few other minor limitations.

1. The input to the CUSUM algorithm is the login attempts during a time interval. If the number of login attempts made by a distributed attacker is too low (lower than the threshold set by the CUSUM detector), then their detector returns a false negative.
2. If the consecutive input events to their detection algorithm do not form a coordinating password guessing attack, then the algorithm returns a false alarm.
3. The authors have not mentioned about distributed uncoordinated stealthy SSH password guessing attacks.

Honda et al. [71] [72] discovered two different types of stealthy password guessing attacks and proposed data mining approaches to detect them. The attack targets only RDP service. Several client source IPs coordinate to attack a server

machine. In Saito et al. [72], the attack machine keeps executing a password guessing every few intervals. All the source machines involved in the attack had the same number of login features: total number of logins, the average number of logins, the standard deviation of logins. Their method was proposed to detect only this particular type of stealthy password guessing attacks. They do not mention anything about stealthy single-source password guessing attack.

All the above methods do not detect different variants of stealthy password guessing attacks. Moreover, no method describes the detection of stealthy single-source SSH password guessing attacks. The authors took the challenge of detecting stealthy single-source SSH password guessing attacks. The challenge arises because it is hard to segregate stealthy single-source SSH password guessing attacks and stealthy distributed SSH password guessing attacks. Hence, different methods to segregate network attacks were thoroughly investigated. The following section provides methods to segregate network attacks.

## 2.4 Segregation of Network Attacks

Conti and Abdullah [73] used parallel coordinate plots to visualize different network attacks and to differentiate them manually by visualization. The plot is used to detect stealthy port scanning found in honeynet traffic. Application of their technique on stealthy SSH password guessing attacks will not differentiate single-source attacks from distributed attacks. The fingerprints (parallel coordinate plots) of stealthy single-source SSH password guessing attack and stealthy distributed SSH password guessing attack appear similar and renders difficulty in visually detecting them. Hence, stealthy single-source SSH password guessing attacks cannot be detected using their approach.

Pouget and Dacier [74] used association rule learning and proposed a methodology to segregate different attack tools. The attack tools can generate the same type of attacks (i.e., two instances of port scanning attacks) or different types of attacks (one is port scanning, one is password guessing). For example, assuming the output of the approach detected three attack tools: first attack tool does port scanning, the second attack tool does port scanning, third attack tool does password guessing. Their approach created clusters using the network traffic and assigned each cluster to an attack tool. The stealthy single-source attack is generated by individual hackers and it is highly probable for two different attacks to have different characteristics (because the attackers are different). Hence, stealthy single-source SSH password guessing attack does not form a cluster and cannot be detected using their approach.

A similar method of grouping attack patterns was proposed by Thonnard and Dacier [75]. They used the graph-based clustering technique on the honeynet traffic to group similar attack patterns. The only feature used in their method is the time signature of the network traffic. If the two time signatures are similar, then as per their approach the network traffic originated from the same source or same attacker. The time signature generated by stealthy single-source SSH password guessing attack and stealthy distributed SSH password guessing attack is very similar. Hence, their approach cannot be used to detect stealthy single-source SSH password guessing attacks.

Choi et al. [76] also used parallel coordinate plots to differentiate network attacks. Their work was motivated by Conti and Abdullah's work [73]. Several network features (source port number, destination port number, and so on) were used in the parallel coordinate plots. The signatures generated by stealthy



single-source SSH password guessing attacks and stealthy distributed SSH password guessing attacks look similar and their method cannot be used to detect stealthy single-source SSH password guessing attacks.

Sqalli et al. [77] proposed a method to segregate malicious activities found in honeypot traffic. Their approach made use of entropies of network parameters: source IP address, destination IP address, source port number, destination port number, and so on. Their approach proved the segregation of SSH password guessing attacks from other malicious network activities (like port scanning and malicious file downloads). Two different types of stealthy SSH password guessing attacks create minimal change in the entropies and their changed entropies are below the threshold to be detected by their approach.

Ghourabi et al. [78] used spectral clustering algorithm and sequence clustering algorithm to classify web service attacks. Several web service attacks exist: performing SQL injection on web services, Meta-character injection on web services, and so on. The clustering model is built from the features obtained from the packet header and packet payload information. A few of the features are SOAP request type, SOAP response type, SOAP message size, SOAP header size. The above features are specific to a web service and cannot be applied for clustering SSH attacks. Sequence clustering of stealthy single-source SSH password guessing attack and stealthy distributed SSH password guessing attack would produce the same results. Hence, stealthy single-source SSH password guessing attacks cannot be detected using their model.

In the work carried out by Katerina et al. [79], the malicious web traffic was classified into web attacks and vulnerability scans, using machine learning algorithms. The web attacks comprise of DoS attacks, password guessing attacks,

sending spam messages, Remote File Inclusion (RFI) attacks, SQL injection attacks, and Cross-site scripting (XSS) attacks. The vulnerability scans had different scanning techniques on the static and dynamic web pages. The stealthy SSH password guessing attacks cannot be classified using their technique.

## 2.5 Determining the Attack Participants

A few other research works have shown interest in determining the nature of the attackers. Malecot et al. [70] suggest that most of the attackers involved in distributed password guessing are using Linux systems. Javed and Paxson [53] proposed a method to segregate the attackers. The method was part of detecting stealthy SSH password guessing attacks. Different botnets were grouped based on commonality found in the username or server IP. Abdou et al. [61] find that the password guessing from a botnet occurs from a single subnet.

## 2.6 Detection of SSH Compromises

Detecting an SSH compromise is a challenging task because the SSH packets in the authentication protocol and in the connection protocol are encrypted. The SSH Authentication protocol packets are only involved in sending login credentials (for example, username and password) and verifying it through status messages. The SSH Connection protocol packets appear only if the authentication is successful. The packets in the SSH connection protocol are the interaction between the client and the server. Since the packets in both layers are encrypted, detecting the presence of the SSH connection protocol in the network traffic is tedious. SSH connection protocol signifies a compromise. There are two main approaches men-

tioned in the literature that addresses this issue.

Satoh et al. [80] proposed a methodology to detect SSH compromises using Ward Clustering algorithm. The methodology examines the network traffic to obtain feature values. The machine learning features are packet size and packet direction. The packet direction is either incoming packet to the server or an outgoing packet from the server. They detect the presence of SSH connection protocol in the traffic. Their work do not discuss the practical applicability of their methodology, which we have addressed with the help of a real-time detection mechanism. During data analysis, we found some TCP flows in which the attacker closes the SSH session immediately after a successful login (no packets in the SSH connection protocol). The model (by Satoh et al. [80]) does not mention these types of TCP flows.

Hofstede [81] improved the SSHCure tool by adding a mechanism to detect SSH compromises. As per them, SSH compromise is divided into four categories.

1. Maintain connection, continue dictionary
2. Maintain connection, abort dictionary
3. Instant logout, continue dictionary
4. Instant logout, abort dictionary

“Maintain connection” is that the attacker keeps sending packets to the server, even after a successful login attempt. “Instant logout” is when the client terminates the TCP connection immediately after logging in successfully to the SSH server. These four scenarios were observed in the network traffic to detect SSH compromises. As mentioned in the fourth item, they noticed the termination of SSH session after a successful connection.

## 2.7 Tabular Summary of Malicious Network Activities and their Countermeasures

Table 2.1: Summary of Malicious Network Activities and Countermeasures Techniques

Malicious SSH Network Activities	Work	Approach to address them
Non-Stealthy SSH	Kumagai et al. [63]	High rate of DNS queries
Password Guessing Attacks	Maryam M. Najafabadi et al. [64]	Machine Learning algorithms
	SSHCure [66]	Flow rate
Stealthy SSH Password Guessing Attacks	Malecot et al. [70]	Visually find similarity between two quadtree mappings
	Javed and Paxson [53]	CUSUM algorithm
	Honda et al. [71]	Data Mining (RDP service only)
	[72]	
Segregation of Network Attacks	Conti and Abdullah [73]	Manual detection using parallel coordinate plots
	Pouget and Dacier [74]	Association rule learning
	Thonnard and Dacier [75]	Graph-based clustering method

Malicious SSH Network Activities	Work	Approach to address them
	Choi et al. [76]	Parallel coordinate plots
	Sqalli et al. [77]	Entropies of network parameters
	Ghourabi et al. [78]	Spectral (and sequence) clustering algorithm to classify web service attacks
	Katerina et al. [79]	Machine learning algorithm to classify web attacks and vulnerability scans
Identifying Attack Participants	Javed and Paxson [53]	Username, server IP used in a heuristic-model
Detect SSH Compromises	Satoh et al. [80]	Ward Clustering algorithm
	Hofstede [81]	Behaviour of dictionary attack added in SSHCure tool

# Chapter 3

## Honeynet Architecture

### 3.1 Introduction

In this chapter, the honeynet architecture used to garner password guessing attacks is discussed.

A honeypot [6] does four different functions: Data Capture, Data Control, Data Analysis, Data Alerting. “Data Capture” is the ability of the honeypot to capture useful information about the attacker. “Data Control” is the ability to restrict the damage caused by the honeypot on other networks. “Data Analysis” is the ability to conduct forensics analysis to discover the attacker’s methodology. “Data Alerting” is the ability to alert a suspicious activity.

In a Generation I honeypot [6], data capturing is done by an Intrusion Detection System (IDS) and data control is achieved by a firewall. The firewall limits the number of outbound connections, thereby preventing Denial-of-Service (DoS) attacks and scanning attempts. Also, routers could be configured with ACL rules to block spoofed traffic. In a Generation II honeypot [82], the architecture was improved by taking care of the overload on the data control module. Honeywall

(eeyore tool) was involved in data control. Generation II honeypot had more flexibility, manageability, and security of the honeypot compared to Generation I honeypots. The architecture of Generation III honeypot is very similar to Generation II honeypot. The Honeywall is used with three interfaces. Data capture module is improved by using Sebek modules, Snort, and iptables logs.

Generation II honeynets can rarely be detected by hackers, because, the honeypots used are high-interaction systems. The traffic from the honeypots to the outside world is restricted using a Honeywall [83], which acts more like a firewall. There is a risk associated with honeynets. If a hacker comes to know that these systems are honeypots, then he could use it to attack the production servers.

### 3.1.1 Issues in Deploying a Honeypot

The following are some of the issues in deploying a honeypot [84].

#### 1. Liability Challenges

- (a) Honeypots can be used by a victim machine to cause DDoS attacks on other systems. An attacker would send an SYN packet with the source IP address of the target (for example, targeting a big bank). The honeypot would reply with an SYN-ACK packet to the target. The target would discard it or reply with an RST packet. But, the target has utilized some of the bandwidth and processing power to discard or reply with a message.
- (b) Assume that the honeypot system is not configured properly for outgoing network traffic. In a high-interaction honeypot, a malware might directly send several SYN packets to a target host (for example, a big

bank). The bank might sue the organisation or the administrator for carelessly allowing all network traffic to their system.

This chapter is organized as follows: Section 3.2 briefly describes the different honeypot tools used in the honeynet architecture. A detailed description of the architecture is given in Section 3.3. Pre-processing of the captured network traffic and log files is explained in Section 3.4. Finally, the statistics of the password guessing attacks are discussed in Section 3.5.

## 3.2 Honeypot Tools

### 3.2.1 Dionaea

Dionaea [7] is an updated version of Nepenthes [85]. It is a LIHP built using C, Python, and SQLite. It supports the TLS protocol and the IPv6 protocol. The software of the Dionaea is built using several modules. Some of them are: vulnerability module, shell emulation module, a download module, submit handler. Vulnerability module is used to emulate vulnerable parts of the network services. Shell Emulation modules are used by attackers to interact with a shell and execute commands. Download Handler contains functions to download files from remote locations. Submit Handler involves in transferring log files to a remote database for further analysis.

### 3.2.2 Glastopf and Snare

Glastopf [8] or its successor Snare [86] is an LIHP specially designed for capturing web attacks. It listens on port 80. Glastopf simulates web vulnerabilities in its



Python module to capture web attacks (password guessing attacks, remote file inclusion attacks, SQL injection attacks, local file inclusion attacks). On receiving an HTTP request, Glastopf decodes the request parameters and intelligently provides an appropriate dynamic HTTP response message. It has support for multi-stage attacks and uses SQLite DB or MySQL DB for storing logs.

Snare is a successor to Glastopf. Snare works in coalition with Tanner. Snare sends all HTTP requests to Tanner. Tanner tool decides the correct response.

### 3.2.3 Kippo

Kippo [10] is a MIHP that listens on port 22, which is for Secure Shell connection to a remote server. It provides a shell environment to an attacker to perform attacks like password guessing attacks, remote file downloads, and so on. Kippo can also be used to capture and record session interactions.

Initially, the client connects to the SSH server. Then the conversation is carried out to perform algorithm negotiation and establishment of security keys for encryption/decryption and integrity checking. The client sends a username and a password in the SSH authentication protocol. The login credentials (username, password) are verified against the “userdb” file found in the Kippo tool. If the login credentials do not tally, the client is returned a failure message. Once the connection is successful, then the client is allowed to execute any number of UNIX commands. All UNIX commands entered and the output is stored in rotating log files.

The Kippo server can be detected by a clever attacker. The output of all the UNIX commands (like ls, cp, mv) is stored in text files of the Kippo software. For example, typing “w” command in the computer terminal will return the same list

of logged users every time. In addition, Kippo does not support several complex UNIX commands (i.e., only a small set of commands are implemented in the Kippo tool). An inquisitive attacker would detect the honeypot tool manually. On the other hand, during the password guessing stage, the tool cannot be detected.

### **3.2.4 HoneyD**

HoneyD [87] is a LIHP that is capable of simulating various operating systems in a virtual network environment. HoneyD calls each of the systems a personality engine, tuning the response packet in agreement with the OS kernel. For example, TTL (Time To Live) value in TCP header will be different for a Windows, Ubuntu, and Mac OS X. HoneyD gives an appropriate value for TTL to an attacker. In general, it simulates the network stack of various operating systems.

### **3.2.5 J-Honeypot**

J-Honeypot [88] is a LIHP written in Java language. It has a web-based monitoring console and connects to a rule-based intrusion detection engine. J-Honeypot does not capture penetration attacks but helps in knowing attacks at the network level. It is especially useful if the need is only to know the kind of services that are attacked by hackers. Furthermore, it helps to blacklist the attacker's IP addresses dynamically.

The level of interaction is minimal in the J-Honeypot. When a malicious entity sends an SYN packet to the server, the J-Honeypot sends an SYN-ACK or an RST packet based on whether the port is opened or closed. When the malicious entity sends an application layer data, it is logged and the server only acknowledges the packet. Thereafter, the J-Honeypot server sends an RST packet to terminate the

connection.

The honeypot does not provide much interaction with the attacker. It is hard to gather password guessing attacks using this tool.

## 3.3 Architecture to Collect Malicious Traffic

### 3.3.1 Honeynet Architecture

A honeynet system was built from scratch in the server room of Birla Institute of Technology and Science (BITS), Pilani - Hyderabad campus. The installation and configuration were done on a DELL PowerEdge Blade server. The machine configuration of the DELL server is shown in Table 3.1.

Table 3.1: Server machine configuration

System information	
Processor	Intel Xeon processor
Speed	2.67 GHz
RAM	4 GB
Disk space	600 GB
OS	Ubuntu 16.04

The hostname of the server is “BITS-OS-PC”. The server had two Network Interface Cards (NIC): one of them was Idle, the other was configured with a public IP address.

In this architecture (as shown in Figure 3.1), the traffic from the Internet is directed towards the DELL server using the Cyberoam (router and firewall).

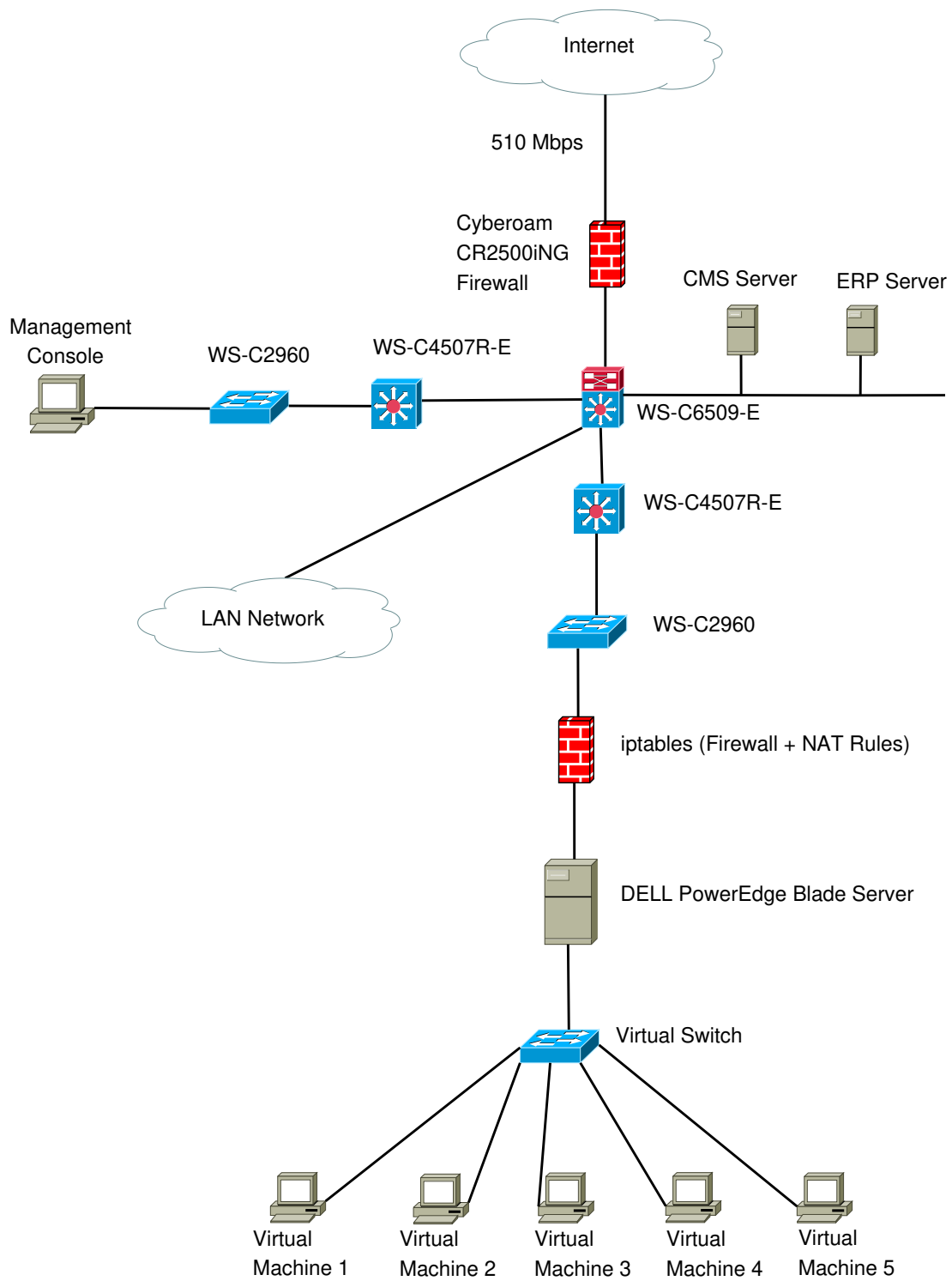


Figure 3.1: Honeynet Architecture

The Cyberoam lies between the Internet and the DELL Server. The traffic from the Internet Service Provider (ISP) reaches the university campus through the Cyberoam firewall. There are a few Cisco core switches between the Cyberoam and the DELL server.

The Cyberoam connects with a separate LAN network. The campus uses the LAN network for legitimate traffic. Any new traffic generated from the honeynet cannot propagate the internal LAN network of the campus. The Cyberoam also connects with the internal servers (CMS server, ERP server, Web server) found in the DMZ zone.

A management console monitors and controls the honeynet. The management console is a computer system in the Networks Research Lab. There are a couple of network switches between the console and the honeynet system. Firewall rules were configured in the Cyberoam, such that, traffic goes only between the honeynet and the management console. The console cannot connect to any systems in the campus LAN network. Moreover, none of the systems in the campus network was able to access the honeynet or the management console.

## **Virtual Network**

Several Virtual Machines (VMs) were created on the physical server. Virtual Machine Manager (VMM) [89] was used to create the virtual machines. libvirt module in the Ubuntu OS supported the creation of virtual machines.

libvirt module in VMM used the concept of a virtual network switch. The virtual network switch operates in three routing modes: NAT mode (default), Routed mode, Isolated mode. By default, virtual network switch operates in NAT mode. In NAT mode (Network Address Translation mode), the virtual hosts are

assigned private IP addresses. All virtual hosts are in a separate network from the physical machine. Each virtual host can communicate with other virtual hosts directly. The virtual switch keeps sending “HELLO” packets to maintain a tree structure of the network.

By default, the virtual switch also does not allow incoming connections to the virtual machines. Hence, NAT rules are required to direct the traffic to the correct virtual machines. iptables is installed on the DELL PowerEdge Blade server to direct the traffic to the correct virtual machine. iptables is a default firewall (software firewall) in many Linux based operating systems with the capability of Network Address Translation. The NAT rules are defined in the iptables. In the NAT mode, the virtual switch does IP masquerading.

Five virtual machines were installed on the VMM. The Operating System running on the virtual machine was mini-Ubuntu 14.04 LTS. Honeypots discussed in the previous section (Section 3.2) were installed on the virtual hosts. The list of honeypots on the virtual machines is shown in the Table 3.2.

None of the virtual machines has access privileges to connect to other virtual machines. Each virtual machine (VM) has a set of firewall (iptables) rules governing the traffic between the VMs. The firewall rule does not allow the traffic from one VM to enter another VM. It is done to segregate the network traffic and to avoid a compromised VM system to affect other ones.

### **Tools to Collect Malicious Traffic**

Every honeypot runs a set of services. Table 3.2 shows the list of services (ports) run in each one of the honeypots. The choice of services depends on the capability of a given honeypot.

Table 3.2: HoneyNet Virtual Machines

Virtual Machine No.	Hostname	IP Address	HoneyPot Tool	Port No. (Service Name)
Virtual Machine 1	hr	192.168.122.101	Snare	80 (HTTP)
Virtual Machine 2	acct	192.168.122.102	Kippo	22 (SSH)
Virtual Machine 3	cms	192.166.122.103	Dionaea	21 (FTP), 1433 (MS-SQL), 3306 (MySQL)
Virtual Machine 4	adm	192.166.122.104	HoneyD	23 (Telnet), 25 (SMTP), 53 (DNS), 110 (POP3)
Virtual Machine 5	-	192.166.122.105	J-HoneyPot	135, 137, 138, 139, 1080, 1243, 12345, 12348, 27374, 31337

Kippo honeypot captures only SSH related attacks (SSH port scanning, SSH password guessing, DoS attacks after an SSH compromise). Cowrie is an advancement over Kippo honeypot. Cowrie was not deployed because the initial data was collected using the Kippo tool. For SSH protocol, HoneyD and Dionaea captured only password guessing attacks. Glastopf/Snare had no capability to capture SSH attacks. J-Honeypot cannot record SSH password guessing attacks. Most of the other tools were not that capable as Kippo honeypot in capturing SSH related attacks.

Glastopf and Snare were designed and developed, considering several hundreds of web vulnerabilities. Hence, they can capture several web attacks. On the other hand, Kippo and Cowrie cannot catch HTTP attacks. Moreover, HoneyD and Dionaea do not have as many features as Glastopf/Snare for web attacks.

HoneyD and Dionaea are sound enough to capture password guessing attacks on some of the common network services (Telnet, SMTP, POP3, MS-SQL, MySQL). The firewall allows traffic to the ports mentioned in Table 3.2 only.

Each one of the honeypots is configured to allow more attacks on it. For example, the Kippo honeypot was configured with a few changes: change of default hostname, setting default usernames and passwords, change of the file system contents.

The malicious traffic was captured using several log capability provided by the honeypot tools. The network traffic was captured using a packet capturing tool (“tcpdump”). tcpdump stored the files in pcap format. The pcap files were processed to remove unwanted network traffic. The unwanted traffics were Ubuntu updates and upgrades, legitimate traffic did for testing purpose, and honeypot status messages.



Table 3.3: HoneyNet Time Periods

Data set No.	Time Period	Duration (days)	Total Traffic (MB)	Total packets
Data set 1	22 July 2014 to 16 August 2014	25	86	669,831
Data set 2	11 April 2017 to 29 April 2017	19	893	35,67,125
Data set 3	5 June 2017 to 10 January 2018	220	4,500 (approx.)	1,36,09,085

### 3.3.2 Network Traffic Collection

The honeyNet system was active only on certain time periods (Table 3.3). The system ran incessantly, day and night, with minor interruptions. The interruptions were caused by shutdowns to retrieve the log files and restarting of the honeypot tools due to software bugs in the tools.

The software and network configuration of all the data sets are the same. J-honeypot tool captured less important traffic in data set 1, and was not used while collecting data set 2 and data set 3. Glastopf was used in data set 1. Snare is an advanced version of Glastopf and was used for collecting malicious traffic in data set 2 and data set 3.

### 3.3.3 Advantages

The following list provides the advantages of our honeyNet architecture.

1. Our honeynet design captures stealthy password guessing attacks.
2. In this design, several honeypots are working together to increase the attack surface.
3. Appropriate blocking rules added to the Cyberoam protects the system from damages.
4. The honeynet is resilient to penetration attacks because the honeypots do not give full access to the operating system. Hence, an outsider cannot perform severe damage to any of the systems.
5. The attackers cannot propagate any malware using the honeynet systems, because the honeypots do not allow the creation of new TCP connections.
6. The honeynet architecture is robust. Even by chance if anyone of the honeypot is brought down by an attacker, the other honeypots are not affected.
7. Virtualization helps in reducing the hardware cost.

### **3.4 Pre-processing of Network Traffic and Logs**

There was unwanted traffic in the capture files. For example, HoneyD honeypot periodically sends its capture log to either `hpfriends.honeycloud.net` (hosted at 81.166.122.240) or `www.honeyd.org` (hosted at 207.158.15.70). All these log packets used HTTP protocol. These packets are not malicious and are used to collect all HoneyD logs in a centralised database. Besides, there were unwanted DNS queries in the pcap files. HoneyD system was configured to connect to a domain

No.	Time	Source	Destination	Protocol	Length	Info
16	22920.369203	192.168.122.102	54.148.18.101	TCP	130	2222 → 50570 [FIN, PSH, ACK]
17	22927.665573	192.168.122.102	54.148.18.101	TCP	130	2222 → 51332 [FIN, PSH, ACK]
18	35000.443392	192.168.122.102	54.148.18.101	TCP	130	2222 → 41620 [FIN, PSH, ACK]
19	35010.043543	192.168.122.102	54.148.18.101	TCP	130	2222 → 43196 [FIN, PSH, ACK]
20	35050.236149	192.168.122.102	54.148.18.101	TCP	130	2222 → 48986 [FIN, PSH, ACK]
21	35062.524345	192.168.122.102	54.148.18.101	TCP	130	2222 → 46484 [FIN, PSH, ACK]

Figure 3.2: FIN/FIN-ACK packets with a private IP address

name. Hence, the system needs to run a DNS query to obtain IP addresses corresponding to those domain names. These DNS queries were also not malicious and purged from the original pcap files.

Snare honeypot contacted Tanner server (212.47.229.233) periodically to download dorklists. These dorklists kept the webserver updated with the latest URL strings. These packets are unnecessary for attack analysis and removed from the pcap files.

Many pcap files had traffic to Ubuntu servers (Ubuntu servers refer to security.ubuntu.com, in.archive.ubuntu.com, ppa.launchpad.net, changelogs.ubuntu.com, us.archive.ubuntu.com). This traffic was from system applications doing periodic security updates and application updates. These packets are not malicious and discarded from the attack analysis.

The capturing tool stored some of the packets with private IP addresses. All these packets are control packets used for connection termination (FIN/FIN-ACK packets) originating from the honeynet server. The occurrence of these is due to the performance of NAT (iptables) not translating the private IP address to a public one. Connection termination packets add very little value to this work's objective. Hence, they are removed.

Figure 3.2 shows a few TCP connection termination packets with source IP address 192.168.122.102 (i.e., a private IP address).

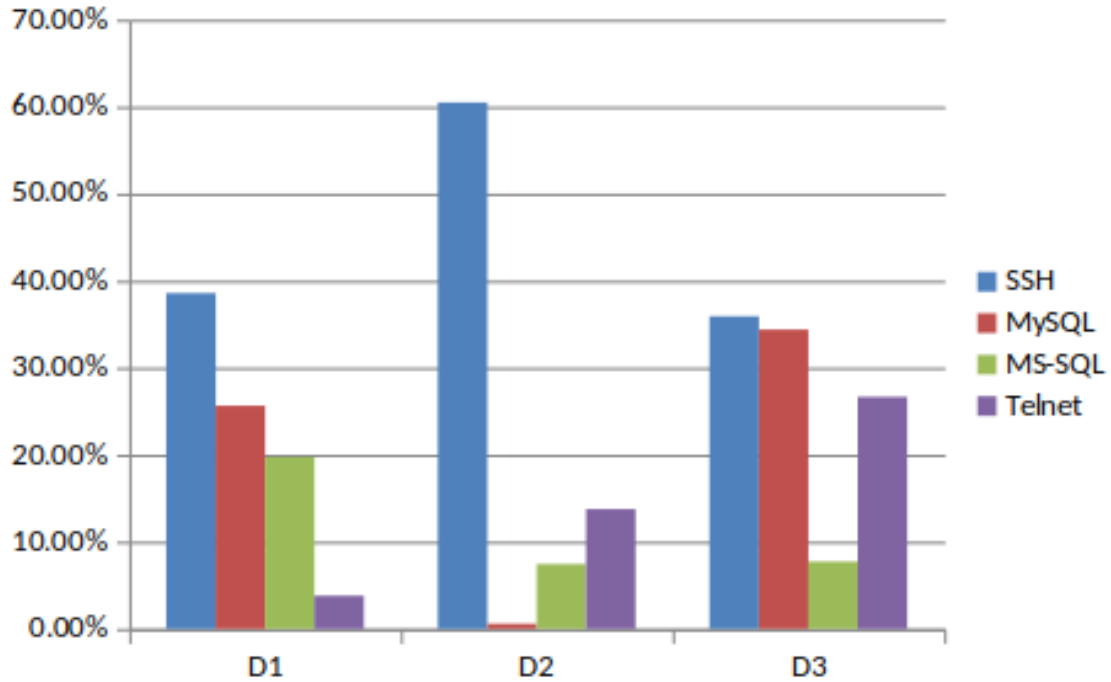


Figure 3.3: Most wanted open ports

### 3.5 Results and Discussion

The open ports which had huge traffic in data set 1, data set 2, and data set 3 is shown in Figure 3.3.

The database services MySQL and MS-SQL had an approximately equal number of flows (6896 and 5309 flows) for data set 1. The percentage of MS-SQL traffic is nearly the same in data set 2 and data set 3. In data set 2, the percentage of traffic to MySQL port was meagre (0.57% (603 flows)).

The intensity of traffic to each service is completely unpredictable. The amount of traffic to each service depends on the time when the attack is carried out. The huge traffic to the services SSH, MySQL, MS-SQL, and telnet could be because of the login capability provided by these services. The login system helps an attacker

to carry out malicious activities on the server. There were many login attempts made to the above services.

The works [59, 60] deployed a honeynet with several services. The highly attacked network service was not discussed by them. On the other hand, most of the network traffic in our data collection was to the SSH service. Hence, a detailed analysis was done on this service.

# Chapter 4

## Identifying each Stealthy SSH Password Guessing Attack Instances

### 4.1 Introduction

The objective of this chapter is to segregate different instances of stealthy SSH password guessing attacks. Each instance refers to a single botnet (or a single machine) that is involved in stealthy password guessing. It is similar to finding a group of attackers (or a single attacker) involved in a particular attack.

The works [70, 53, 61] also explored the group of participants in an attack (Section 2.5). The author took a different task of segregating each attack instances. This is required in many network forensics applications. For example, if a server is crashed because of a DDoS attack, it is legally important to find the machine(s) that were involved in the attack. Hence, knowing the attack participants has a

high value.

The approach is based on finding the similarities in the content of the SSH traffic. Most of the SSH information was readily available in the Kippo log files, and the network traffic (pcap files) is used only to obtain the client operating system. Proper justification is also given for the choice of the individual SSH fields.

This chapter is organized as follows: Section 4.2 explains the different types of TCP flows in the honeynet traffic. Section 4.3 approaches the attack participants in a different viewpoint. Section 4.4 looks into the statistics of certain items in the SSH traffic, that are useful in building the model. Section 4.5 details the model to segregate different attack instances. Section 4.6 provides the results and discusses a few limitations of the model. Finally, Section 4.7 gives the conclusion.

## 4.2 Data Pre-processing

### 4.2.1 Processing of TCP flow pcap files

As mentioned in Section 3.3.1, the packets were captured in “tcpdump” tool. From 11 April 2017 to 10 January 2018 (the duration for data set 2 and data set 3 as in Section 3.3.2) the tcpdump tool executed continuously capturing packets. Hence, for TCP flow pcap files the data set 2 and data set 3 is combined together in a single data set (“D23”). The data set comprising pcap files obtained during data set 1 duration is referred as “D1”.

Several different types of TCP flow pcap files were observed in the collected network traffic.

Note: The second column (“D1 count”) and the fourth column (“D23 count”)

Table 4.1: TCP flow types

TCP flow types	D1 count	D1 (%)	D23 count	D23 (%)
No SYN packets	4	0.01%	70	0.07%
One SYN packet	2287	8.50%	26154	24.83%
Two SYN packets	23228	86.31%	49623	47.12%
n-SYN packets	1392	5.18%	29475	27.98%

are the number of TCP flows.

In Table 4.1 different types of pcap files in the data set is described. “No SYN packets” refer to TCP flow pcap files having no SYN packets in the start of the TCP conversation. A SYN packet is a TCP packet with SYN flag set to 1. There were two types of flows in this category.

1. ACK scanning packets which start with an acknowledgement packet (TCP packet with ACK flag set to 1) and ends with a reset packet (TCP packet with the RST flag set to 1).
2. NAT error packets having only the last few packets of a TCP conversation (without the SYN packets)

“One SYN packet” refers to TCP flows having only one SYN packet in it. It refers to SYN scanning flows, which start with a SYN packet and ends with a RST packet.

“Two SYN packets” refer to normal TCP flow packets having SYN packet followed



by a SYN-ACK packet. “n-SYN packets” implies TCP flows having more than two SYN packets. Due to the presence of multiple TCP flows with same source port number, same destination port number, same source IP address, and same destination IP address, there is an appearance of many SYN packets. Another reason for multiple SYN packets in a flow is because of a non-responding client application. Consider the following scenario.

1. Assume a client sends a SYN packet.
2. The server responds with an SYN-ACK packet.
3. Now, the expected behaviour is for the client (or the malicious host) to reply with ACK packet and proceed with the communication. Instead, the client does not respond. It is an expected behaviour if the source IP address is forged in the SYN packet.
4. When the client does not respond, the server assumes that the SYN-ACK packet is lost. So, the server keeps sending SYN-ACK packet for a few couple of times before giving up. For example, in a typical Linux machine, the time interval between successive SYN-ACK packets increases exponentially. After five retries, the server gives up.

Only the flow types “No SYN packets”, “One SYN packet” and “Two SYN packets” are considered for analysis.

#### **4.2.2 Processing of Kippo log files**

The packets in the SSH Authentication protocol and SSH Connection protocol are encrypted and does not provide useful information to the packet analysers. The

kippo log files provide a considerable amount of information about the content in the SSH Authentication protocol and SSH Connection protocol.

In order to understand the contents of kippo log files, the report [90] written by Kamil Koltys was helpful. Using the log files, Java programs were written to extract SSH logins (usernames and passwords), SSH version, attacker's (client) IP address, SSH encryption/decryption algorithm, SSH compression algorithm, SSH integrity algorithm and so on. Each content is appropriately stored in the correct TCP flow. The output of the program returns a list of TCP flows.

## 4.3 Source of the Attacks

### 4.3.1 Source IP Address

It is useful to know the origin of an attack. It helps in doing forensic analysis. The origin refers to country location, the tool used, the characteristics of the attack tools, and so on. The location can easily be obtained using the source IP address. Sometimes the source IP address may not belong to the attacker: in the case of a bot installed on a victim computer, in the case of a proxy server the attacker uses to hide his exact location.

In data set 1, approximately 3050 unique IP addresses belong to 77 different countries, and in D23 (data set 2 and 3), nearly 26,070 unique IP addresses belong to 153 different countries. Most of the IP addresses belonged to China accounting for 61.25% of the traffic (TCP flows) in data set 1 and 49.49% in D23 (as shown in Figure 4.1). Besides China, 18.97% of the attackers in data set 1 belonged to the USA (11.76% in D23). In general, more than 65% of the attackers were from Asia (all data sets). The website maxmind.com [91] provided the geolocation for

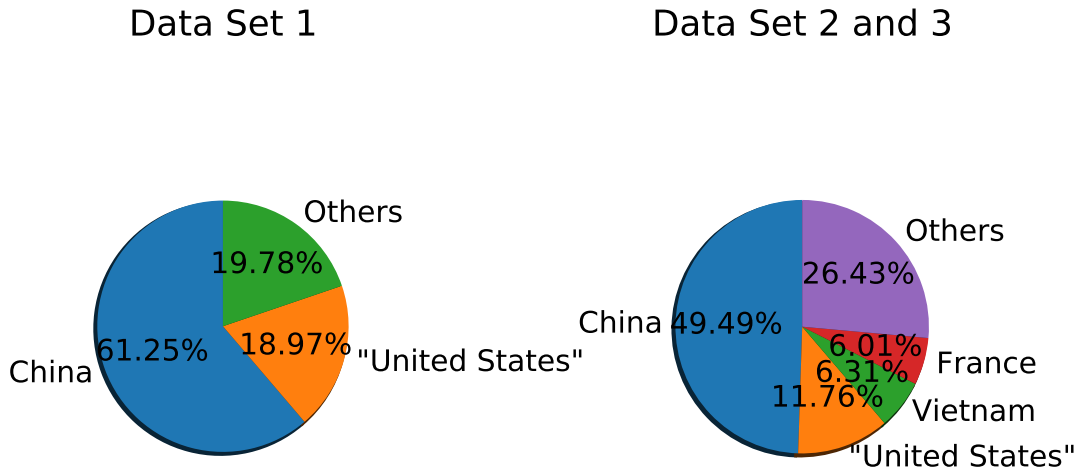


Figure 4.1: Geo-location of IP Addresses

all the IP addresses.

Several attackers (IP addresses) kept attacking the honeynet system (huge number of TCP flows). These attacker's countries are China, Germany, Turkey for data set 1 and China, France for D23.

Abdel Rahman et al. [61] analysed the number of TCP flows with respect to the country's IP block. The block addresses were assumed to be classful addresses (sub-blocks /8, /16, /24). The country's block addresses have classless addressing scheme. Hence, the analysis with classless addressing is done by us.

All the subnets in data set 2 and data set 3 of Table 4.2 belong to China. It is notable that Abdou et al. [61] got China as the first country.

Table 4.2: Busy IP blocks

Data set no.	IP blocks	Total flows	Percentage
2	116.8.0.0/14	1833	6.81
2	61.160.0.0/11	1732	6.44
2	60.160.0.0/11	1282	4.76
3	116.16.0.0/12	14441	13.71
3	58.192.0.0/11	12444	11.82
3	59.32.0.0/11	9675	9.19

### 4.3.2 Source Port Number

The source port number of every TCP flow is randomly assigned by the Operating System of the machine. The range in which the source port is chosen varies from one OS to another. As per IANA standard, a dynamic port number should be in the range, 49,152 through 65,535 [92]. For some Linux-based systems like Ubuntu 16.04, the range is from 32,768 to 60,999. For Windows 10, it is from 16,384 to 49,152.

On observing the source port number of the network traffic, the count of each port number was not uniformly distributed (as shown in Figure 4.2). Hence, most of the attacks are originating from a couple of source port numbers. The source port 6000 occurred most of the time in data set 2 (884 TCP flows, as shown in Figure 4.2) and in data set 3 (167 TCP flows). The source port 6000 mostly attacked port numbers 1433 (345 TCP flows), 3306 (166 TCP flows) and 22 (44 TCP flows to SSH). Some port numbers appear a lot many times because its value is hard-coded in the malicious client program.

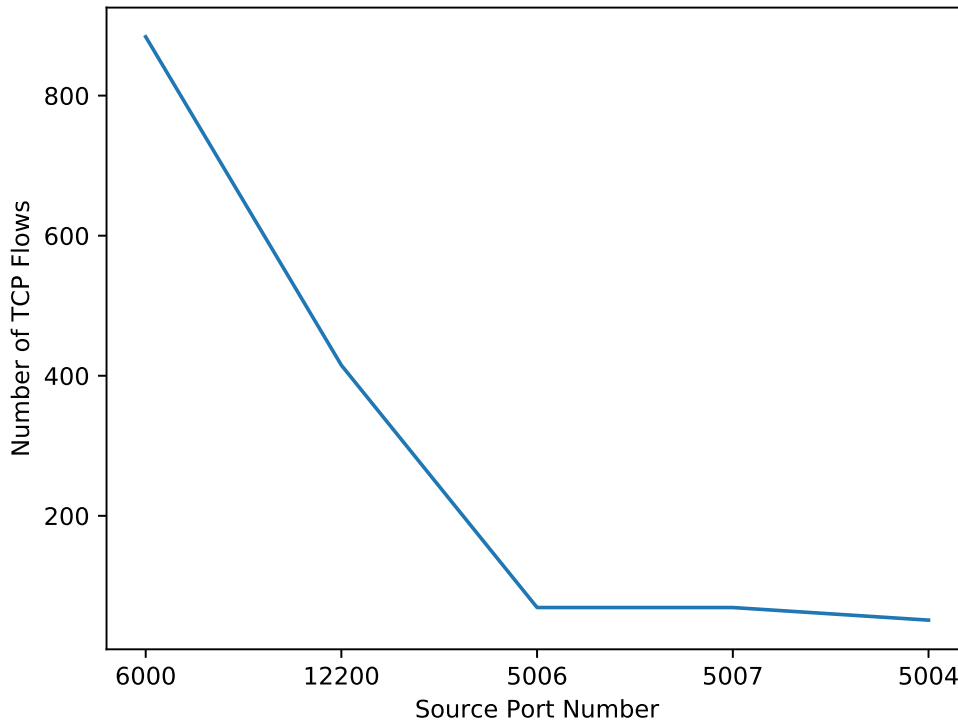


Figure 4.2: Source port distribution

## 4.4 Secure Shell (SSH) Traffic Analysis

### 4.4.1 Login Credentials

There were approximately 10,500 login attempts in data set 1 (D1) and 107,782 login attempts in data set 2 and 3 (D23). In a single TCP flow, an attacker can try 20 failed login attempts before termination of the connection [14].

Most of the login attempts to the SSH server were failed ones. There were only a few successful login attempts. There were two types of successful logins.

1. After a successful login, the client immediately terminated the connection.

2. After a successful login, the client executed a few couple of commands in the Terminal and then terminated the connection.

Hofstede et al. [81] made the same observation while studying different attack tools. Most of the login attempts after a successful connection did not execute any commands. Our hypothesis is that the attackers shared the login credentials with other hackers or sold in blackhat community. Other hackers would later on login to the compromised server to execute some malicious activities. Moreover, there were occurrences of TCP flows where the first login attempt was successful. In these TCP flows, the attacker executed several commands.

#### 4.4.2 SSH Client Library

Table 4.3: Most used SSH client library

Library version	Total count	Percentage
SSH-2.0-PUTTY	29302	47.15
SSH-2.0-Ganymed Build_210	4476	7.2
SSH-2.0-libssh-0.1	3745	6.03
SSH-2.0-libssh2_1.4.2	3506	5.64

The plaintext packet in the SSH Transport Layer protocol contains the SSH client library name. It is transmitted in the SSH handshaking packet. The client version is used to identify the attack tool. There were many different SSH client versions, and their version numbers were also different. Table 4.3 depicts the most often used client SSH library for combined data sets (data set 1, data set 2, data set 3).

### 4.4.3 Login Attempts in a TCP flow

Table 4.4: Number of login attempts in a TCP flow

No. of attempts in a TCP flow	No. of TCP flows (D1)	% (D1)	No. of TCP flows (D23)	% (D23)
1	5677	53.8	10886	23.69
2	448	4.25	2328	5.07
3	127	1.2	21806	47.46
4	62	0.59	108	0.24
5	62	0.59	419	0.91
6	41	0.39	32	0.07

Table 4.4 shows the count and the percentage of TCP flows that had different login attempts from one to six. In data set 1, 6417 TCP flows (60%) had less than or equal to six login attempts per flow. In data set 2 and 3 (D23), 35,579 flows (77%) had less than seven login attempt per flow. Hence, most of the malicious traffic is designed to have less number of login attempts per flow.

The default configuration in OpenSSH server is to allow only a maximum of six login attempts in a TCP flow. Moreover, fail2ban security tool blocks login attempt after three login failures in a duration of ten minutes.

## 4.5 Model to Segregate Attack Instances

In Figure 4.3, a flow chart is shown that segregates individual attack instances that carry out stealthy SSH password guessing attacks. The first block is already

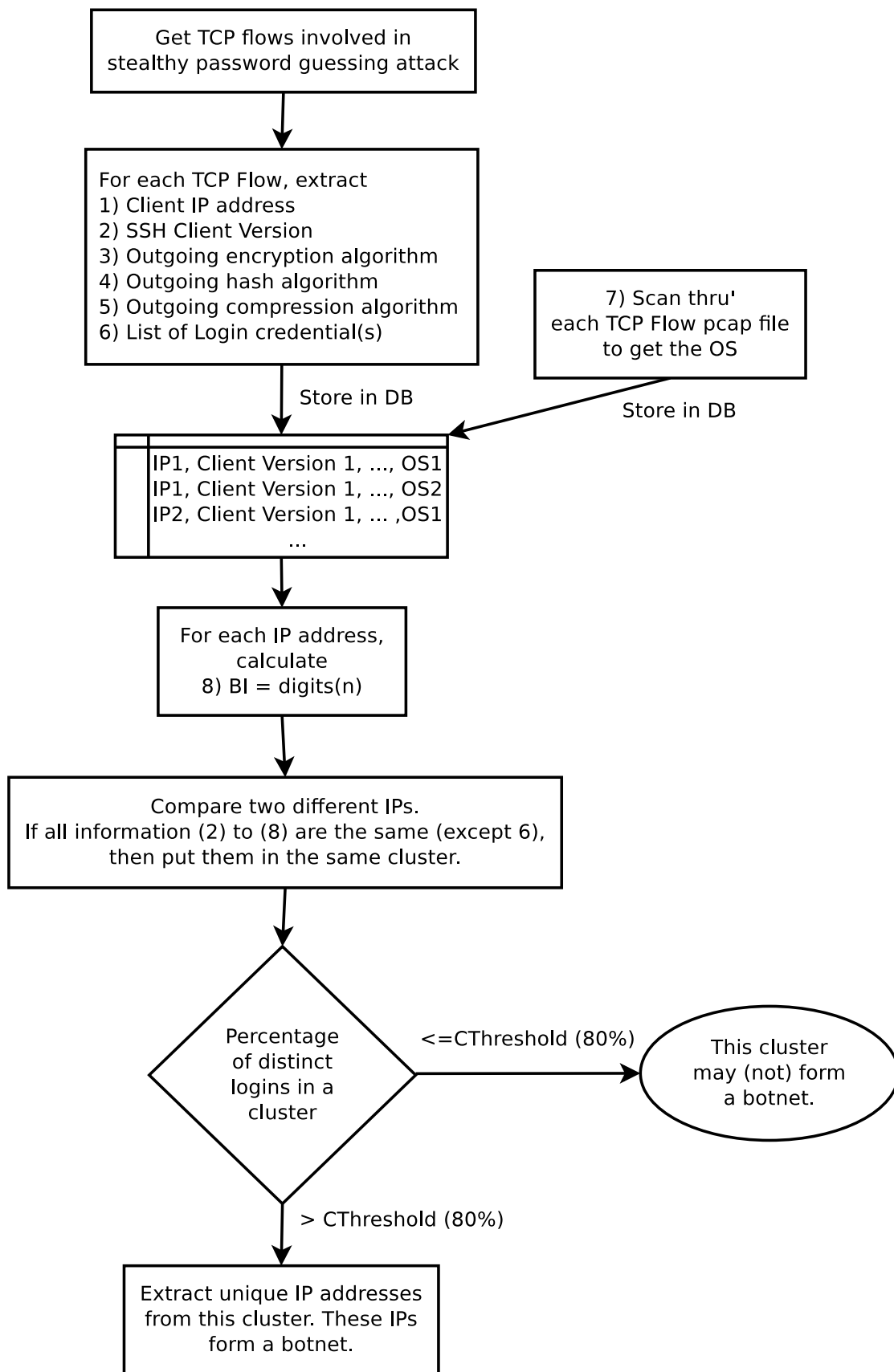


Figure 4.3: Flow chart for segregating attack instances



discussed in Section [4.2.2](#).

### **4.5.1 Extraction of Stealthy SSH Password Guessing Attacks**

The log files of Kippo honeypot has both stealthy SSH password guessing attacks and non-stealthy SSH password guessing attacks. Java programs were written to extract stealthy SSH password guessing attacks from the log files. The parameters ( $x=3$ ,  $y=600$  seconds,  $z=600$  seconds) was used to classify the attacks. These parameters are the default values of “fail2ban” security tool ([Section 1.5](#)).

The output of this method is the list of TCP flows pertaining to stealthy SSH password guessing attacks.

### **4.5.2 Store Specific TCP Flow Information**

For each TCP flow in the stealthy SSH password guessing attack, the following details are extracted and stored in a separate location.

1. Client IP address
2. Operating System
3. SSH client version
4. SSH outgoing encryption algorithm
5. SSH outgoing hash algorithm
6. SSH outgoing compression algorithm
7. List of login attempts (username, password)

The Operating System is determined from the pcap files (Section 4.2) with the use of p0f tool [93] (passive fingerprinting tool).

The OS fingerprinting is done by analysing the network and transport layer headers of a packet. The two important fields used in detecting the Operating System are Time to-live (TTL) and TCP Window Size.

In a botnet executing SSH password guessing attack, all bots are written in the same language. The SSH client library is the same in all the bot programs. The bot program with the library is made compatible with the Operating System of the machine. Moreover, the choice of the encryption algorithm, the hash algorithm, and the compression algorithm depends on the version of SSH installed on that specific Operating System. Hence, the above information is highly valuable in detecting a botnet instance.

All the above information is stored in a separate location grouped by source IP address.

### 4.5.3 Segregation of Attack Instances

Behavioural index (BI) tells the nature of login attempts from an attacker. It is the number of digits in an integer. For example, the *BI* for 2 is 1, and the *BI* for 850 is 3. For each set of flows belonging to a particular IP address, the behavioural index is calculated. In Figure 4.3,  $n$  is the total number of login attempts from an IP address and  $digits(n)$  returns the number of digits in  $n$ .

The flows of two different IPs are compared. If the details: SSH client version, SSH outgoing encryption, SSH outgoing hash algorithm, SSH outgoing compression algorithm, OS, BI, are the same, then they are placed in the same group.

Finally, each of the login credential (username, password, authentication type)

is inspected within a group. A login credential is a combination of the username, the password, and the authentication type. Next, the number ( $N_1$ ) of distinct login credentials is found. (For example, if the list of login credentials is  $\{u1,p1,a1\}$ ,  $\{u2,p1,a1\}$ , and  $\{u1,p1,a1\}$ , then distinct login credentials are  $\{u1,p1,a1\}$  and  $\{u2,p1,a1\}$ , and  $N_1=2$ ). If the percentage of distinct login credentials ( $(N_1 * 100/N)$ , where  $N$  is the number of login credentials ) is greater than a threshold, then with high probability we can infer that the group forms a botnet or a single-source attacker.

## 4.6 Results and Discussion

### 4.6.1 Results

In Table 4.5 and Table 4.6, the column  $N_{ip}$  is number of distinct client IPs, the column  $N_t$  is the total number of login attempts, and the column  $N_u$  is the percentage of unique login attempts.

Table 4.5: Segregation of attack instances - data set 1

Common string	$N_{ip}$	$N_t$	$N_u$ (%)
Windows XP,SSH-2.0-libssh-0.4.6, aes256-ctr,hmac-sha1,none,1	1	2	1.00
Linux,SSH-2.0-libssh2_1.4.1, aes128-ctr,hmac-sha1,none,1	1	2	1.00
Linux,SSH-2.0-libssh2_1.4.1, aes128-ctr,hmac-sha1,none,4	2	1024	0.83

In data set 1, there were three different groups (or botnets) as shown in Table 4.5. All different variants of Linux are combined into the group “Linux”. The first column is the common string used to group a set of sessions.

In the last row of Table 4.5, two IP addresses were involved. The first IP address tried several login attempts (850) over a 20 days period, where the inter-arrival time between each attempt was nearly 30 minutes. The second IP address tried 174 login attempts over a four days period, where the inter-arrival time between attempts was nearly 30 minutes. In both the sessions, the number of login attempts in each TCP flow was mostly one.

A similar trend is observed in data set 2 and data set 3 (combined together).

Table 4.6: Segregation of attack instances - data set 2 and data set 3

Common string	$N_{ip}$	$N_t$	$N_u$ (%)
null,SSH-2.0-libssh2_1.8.1-20170115, aes128-ctr,hmac-sha1,none,1	3	8	0.88
null,SSH-2.0-libssh2_1.4.1,aes128-ctr, hmac-sha1,none,1	1	1	1.00
null,SSH-2.0-OpenSSH_6.9p1 Ubuntu-2, blowfish-cbc,hmac-md5,none,1	1	2	1.00
null,SSH-2.0-PUTTY,aes128-ctr ,hmac-sha1,none,1	6	20	0.90

Common string	$N_{ip}$	$N_t$	$N_u$ (%)
null,SSH-2.0-7.15 FlowSsh: Bitvise SSH Client (Tunnelier) 7.15 - stnlc, aes256-ctr,hmac-sha1,none,1	4	4	1.00
null,SSH-2.0-paramiko_2.4.0,aes128-ctr, hmac-sha1,none,1	1	1	1.00
null,SSH-2.0-OpenSSH_6.7p1 Raspbian-5, blowfish-cbc,hmac-md5,none,1	1	2	1.00
null,SSH-2.0-paramiko_2.3.1,aes128-ctr, hmac-sha1,none,1	3	15	0.87
null,SSH-2.0-libssh2_1.7.0,aes256-cbc, hmac-sha1,none,1	1	5	1.00
null,SSH-2.0-nsssh2_4.0.0034 NetSarang Computer, Inc.,blowfish-cbc,hmac-md5,none,1	1	2	1.00
null,SSH-2.0-PuTTY_Release_0.69, aes128-ctr,hmac-sha1,zlib,1	10	10	0.80
null,SSH-2.0-paramiko_1.10.1,aes128-ctr, hmac-sha1,none,1	1	1	1.00
null,SSH-2.0-libssh2_1.7.0,aes128-ctr, hmac-sha1,none,1	22	179	0.92
null,SSH-2.0-OpenSSH_4.3,aes128-ctr, hmac-sha1,none,1	14	51	0.92

## 4.6.2 Limitations

The model does not differentiate stealthy single-source SSH password guessing attacks from stealthy distributed SSH password guessing attacks. Each group (cluster) represent a stealthy single-source attacker or a stealthy distributed attacker.

The model assumes that the SSH client version is statically stored in the configuration file of the client machine. The attacker could upgrade the malware by dynamically changing the SSH library name in the SSH handshaking packets. The attacker can also change the outgoing encryption algorithm, outgoing hash algorithm, and the outgoing compression algorithm.

The input to the model is TCP flows. Instead, the model can be improved by feeding TCP sessions (as described in [Section 1.4](#)).

## 4.7 Conclusion

The approach is simplistic and helps in differentiating instances of stealthy SSH password guessing attacks. In other words, the methodology identifies individual attackers (or a group of attackers) involved in stealthy SSH password guessing attacks.

The honeypot log files are the input to the model. The model parameters are the fields of SSH conversation. These fields (SSH client version, Usernames, Passwords) are minutely scrutinized, and their significance is given.

Javed and Paxson [53] identified instances of stealthy SSH password guessing attacks. Their model finds commonality in the set of usernames or a set of server IP addresses (in the case of multiple servers). Our approach includes the fields in

the SSH Transport Layer protocol and the SSH Authentication protocol. It finds commonality by inculcating several fields.

# Chapter 5

## Detection of Stealthy Single-Source SSH Password Guessing Attacks

### 5.1 Introduction

There are two types of stealthy SSH password guessing attacks: one generated by a single-source, another generated from a distributed network. The former is generated manually, and the latter is generated from a botnet. A comprehensive description of these attacks is explained in [Section 1.5](#).

Several methods [[70](#), [53](#), [71](#), [72](#)] were proposed to detect stealthy distributed and coordinated SSH password guessing attacks. As such, there is no research work that particularly describes a method to detect stealthy single-source password guessing attacks in honeynet traffic. Javed and Paxson [[53](#)] described the hardness of segregating stealthy single-source SSH password guessing attacks and stealthy



distributed SSH password guessing attacks. It is difficult to differentiate because both these attacks make login attempts at a low rate. In honeynet traffic, stealthy single-source and stealthy distributed SSH password guessing look similar and hard to classify them into different groups.

These two attacks cannot be differentiated by analysing the server logs and by analysing the login attempts. The contents of server logs (OpenSSH) are client IP address, client port number, and login username. A honeypot (Kippo) stores additional information like client SSH library version, the list of negotiated cryptographic algorithms, the authentication type, and the password. Both single-source attacker and botnet use standard SSH protocol [13, 14, 15] to attack the server. With this information, it is impossible to determine the nature of the client (whether it is a stealthy single-source attacker or a stealthy distributed attacker).

Assume a client IP address  $x_1.x_2.x_3.x_4$  makes  $N$  login attempts on a particular SSH server. The  $N$  login attempts are made on different times (For example, if  $N$  is 4, then the first login attempt is made at time  $t_1$ , second at  $t_2 = t_1 + 2$  days, third at  $t_3 = t_2 + 55$  minutes, and the fourth at  $t_4 = t_3 + 23$  hours.). It is hard to determine whether the attack is from a single-source or a distributed network (a bot during the infection phase have chosen the same server IP address multiple times).

This thesis is making an attempt to segregate stealthy single-source SSH password guessing attacks from stealthy distributed SSH password guessing attacks. Specifically, there are two main contributions in this work.

1. In the data extraction stage (Section 5.2), the method to extract the sessions is enhanced by using TCP flows.
2. The features (Section 5.3) to differentiate stealthy single-source SSH pass-

word guessing attacks from stealthy distributed SSH password guessing attacks are identified. For feature 1, a mathematical analysis is carried out. A mathematical expression is given for feature 2. Among the different possible ways of estimating the total duration, the appropriate ones are chosen for feature 3 and feature 4. For all the features, a detailed theoretical justification is given.

3. The application (Section 5.4) of the clustering technique (specifically, density-based clustering) is explored and justified with proper reasoning.

This chapter is organized as follows: Section 5.2 describes the steps to extract stealthy SSH password guessing attacks from the Kippo log files. Section 5.2 also explains the model to decide the session gap and the two different types of stealthy SSH password guessing attacks found in the log files. In Section 5.3, the four significant features involved in detecting stealthy single-source SSH password guessing attacks is explained with proper justification. Section 5.4 explains the model for detecting the attacks. The results are discussed in Section 5.5. Finally, the conclusion 5.6 describes the pros and cons of the model and the future work that is to be carried out in this domain.

## 5.2 Data Extraction

### 5.2.1 Session

The TCP flows (mentioned in Section 4.2.2) cannot be directly used as the input to the proposed model. The purpose of a session is described in Section 1.4. A method to determine the session gap was proposed by Alata [18]. In their technique, the number of sessions for different values of session gap is initially

plotted. There occurs a region where the slope of the curve changes drastically and reaches approximately zero. The region where the slope of the curve is zero occurs when the number of sessions does not change much.

Alata's method was used to determine the session gap for the data set. Alata used inter-arrival time of packets as session gaps. In our case, all the packets are using TCP protocol and inter-arrival time between the last packet of first TCP flow and the first packet of the second TCP flow can be used to calculate the session gap.

Figure 5.1 shows a graph obtained using Alata's method on our data set (data set 1, data set 2, and data set 3). In Alata's method, the y-axis is the number of sessions. In this work, the y-axis is changed to the ratio of the number of sessions to that of the maximum sessions possible. This is done to give a better picture of the number of sessions. In other words, the percentage of sessions is used in the y-axis. The session gap obtained using this technique is 100 seconds.

Even though sessions are obtained using this technique, sometimes two sessions might originate from the same machine. This is the limitation of Alata's [18] technique. To consider this, an approach is followed by us to merge certain sessions. If two or more consecutive sessions have the following information which is same in its sessions, then we merge them.

1. Client IP address
2. SSH Application layer information (SSH Transport sub-protocol)
  - (a) SSH Client Identification String
  - (b) Key Exchange Algorithm
  - (c) Server Host Key Algorithm

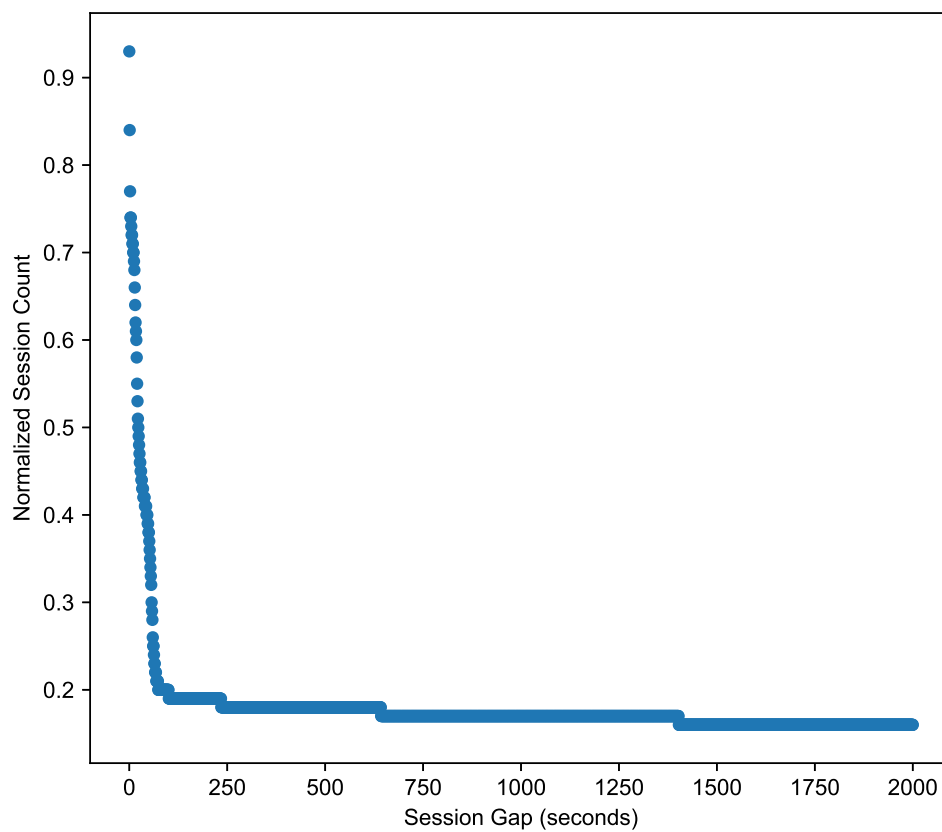


Figure 5.1: Variation in normalized session count with change in session gap

- (d) SSH Encryption Algorithm
- (e) SSH Hash Algorithm
- (f) SSH Compression Algorithm

## 5.2.2 Extraction of Stealthy SSH Password Guessing Attacks

A Java program (as described in Section 4.5.1) was written to extract stealthy SSH password guessing attacks.

```

Result: stealthySessions
sessions = getAllSessions();
foreach session  $\in$  sessions do
    | isStealthy = checkIfStealthy(session);
    | if isStealthy then
    | | stealthySessions.add(session);
    | end
end

```

### Algorithm 1: Segregation of Sessions

There are 590 sessions which are stealthy SSH password guessing attacks. The number of TCP flows in each session varied from 1 to 97. Also, the number of login attempts per session varied from 1 to 97.

As per Javed and Paxson [53], the amount of stealthy distributed SSH password guessing attacks is much higher than stealthy single-source SSH password guessing attacks.

### 5.2.3 Stealthy SSH Password Guessing Attacks

There are several variations of stealthy SSH password guessing attacks. Two different extreme cases found in the data set are discussed below.

1. An attack source (consider a machine with IP address  $x_1.x_2.x_3.x_4$ ) attacked only once (only one login attempt) for a long duration (several months). For example, the client machine (IP address:  $x_1.x_2.109.20$ ) made only one login attempt (username: “admin”, password: “admin”, login date: “10-July-2017”) during entire duration of the honeynet execution. In the case of a legitimate user, in most cases (more than 60%), there would be a successful login attempt after one failed login attempt [53]. Hence the appearance of only one failed login attempt in a long duration means there were several machines trying to attempt only once on the server machine. Moreover, it would be hard to block such a login attempt in the server-side.
2. An attack source (consider IP address  $x_1.x_2.x_3.x_4$ ) made several login attempts and the inter-arrival time between the login attempts is several minutes/days/months. For example, the IP address  $x_1.x_2.115.55$  made 97 login attempts and the minimum time difference between any two consecutive login attempts is 2435 seconds (40 minutes). A plot showing the frequency of the inter-arrival times (IAT) is shown in Figure 5.2.

## 5.3 Data Collection and Feature Analysis

There are several features that will differentiate between stealthy single-source SSH password guessing attack and stealthy distributed SSH password guessing attack. The features are described in-depth in the following sub-sections.

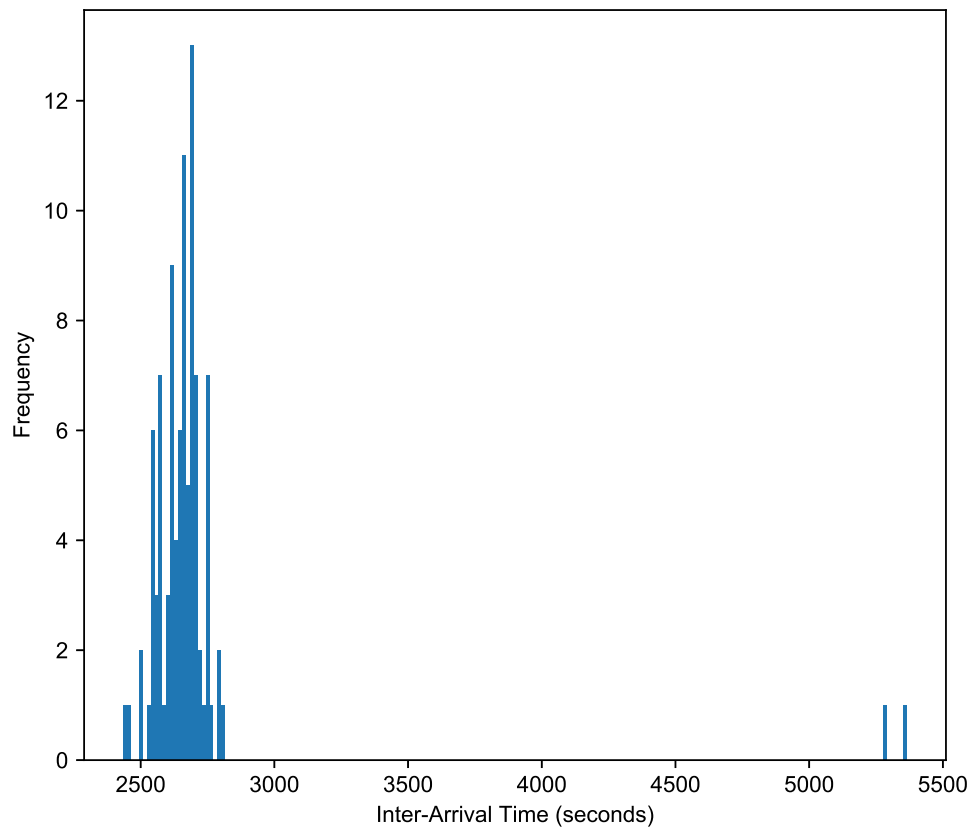


Figure 5.2: Persistent Stealthy Attack by the IP address  $x_1.x_2.115.55$

Each of the features is calculated for a given session. Each feature is also represented numerically to be used in the machine learning model. Each one of the features is obtained from domain knowledge, and the solid reason is given for its appearance in this work. The list of the features is given below.

1. Number of login attempts in a session.
2. Percentage of login attempt repetition in a session.
3. Time difference between the first port scanning flow and the first TCP flow with a login attempt.
4. Minimum time difference between two TCP flows with login attempts.

### 5.3.1 Number of Login Attempts

This feature is the total number of login attempts made by a client  $x_1.x_2.x_3.x_4$  in a session. Each login attempt comprises a single username and a single password.

Let  $N$  be the total number of login attempts needed to break into an SSH server.  $N$  is an average value, and many mathematical studies are done to analyse it. Hence, an attacker (either single-source or distributed) should try all the  $N$  logins before logging successfully into the system.

For a stealthy single-source attacker, all  $N$  login attempts are made by him. For a stealthy distributed attacker, the  $N$  logins will be shared by the botnet. Let  $B$  be the number of bots in a botnet. In the bootstrap stage [38] of the botnet, there will be only a few bots in the botnet. After several minutes or hours, a botnet might contain several thousands of bots. In the steady-state (which occurs within a few days), the number of bots in a botnet is usually 100 thousand. [38]. To efficiently



make use of the botnet, the botmaster will distribute the login credentials to the bots. There are two ways to achieve this:

1. This occurs during the attack phase. The technique is followed by a stealthy distributed and coordinated password guessing attack. The C & C server divides the  $N$  login credentials among the bots. The C & C server sends  $N' = N/B$  login credentials to each bot. Each bot tries only  $N'$  logins on the SSH server. All the bots attack only the same SSH server. The target IP address of the SSH server is sent by the bot-master (C & C server). In the server-side, several client IP addresses would have made login connections to the server. But, for each client, only  $N'$  login attempts would be found.  $N'$  is much lesser than  $N$ , and is used to differentiate between a single-source attacker and a distributed attacker.
2. This occurs during the infection phase. The technique is followed by a stealthy distributed and uncoordinated password guessing attack. The entire login list is hard-coded in the bot program. The C & C server does not send the login list through the network to the bots. When the bot program is written, the programmer (the hacker) has hard-coded the list [38]. Each bot randomly chooses the victim IP address. Since password guessing is done to propagate malware, the IP address of a vulnerable SSH server is chosen randomly by each bot. It is highly likely that two or more bots will target a particular SSH server. Hence, it is not necessary for each bot to try all the  $N$  logins on an SSH server. Therefore, each bot randomly chooses  $R$  ( $\ll N$ ) logins for each victim server. Over a long duration, a particular victim server would have encountered all the logins ( $N$ ) from different client machines. In a Mirai botnet [33],  $N$  logins were only the default (or factory)

logins of IoT devices.  $N$  took a low value (around 30).

After obtaining this feature from all sessions, it is indicative that a high number of logins in a session is likely a stealthy single-source SSH password guessing attack. A low value of the number of logins in a session is a stealthy distributed SSH password guessing attack. It is not always true. For example, consider a use case as follows: A single-source attacker tries only a few logins (less than 10), and all the logins are default or factory logins of IoT devices. In this case, the attacker, after trying these logins, gives up and goes to the next victim server. In other words, the attack mimics the behaviour of stealthy distributed SSH password guessing attack. The attack could also be enhanced by trying each login on all the server IP addresses and then moving on to the next login in the list.

### 5.3.2 Repetition of Login Attempts

Nicomette et al. [19] describes the sharing of dictionaries among different attackers. Clustering of different dictionary attacks shows that more than 20% of the attacks have high-correlation among the set of login credentials. In other words, for some of the attackers, the set of login credentials (i.e., both username and the password) were the same. They do not mention the repetition by the same IP address.

There are two works [61, 58] that discusses the repetition of the same logins (same username and the same password) by the same attacker. As per Abdou et al. [61], the reattempting of the logins occur because of several reasons: (a) User might have changed the password after a long gap. Hence, reattempting the same login after a long time might lead to a successful connection. (b) It is due to a bug in the bot program that leads to repetition of the same login credentials on the same victim server. According to Romain et al. [58], it is the reason (a) mentioned

by Abdou et al.

As per our findings, each bot is hard-coded with a list of login credentials. Each bot randomly tries a set of logins on a victim machine. The same bot tries a couple of same logins on the same server again. In a long duration, it is highly likely that the same victim is chosen by the same bot with a few repeating logins.

In a stealthy single-source SSH password guessing attack, the source machine is totally aware of the victim server and the login list. Hence, there is no chance of a repetition of login credentials.

In the case of a stealthy distributed SSH password guessing attack, repetition of login credentials occurs. Hence, the repetition of login credentials in a session is used to determine whether the attack is from a single-source or a distributed entity.

The challenge is to represent this repetition in the form of a numeric value to be used in the machine learning model. Shannon entropy [94] finds the number of distinct items and provide a representation of these items in a group. The repetition is quantified using Shannon entropy. Equation 5.1 denotes the repetition of logins in a session.

$$RM = 1 - \frac{Entropy}{\log_2 N} \quad (5.1)$$

$(N > 1)$

In Equation 5.1,  $N$  is the number of login attempts made by a client machine in a particular session (Note: it is different from  $N$  used in the Section 5.3.1. *Entropy* is the Shannon entropy.  $RM$  is the Repetition Metric and ranges between 0 and 1. When  $N = 1$ , there are no more logins to check the repetition and the value of  $RM$  is taken as zero.

The intuition behind Repetition Metric is explained using the below example. Let  $N = 4$  (i.e., four login attempts in a session),  $\{u,p\}$  denotes a particular username and a specific password. Here,  $u_i$  and  $u_j$  are two different usernames ( $1 \leq i \leq 4, 1 \leq j \leq 4$ ).

(a)  $\{u_1, p_1\}, \{u_1, p_1\}, \{u_1, p_1\}, \{u_1, p_1\} \Rightarrow$  RM is 1

(b)  $\{u_1, p_1\}, \{u_1, p_1\}, \{u_1, p_1\}, \{u_2, p_2\} \Rightarrow$  RM is 0.59

(c)  $\{u_1, p_1\}, \{u_1, p_1\}, \{u_2, p_2\}, \{u_2, p_2\} \Rightarrow$  RM is 0.5

(d)  $\{u_1, p_1\}, \{u_1, p_1\}, \{u_2, p_2\}, \{u_3, p_3\} \Rightarrow$  RM is 0.25

(e)  $\{u_1, p_1\}, \{u_2, p_2\}, \{u_3, p_3\}, \{u_4, p_4\} \Rightarrow$  RM is 0

When all the usernames and passwords are different, then  $RM$  is equal to 0. When all the usernames and the passwords are the same, then  $RM$  is equal to 1. In the second and the third row, only two usernames and two passwords exist. But, the first username is repeated three times in the second row and repeated only two times in the third row. Hence, the former is given a slightly higher  $RM$  than the latter.

Therefore, a high value of  $RM$  indicates a stealthy distributed SSH password guessing attack and a low value of  $RM$  indicates a stealthy single-source SSH password guessing attack.

There is a use case where the above will not satisfy: Consider a distributed attacker is not repeating the logins by improving the botnet malware. Under such a situation,  $RM$  would not completely describe the nature of the attacker.

### 5.3.3 Port Scanning and Password Guessing

Port scanning is a technique to determine the open or closed or filtered ports on a network device. There are various ways of doing port scanning. Some of them are SYN scan, ACK scan, FIN scan, and XMAS scan. Every single method is designed in such a way to exploit the nature of the network protocol. For example, in an SYN scan, the network device will always reply with an SYN-ACK packet. Hence, the client knows that the port is open.

Botnets do a very fast port scanning to extract all the SSH servers that have port 22 or 2222 open [38]. Mirai botnet [33] did port scanning on Telnet server (port 23/2323). The port scanning was done before launching the password guessing attack. Mirai variants targeted SSH port.

Port scanning is a prerequisite for password guessing. In the case of a single-source attacker, there is a delay between the port scan and password guessing. This occurs because none of the tools that involve in single-source password guessing is capable of doing port scanning. Moreover, it is not the requirement of these tools to do port scanning because their purpose is to do penetration testing by white hat hackers. In the case of a distributed attacker, the delay between the port scan and password guessing is considerably less.

The feature is the time difference between port scanning and the first login attempt. It is taken as the start time of the port scan and the start time of the TCP flow containing the first login attempt. For some cases, the calculation of this feature was tedious. The following assumptions were made to counter it.

1. When the port scanning is not available for a session, then the feature value is taken as “Not Available”.
2. If there are multiple port scans and login attempt combinations (in a session),

then the minimum duration is taken.

### 5.3.4 Nature of TCP Connections

Multithreading is implemented in all the single-source password guessing tools to reduce the overall time to crack a system. Multithreaded password guessing is achieved by creating multiple threads and assigning a partial login list to each thread. All the different threads attack the server at the same time. In the server logs, multiple TCP flows appear from the same client machine.

Multithreading leads to consumption of more RAM space and more network bandwidth. Otherwise, the total time to crack a password is considerably reduced.

On the computer resources of a single-source attacker, there is an option of using multi-threaded password guessing tools. The tools were willingly installed by the hacker. In the case of a distributed attacker, the tool (bot malware) is hidden from the real user of the network device (desktop, laptop, IoT device, server). Hence, a distributed attacker does not use multi-threading. Mirai botnet does not use multiple threads to attack a single host.

Consider a single-source attacking machine making multiple connections to the server. Each connection is associated with a thread. Figure 5.3 shows two TCP flows originated from the same client. The TCP flows are gathered in the server-side. In this figure, the y-axis represents the TCP flows. For TCP flow 1, the time when the SYN packet was received by the server is  $t_1$ . For TCP flow 2, the time when the SYN packet was received by the server is  $t_2$ . It is obvious that TCP flow 1 started before TCP flow 2.  $\Delta t$  is the time difference between  $t_1$  and  $t_2$ .

$\Delta t$  is used to represent this feature. A large of  $\Delta t$  indicates the absence of multi-threading. A low value of  $\Delta t$  conveys that multi-threading is used. For the

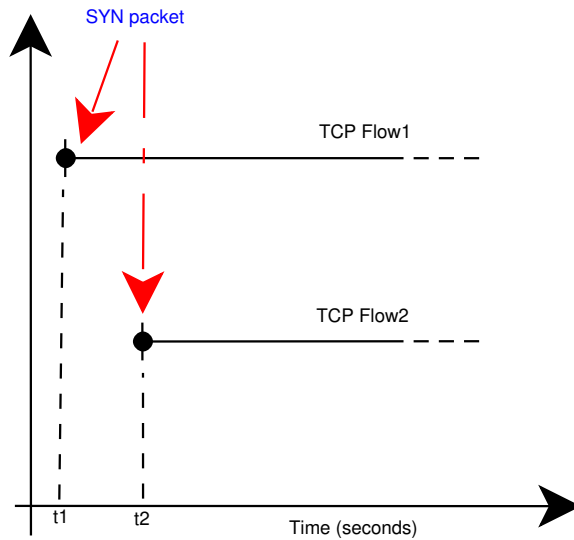


Figure 5.3: Effect of multithreading on the TCP flows

sake of the machine learning model, the upper limit of  $\Delta t$  is taken as 600 (detection time used in “fail2ban” security tool).

It can be proved that the TCP flows were created using multithreading by tracing the identification fields (IPID) in the individual packets. If the client uses a global counter for IPID field, then each successive outgoing packet from the client will have the IPID field incremented by one.

Figure 5.4 shows the identification numbers of the packets in two different TCP flows of  $x_1.x_2.70.26$ . The two TCP flows start at the same time. The packet with identification number 21023 (TCP flow 2) appeared between the packet with identification number 20995 (TCP flow 1) and the packet with identification number 21152 (TCP flow 1). Moreover, starting from 18441 (TCP flow 1) many of the identification numbers are assigned alternately between TCP flow 1 and TCP flow 2. Hence, both these TCP connections were created by different threads/processes in the client machine.

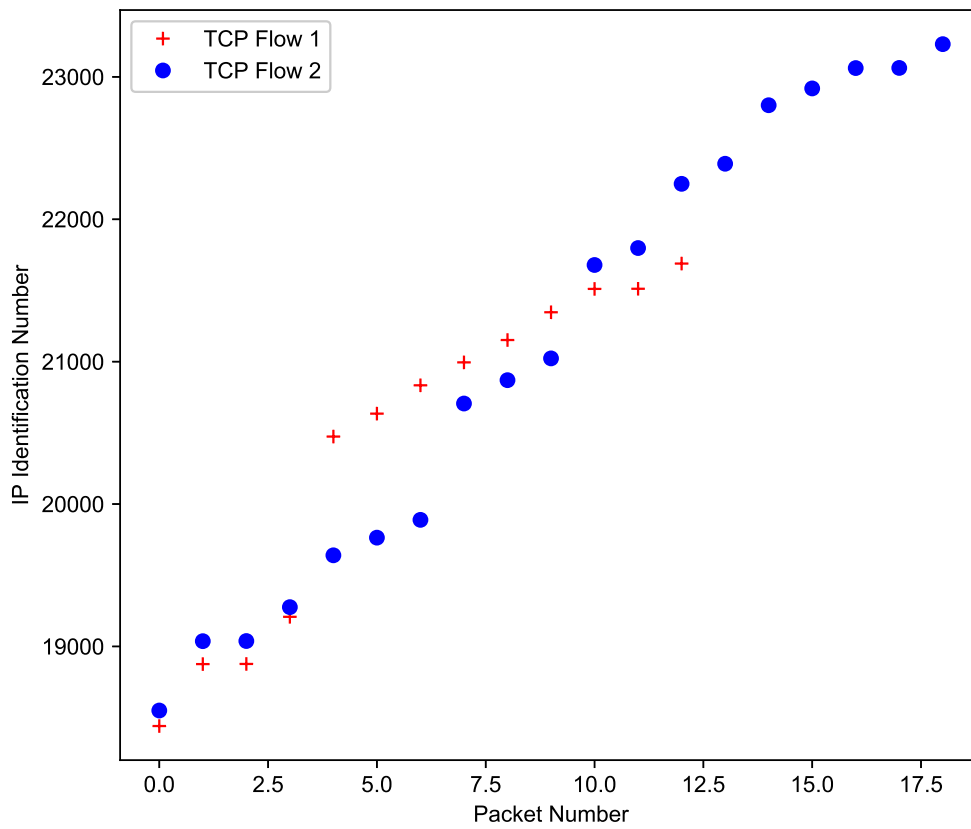


Figure 5.4: Identification numbers of two TCP flows of  $x_1.x_2.70.26$



A low value of  $\Delta t$  indicates a stealthy single-source attacker and high value indicates a stealthy distributed attacker.

## 5.4 Model Fitting of the Data

### 5.4.1 Reason for Clustering Model

Clustering is chosen as the method to segregate these attacks: stealthy single-source SSH password guessing attack and stealthy distributed SSH password guessing attack. The reason for the choice of clustering is mentioned below.

1. All bots in a botnet have similar features and form a cluster in the feature space.
2. There are density-based methods in clustering that will separate the outliers (noise points). In the following section, it is shown that an outlier forms a stealthy single-source SSH password guessing attack.
3. Moreover, the pcap files and the Kippo log files cannot be manually inspected to assign labels. There is no methodology found in the literature to assign the labels. Also, there is no publically available data set with labels.

Hence, clustering was chosen as the technique to differentiate the attacks.

The entire model is portrayed in Figure 5.5. Each one of the blocks is described in the below paragraphs (except the first block).

### 5.4.2 Feature Distribution

All the four features portrayed a heavy tail distribution (Figure 5.6). Heavy tail distributions naturally make clusters. In Figure 5.6, *Feature 1* is the number

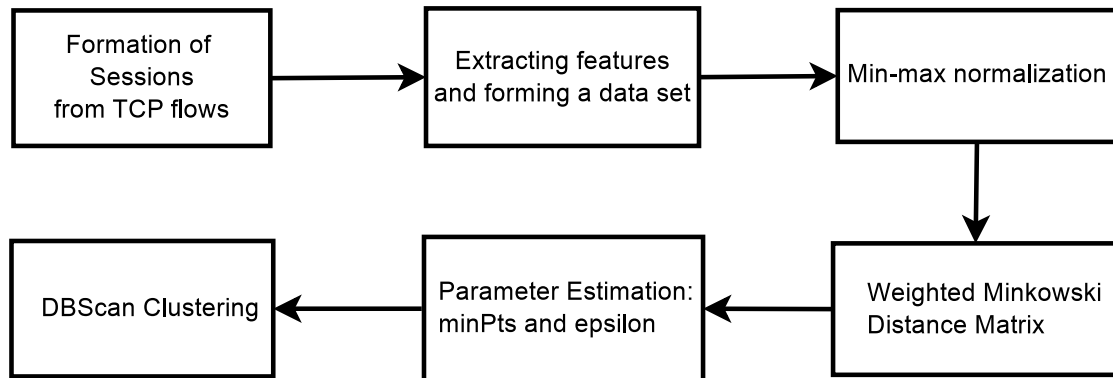


Figure 5.5: Clustering Model

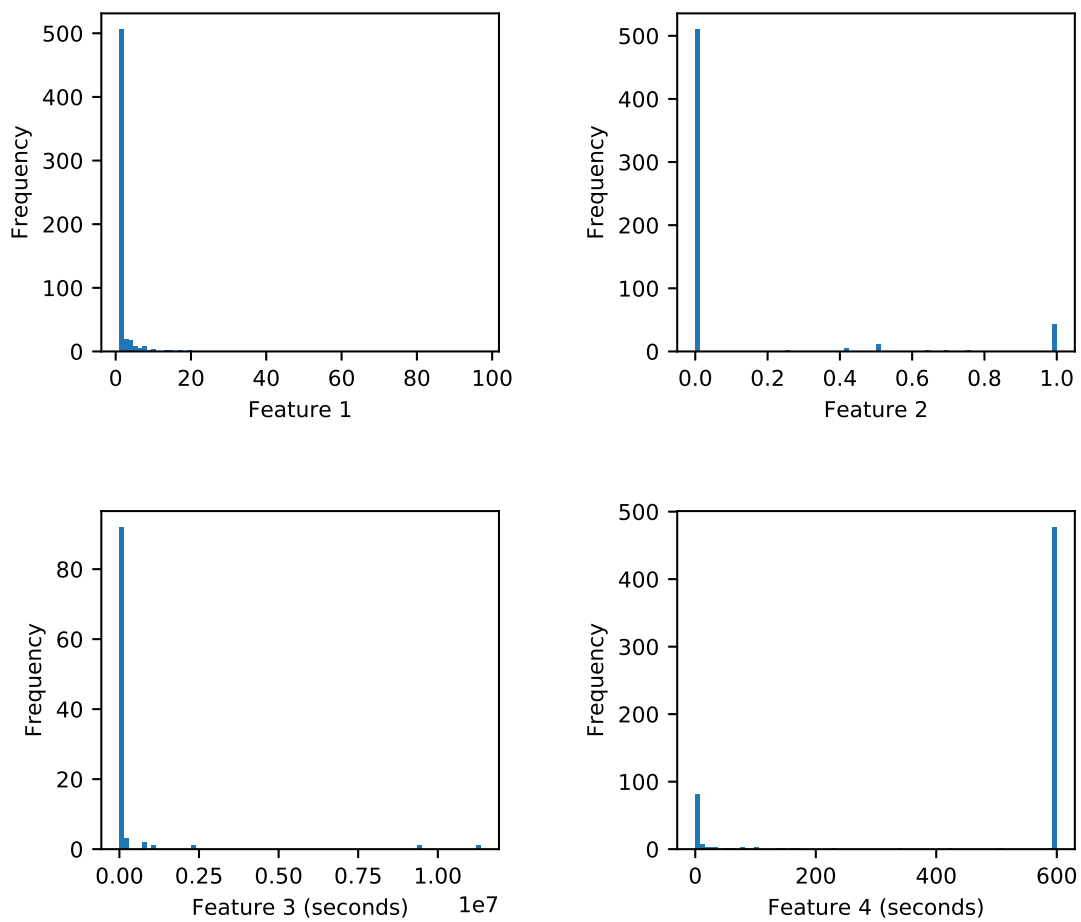


Figure 5.6: Distribution of the features

Table 5.1: Parameters of the Distribution

Feature No.	Minimum Value	Maximum Value	Skewness
Feature 1 (f1)	1	97	10.97
Feature 2 (f2)	0	1	2.47
Feature 3 (f3)	0	11353856	6.74
Feature 4 (f4)	0	600	-1.63

Table 5.2: Correlation between the features

	f1	f2	f3	f4
f1	1.000000	0.365205	-0.049399	-0.099109
f2	0.365205	1.000000	-0.108007	-0.085697
f3	-0.049399	-0.108007	1.000000	-0.009188
f4	-0.099109	-0.085697	-0.009188	1.000000

of login attempts in a session (Section 5.3.1), *Feature 2* is the repetition metric (Section 5.3.2), *Feature 3* is the time between the first scanning flow and the first login attempt (Section 5.3.3), *Feature 4* is the time difference between TCP flows within a detection window.

The minimum value of each feature, the maximum value of each feature, and the skewness of each feature is shown in Table 5.1.

### 5.4.3 Feature Selection

Table 5.2 shows the linear correlation between any two features. The maximum value in the correlation matrix is 0.365, and the minimum value in the correlation matrix is -0.108. The maximum and the minimum values are considerably lesser than +1 and -1. Hence, there is no correlation between the features. Therefore, there is no need to do feature selection or feature extraction. As described in Section 5.3, all the four features are relevant to this clustering problem.

### 5.4.4 Min-Max Normalization

The feature 2 (f2) varies only between 0 and 1, whereas all other features have an upper limit of a three-digit number. Feature 1 (f1) varies between 1 and 97, while the upper bound of feature 3 and feature 4 are greater than 500. Hence, while calculating the similarity between objects using a distance measure, the feature with high values (feature 3 and feature 4) dominate features with low values (feature 2 followed by feature 1). In other words, the final distance value is influenced mostly by feature 3 and feature 4. Hence, all features should be standardized to the same range (for example, between 0 and 1).

The two main techniques to scale the data to a finite range are min-max normalization and z-score normalization [95]. Since the distribution is not gaussian, z-score normalization is not a good fit for this data set. Moreover, the mean and the standard deviation computed in z-score normalization will be influenced by the outliers. Min-max normalization will only scale the high values to low values without affecting the distribution. Hence, min-max normalization is used for scaling the data. The min and max values are 0 and 1, respectively.

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}} \quad (5.2)$$

In Equation 5.2,  $X$  is the value of an attribute,  $X_{min}$  is the minimum value of  $X$  and  $X_{max}$  is the maximum value of  $X$ .  $X_{norm}$  is the normalized value, which ranges between 0 and 1.

### 5.4.5 Distance Function

The distance function used is weighted minkowski distance with  $r = 2$  [96].

$$d(x, y) = \left( \sum_{k=1}^4 w_k |x_k - y_k|^2 \right)^{\frac{1}{2}}$$

$$\sum_{k=1}^4 w_k = 1 \quad (5.3)$$

$$0 < w_k < 1$$

In Equation 5.3,  $x$  and  $y$  are data points in the normalized data set.  $x_k$  or  $y_k$  refers to the value of the attribute  $k$ .  $w_k$  is the weight for feature  $k$ . The summation of the weights should equal one, and each weight should be greater than zero and less than one. The weight for each feature is obtained from the relative importance of the feature in the clustering mechanism. In other words, the weights are speculated from the domain knowledge. The following technique is used to obtain the weights.

1. Consider only the feature  $i$  (fi) with weight  $w_i=1$ .
2. Apply DBSCAN and get the silhouette coefficient  $s_i$  (DBSCAN and Silhouette Coefficient is explained in-detail later).

3. Follow steps 1. and 2. for all the features (f1, f2, f3, f4)
4. Let total silhouette coefficient be  $s_t$  ( $s_t = \sum s_i$ )
5. The weight  $w_i$  is  $s_i/s_t$ .

The values of the weights obtained using the above method are:  $w_1 = 0.2749$ ,  $w_2 = 0.2993$ ,  $w_3 = 0.1489$ , and  $w_4 = 0.2769$ . The feature  $f3$  got low weight because it had several missing values.

There are a couple of ways to handle missing values [97]. Missing values do not contribute to the calculation of the euclidean distance and are ignored in this model. Only feature 3 (f3) had several missing values (84%).

### 5.4.6 Clustering

All bots in a botnet are the same executable. Hence, their behaviour is the same. When bots from the same botnet attack a server machine, the characteristics of the features in the server logs look similar. Therefore, the same bots would form a cluster in the server log files. Each cluster would represent a different botnet.

All single-source attacks are made by individual attackers. The characteristics of these features would appear difficult because each of the individual attacker behaviours would be different. Hence, the formation of a cluster is not possible. Moreover, these attacks form outliers in the data set.

In order to determine the stealthy single-source SSH password guessing, the model should display both the clusters and the outliers. This is capable of density-based clustering approaches. The simplest density-based approach is DBSCAN.

DBSCAN uses *minPts* and *epsilon* to create clusters. For different values of *minPts*, a graph similar to Figure 5.7 was obtained. Assume that two bots from

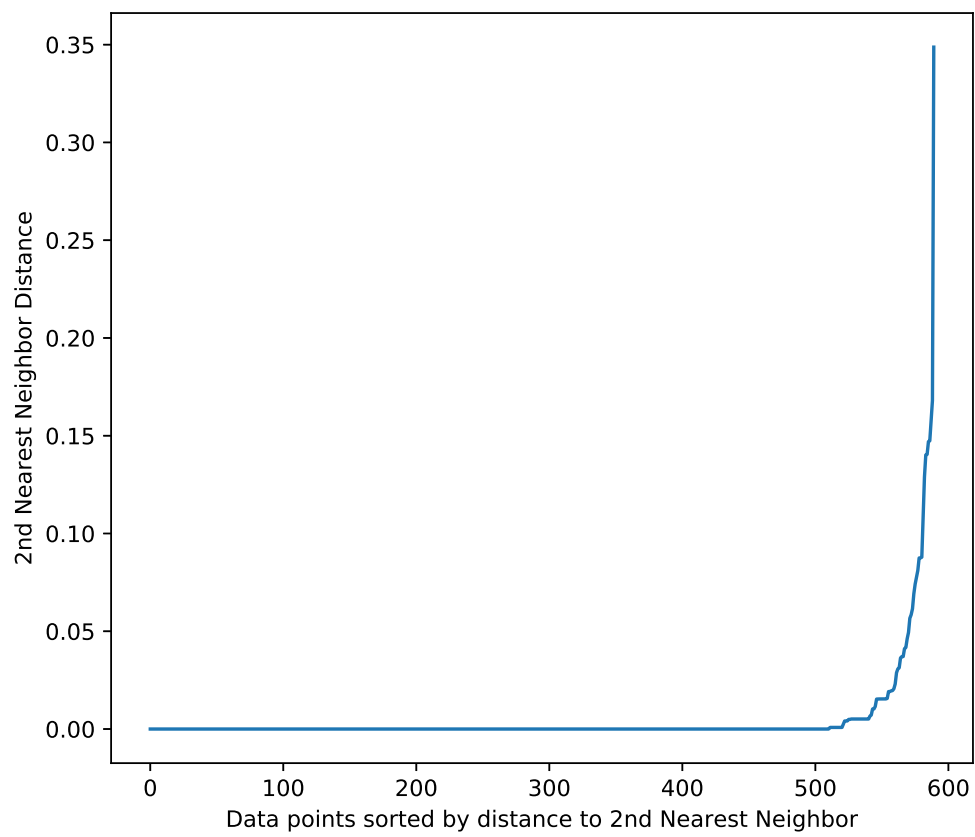


Figure 5.7: K-dist plot for normalized data

Table 5.3: Average value of the features

Cluster No.	No. of data points	f1	f2	f3 (s)	f4 (s)
Cluster 1	305	1.000	0.000	149201 (37)	600.000
Cluster 2	98	2.000	0.000	1892646.8 (5)	600.000
Cluster 3	24	2.000	1.000	0.143 (7)	600.000

the same botnet have attacked the SSH server. In this case, the features of these two bots (or sessions) would be similar. So, a cluster should have at-least two data points. Hence, *minPts* is taken as 2.

The parameter *epsilon* is determined using the  $k^{th}$  nearest neighbour method. In this technique, the weighted Minkowski distance of each data point to its  $k^{th}$  nearest neighbour is taken. Figure 5.7 shows the plot for  $2^{nd}$  nearest neighbour. The value of *epsilon* obtained through this method is  $4.40202835 \times 10^{-8}$ .

## 5.5 Results and Discussion

On the execution of DBSCAN, twenty one clusters were produced. The number of data points in each cluster is 2, 2, 2, 2, 2, 2, 3, 4, 4, 4, 4, 5, 5, 6, 7, 7, 7, 7, 24, 32, 98, 305. Table 5.3 shows the average of the feature vector for three clusters with the maximum number of data points. The feature *f3* had several missing values. Hence, in Table 5.3, for *f3* the value inside the parentheses represents the number of data points with numeric values (Cluster 1: 37 data points out of 305 data points had numeric values. For 268 data points, the value of *f3* is missing). There were 525 core points, 0 border points, and 65 noise points. The noise points had the



following average values for the features:  $f_1$  (10.123),  $f_2$  (0.426),  $f_4$ (221.384). There were 35 noise points with missing values for  $f_3$ . The average of the remaining 30 noise points is 129.333. The noise points had relatively more login attempts. All the noise points belong to stealthy single-source password guessing attacks. The small clusters (with cluster points of 2 and 3) might also belong to the stealthy single-source password guessing attacks.

Silhouette coefficient [52] is used to understand how well the clusters are formed. In the data set, the silhouette coefficient is high (0.776), which confirms that data points within a cluster are too close and between clusters are too far.

Moreover, the silhouette coefficient for each of the data points within a cluster is obtained and plotted in the Figure 5.8. Most of the clusters had data points with the silhouette coefficient greater than 0.776. If the silhouette coefficient is low (some of the cluster 19 data points), then the cluster is not well-formed, and the data points belong to a stealthy single-source password guessing attack.

### 5.5.1 Data Model without Feature 3

As discussed earlier, most of the values (83%) of  $f_3$  are not numeric (missing values). Even though the feature is involved in clustering the attacks, for practical reasons the feature 3 is dropped and the model is rebuilt again. In this sub-section, the features:  $f_1$ ,  $f_2$ ,  $f_4$  are used to built the clustering model.

The clustering model discussed earlier (Figure 5.5) is used again. The scaling technique used was min-max normalization (5.2). The weights used in weighted minkowski distance (5.3) are 0.3230, 0.3517, 0.3254 for the features  $f_1$ ,  $f_2$ , and  $f_4$ , respectively. The DBSCAN parameters are  $epsilon = 0.0009507$  and  $minPts = 2$ , determined using the  $k^{th}$  nearest neighbour method. The difference of  $epsilon$

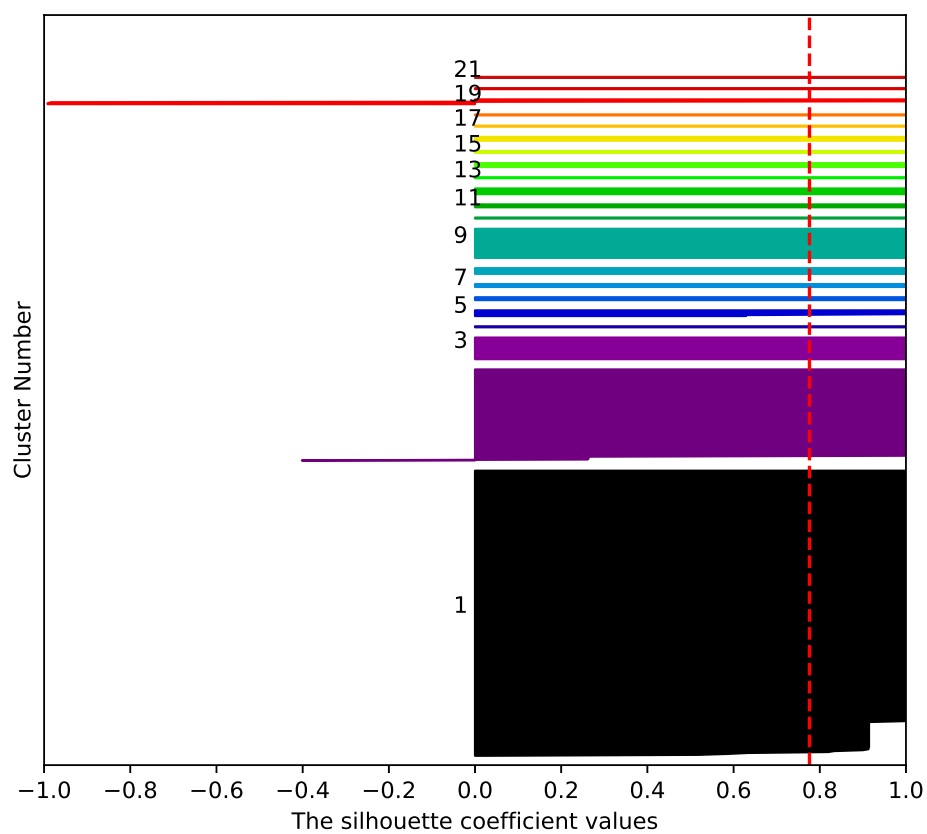


Figure 5.8: Silhouette Coefficients for various clusters

with and without  $f3$  is only 0.001.

The number of clusters is 17. The final silhouette coefficient ( $s = 0.791$ ) is minutely better than the result obtained with the inclusion of  $f3$ . The percentage of data points (considering all the clusters) that had a high silhouette coefficient ( $s_i > 0.8$ ) is 94.58%. The number of noise points is 55.

### 5.5.2 Data Model with Hailmary Dataset

The data set collected in our experiment (Section 3.3) had only 590 data points (stealthy SSH password guessing attacks). Hansteen provided a public dataset [98] containing many data points. Unfortunately, the features that could be extracted from here were only  $f1$  and  $f4$ . The feature  $f2$  was extracted with a minor modification (without the presence of password). A clustering model is build with these ( $f1, f2, f4$ ) features.

In their data set, there were 11,747 data points. The three features had heavy tail distribution (similar to our data set). The correlation matrix revealed the minimum correlation value as -0.45 and the maximum value as +0.14. This shows the features  $f1, f2$ , and  $f4$  are independent. The weights for weighted minkowski distance are  $w1 = 0.3375$ ,  $w2 = 0.3346$ , and  $w3 = 0.3279$ . The *minPts* and *epsilon* found using k-dist method are 2 and  $0.39242067 \times 10^{-3}$  respectively. There were several clusters and outliers. The silhouette coefficient is 0.716.

### 5.5.3 Online Detection

The following methodology (based on online k-clustering approaches) can be used to cluster the attacks in real-time.

1. When a client makes a TCP connection, store the information (in a file)

needed for checking if it belongs to an existing session or not. Also, store information for obtaining the features.

- (a) The time of arrival of the SYN packet and the last packet (FIN packet or RST packet). This is needed for obtaining the session and the feature f4.
  - (b) Check if it is a scanning attempt or a login attempt. This helps finding feature f3.
2. If the TCP connection in step 1. is a login attempt, then store the following details.
- (a) Store SSH Transport Layer protocol details (Client Identification string, Key Exchange Algorithm, Server host key algorithm, Encryption/Decryption Algorithm, Hash Algorithm, Compression Algorithm). These details are needed for verifying the correct session.
  - (b) Store SSH Authentication protocol details (login attempt time(s), user-name(s), password(s)). It is needed for feature 1 and feature 2.
3. If the TCP connection in step 1. is a login attempt, then calculate the features f1, f2, f3, and f4. Now, classify it as a stealthy single-source SSH password guessing attack or stealthy distributed SSH password guessing attack. Store f1, f2, f3, f4, and the label.
4. For the next TCP connection, check the client IP address.
- (a) If the client had already made a TCP connection within the same session, then do steps 1. and 2. Calculate f1, f2, f3, and f4 considering all the past TCP connections. Store f1, f2, f3, f4, and the label in a file.

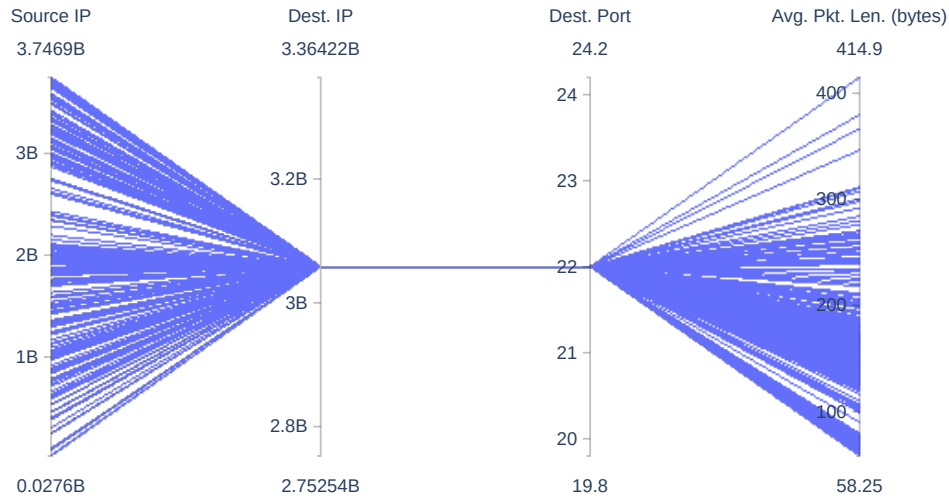


Figure 5.9: Parallel Coordinates Plot from Choi *et al.* work

- (b) If the client has no TCP connection in the same session, then do steps 1., 2., and 3.

The time taken to classify an attack is  $O(1)$ . However, the reliability of the clustering result depends on future login attempts.

#### 5.5.4 Comparison with Existing Works

In Figure 5.9, the parallel coordinate plot (Figure 1 of Choi *et al.* [76]) for our data set (stealthy SSH password guessing attacks) is shown. In this figure, the real source IP address is hidden (to preserve anonymity) by converting it to an integer (represented in the units of billions). Here, “2B” means 2 Billion. “Avg. Pkt. Len.” is the average packet length of all packets in a TCP flow. Every TCP flow in the stealthy SSH password guessing attacks forms a connecting line from

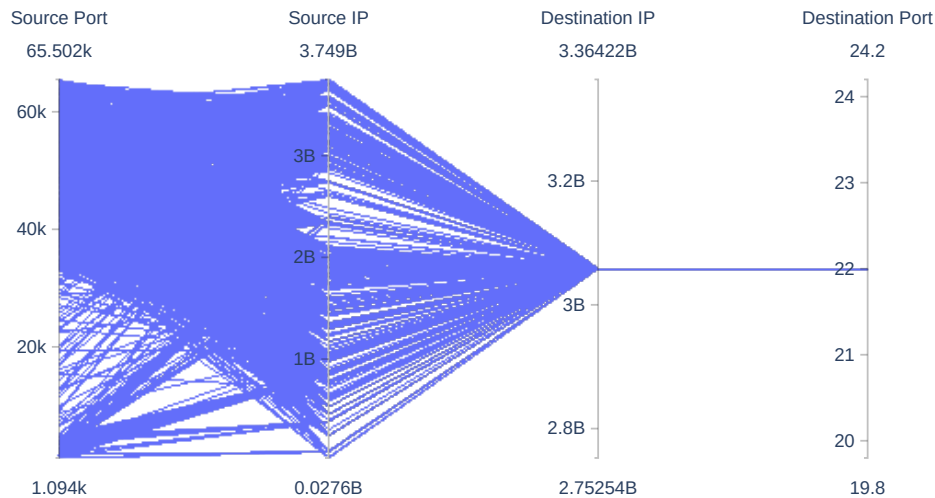


Figure 5.10: Parallel Coordinates Plot from “Conti and Abdullah” work

the “Source IP” axis to the “Avg. Pkt. Len.” axis. There are nearly 1500 TCP flows (our data set) plotted in this figure.

In the case of a stealthy single-source SSH password guessing attack, the “Source IP” axis will have one or more points. In the case of a stealthy distributed SSH password guessing attack, the “Source IP” axis will have several points. For both these attacks, the axes “Dest. IP” (Destination IP) and “Dest. Port” (Destination Port) will remain the same. The destination IP will point to the server IP address, and the destination port is 22 (SSH port number). Consider many different stealthy single-source SSH password guessing attacks on the server. Now, these attacks would create a parallel coordinate plot similar to stealthy distributed SSH password guessing attack. Hence, it is hard to differentiate these two attacks using their parallel coordinate plots.

In Figure 5.10, the parallel coordinate plot (for our data set) used by Conti and Abdullah [73] is shown. Again the “Source IP” axis will have one or more points for stealthy single-source SSH password guessing attacks. The “Source IP” axis will have several points for stealthy distributed SSH password guessing attacks. It is hard to classify these attacks using their plots.

In the work of Pouget and Dacier [74], the first step is to cluster the network attacks based on port sequences. On applying the first step on our data set, the port sequence obtained is “22”. The second step is to cluster the attacks based on Levenshtein distance applied on the application payload of a packet. The application payload can be the same for a stealthy single-source SSH password guessing attacks and stealthy distributed SSH password guessing attacks. Hence, the second step cannot be used to segregate these two stealthy attacks.

The clustering model of Thonnard and Dacier [75] is based on the correlation of SSH traffic captured from multiple honeypots. Our data set is captured using a single honeypot. Their model cannot be tested with our data set. Moreover, their model uses the time signature of the network traffic to infer the origin. The time signature of stealthy single-source SSH password guessing attack and stealthy distributed SSH password guessing attack look similar. Hence, their approach is not capable of differentiating these two stealthy attacks.

The method proposed by Sqalli et al. [77] used entropies to segregate malicious activities found in the honeypot traffic. The entropies were applied on source IP address, destination IP address, source port number, and destination port number. Their intention was to segregate SSH password guessing attacks from other malicious network activities (port scanning, malicious file downloads). The stealthy SSH password guessing attacks (single-source and distributed) create very little

Table 5.4: Summary of Comparisons with Existing Works

Work	Technique Used	Can segregate stealthy SSH password guessing attacks
Conti and Abdullah [73]	Parallel Coordinate Plot	No
Choi et al. [76]	Parallel Coordinate Plot	No
Pouget and Dacier [74]	Clustering and Association Rule Mining	No
Thonnard and Dacier [75]	Correlation of Time Signature	No
Sqalli et al. [77]	Entropy of network traffic	No

change in the entropies, and their changed entropies are below the threshold set by their detection mechanism. Hence, these two stealthy SSH attacks cannot be segregated using their approach.

## 5.6 Conclusion

The model can be applied to any network protocol that supports password authentication. Some of the network protocols are FTP, MS-SQL, MySQL, Telnet. All four features can be obtained for the above protocols.

The above discussion was on password-based attacks. In the future, if there



is any other guessing attack that involves the user to remember any information (passphrase, PIN, a set of images), the model can be applied. Even public-private key-based authentication can be employed, and the model can differentiate the two types of stealthy attacks.

There are duplicates in the data set. The duplicates refer to different stealthy attacks from same/different sources. These duplicates should not be removed because they are distinct attacks originating from same/different sources. Duplicates could also refer to network traffic from several bots forming the same botnet. DBSCAN can efficiently group the duplicates in a single cluster without hindering the formation of other clusters.

Assume that a single-source attacker tries only one login attempt on all the chosen SSH servers (say “n” servers). In this scenario, in each of the server log entries, there would be only one entry referring to the failed login attempt by a specific IP address. In this use case, the model erroneously classifies it as a distributed attacker. This happens because the single-source attacker mimicked the behaviour of a distributed attacker.

A botnet can do stealthy coordinated password guessing and stealthy uncoordinated password guessing. In the future, it would be necessary to understand better and detect these two types of attacks.

Besides, Zhu and Zheng [99] used control theory (modelling non-linear systems using Lyapunov function) to detect periodic Denial-of-Service attacks on Cyber-Physical systems. In the future, authors would like to explore more on control theory methods to segregate attacks.

# Chapter 6

## Detection of SSH Compromises

### 6.1 Introduction

In the previous chapters (in Chapter 4 and Chapter 5), the detection of stealthy SSH password guessing attacks in two different contexts were explored. In this chapter, the problem of detecting SSH compromises in a honeynet network traffic is investigated.

An SSH server is compromised when the malicious entity successfully determines the login credentials. In an SSH network traffic, both the login attempts and the commands executed after the successful connection, are encrypted. Given an SSH session's network traffic (pcap files), it is hard to tell whether the attacker has executed some commands or not (after successful login).

An SSH session may have several encrypted packets (SSH Authentication protocol packets or SSH Connection protocol packets or both). The number of encrypted packets in the SSH Authentication protocol varies. The number of encrypted packets in the SSH Authentication protocol depends on the maximum number of login attempts ( $M$ ) allowed by the client. The value of  $M$  is config-

ured in the SSH server. The number of encrypted packets in the SSH connection protocol depends on the user (i.e., the number of commands entered by the user). Hence, taking an SSH TCP flow with encrypted packets, it is hard to tell the existence of SSH connection protocol packets (except a few use cases where the number of total encrypted packets is minimal).

An SSH flow pcap file consists of two protocols (Authentication protocol, Connection protocol) that are encrypted. So, the objective is to classify SSH compromises from SSH password guessing attacks, given a TCP flow pcap file. In the literature review (Section 2.6), two works [80, 81] have proposed methods to detect SSH compromises by analysing the network traffic. Their approach with limitations was discussed in Chapter 2.

The following sections describe our method to detect SSH compromises in honeynet traffic. Section 6.2 describes the categorization of the SSH network traffic and the tagging of the TCP flow pcap files. Section 6.3 discusses the machine learning features, the choice of machine learning algorithm used, and the performance analysis. Section 6.4 explains the machine learning model. In Section 6.5, a real-time methodology to detect SSH compromises is proposed. Finally, the conclusion is given in Section 6.6.

## 6.2 Data Pre-processing

### 6.2.1 Network Traffic

As described in Section 3.3, the network traffic to and from Kippo SSH server was collected in pcap format. The original pcap files were split using Splitcap tool [100] to create individual TCP flow pcap files. The original pcap files had network

traffic from many different client machines. The processed TCP flow pcap file has network traffic only from a particular client and comprises of packets belonging to one particular session. If the same client makes several connections, then each would be in a separate TCP flow pcap file. Each individual TCP flow is identified by a unique set of client IP address, client port number, server IP address, and server port number. For all the TCP flow pcap files, the server IP address was constant and pointed to the public IP address of the DELL server. Also, the server port number was the same (port number 22) for all the TCP flow pcap files.

All TCP flow pcap files had SSH Transport layer protocol. Pcap files with SSH password guessing attacks had SSH Authentication protocol. Pcap files with SSH compromises had SSH Connection protocol, in addition to SSH Authentication protocol. Each one of the TCP flow pcap files is individually labelled.

### **6.2.2 Class Labels**

On the convenience of describing the attacks, the author created two different labels: severe attacks and non-so-severe attacks.

Severe attacks comprise all SSH compromises that involves the execution of at least one UNIX command on the remote terminal of the SSH server. The login is successful and the attacker decided to run some commands. The connection could have started with a password guessing attack (a few failed login attempts) followed by a successful login attempt, then execution of commands to exploit the system. The attacker could also directly login to the system without a password guessing attack.

Not-so-severe attacks include all SSH attacks that do not include an SSH Connection protocol packets. These attacks are described below.

1. Port scanning on SSH port (port 22) that includes only a couple of network or transport layer packets.
2. Password guessing attack on the SSH port. The entire TCP flow has only login failure messages (i.e., no successful login status packets).
3. Password guessing attack that was successful. After the successful login attempt, the connection is closed by the attacker. These attacks do not have any UNIX commands.

As described, several successful login connections had no UNIX commands. The hacker is only interested in knowing the login credentials. These extracted successful login credentials would be traded in the underground market (i.e., black-hat community without public presence). Also after a successful connection, it is possible for the hacker to carry password guessing attacks on other usernames.

The class label “not-so-severe” was chosen because these attacks will not cause any damage to the Operating System and the file contents of the hard disk. These attacks superficially affect the network bandwidth, but there is no harm to the hardware and the software of the system.

The class label “severe” was chosen because the attack involves the attacker’s ability to execute some commands in the system. The attacker can carry out any malicious activity on the server. Depending on the level of access provided by the user, the activities can change the state of the Operating System or change/delete the contents in the hard disk. Hence, SSH compromises were named as “severe” activities in this work.

Table 6.1: Traffic Classes

Severity Level	Traffic Type	Number of TCP flows
Severe Attacks	SSH Compromise with one or more Unix commands	14
Not-So- Severe Attacks	SSH Compromise with no Unix commands	58
	SSH Password Guessing At- tacks	10308
	SSH Port Scanning	552

### 6.2.3 Tagging the Pcap Files

The TCP flow pcap files were tagged with the class labels. Two approaches were followed to tag the labels.

1. Java programs were developed that would automatically tag scanning TCP flows.
2. Manually each TCP flow pcap file was cross-verified with the Kippo log files and the appropriate class label was assigned.

The final count of the number of TCP flows with each class label assigned is provided in Table 6.1. The number of “not-so-severe” attacks are considerably high compared to the number of “severe attacks”. This represents a class-imbalance problem.

In the data set, only two different types of port scanning were observed.

1. SYN scanning
2. TCP Connect scanning

## 6.3 Design Choices for the Model

### 6.3.1 Machine Learning Features

The features for the machine learning model was obtained through domain knowledge and literature survey. The statistical features were applied to each TCP flow that corresponds to a single SSH application session. The list of features are

1. Total number of packets
2. Total number of received packets
3. Total number of sent packets
4. The sum of all packet bytes
5. The sum of all received packet bytes
6. The sum of all sent packet bytes
7. The sum of all payload bytes
8. The sum of all received payload bytes
9. The sum of all sent payload bytes
10. Mean inter-arrival time between received packets
11. Variance of inter-arrival time between received packets
12. Total number of packets with ACK flag set
13. Total number of packets with PSH flag set
14. Total number of packets with RST flag set

In the above features, “received packets” refer to packets received by the SSH server and “sent packets” are the packets sent by the SSH server.

Each feature was obtained for an individual TCP flow. The features were obtained using a Java program developed using the jnetpcap library [101] and a math library [102]. jnetpcap library helped in reading the packets and counting them. Math library was used to calculate mean inter-arrival time and the variance of inter-arrival time. All of the above features are numerical attributes.

### 6.3.2 Choice of Machine Learning Algorithm

The following set of Machine Learning algorithms were chosen to check the best performance for the above problem.

1. ZeroR
2. OneR
3. PART
4. Decision Tree (J48)
5. Naive Bayes (NB) algorithm
6. Logistic Regression (LR)
7. Support Vector Machine (SVM)
8. k-Nearest Neighbour (kNN)

ZeroR [103] is the baseline classifier provided in the weka suite. The performance of other machine learning algorithms is compared with the baseline classifier. If the performance of a machine learning algorithm is better than ZeroR’s



performance, then it is considered a good candidate for classifying “severe” and “non-so-severe” attacks.

The above list of machine learning algorithms is taken based on the mathematical foundation on which it is built. OneR and PART are rule-based classifiers, whereas, the decision tree is built on a set of rules that form a binary tree. Naive Bayes algorithm is based on probability theory and Logistic Regression is derived from the mathematical function, called the sigmoid function. Support Vector Machine is obtained from optimizing an objective function, that increases the marginal distance between two different classes. k-Nearest Neighbour is based on a voting scheme. Moreover, Logistic Regression, SVM, and k-Nearest Neighbour (k-NN) perform well for a data set with only numeric attributes.

The input parameters for each machine learning algorithm is obtained in an exhaustive manner (i.e., each and every numerical value is entered in sequence). The best set of parameters for each machine learning algorithm is given below.

### **OneR**

1. The numeric attributes are discretized into several intervals.
2. The minimum number of samples in a bucket is 6.

With OneR, the results were not changed when the minimum number of samples in a bucket is set to 3 or 4 or 5 or 6.

### **Decision Tree (J48)**

Both unpruned decision tree and pruned decision tree were built.

1. The attribute evaluation is done using “Information Gain Ratio” [104].

2. For pruned tree, the confidence threshold is 0.25.

## PART

1. The confidence threshold for pruning is 0.25.
2. The minimum number of instances per rule is 2.

## Naive Bayes Algorithm

All features portrayed Gaussian distribution. Hence, Gaussian distribution was used to find the probability of a given class.

## Logistic Regression

Equation 6.1 was used to evaluate the data set in Logistic Regression.

$$L = - \sum_{i=1}^n (Y_i \log P_{SA}(X_i) + (1 - Y_i) \log P_{NSA}(X_i)) + r\theta^2 \quad (6.1)$$

In Equation 6.1,  $L$  is the cost function,  $Y_i$  is the output class in binary values (0,1), and  $X_i$  is a data sample.  $P_{SA}$  and  $P_{NSA}$  stand for the probability of a “severe” and that of a “not-so-severe” attack, respectively.  $\theta$  is the vector of Logistic Regression co-efficients, and  $r$  is the ridge parameter. The ridge parameter used is  $1 \times 10^{-8}$ . Quasi-Newton method was used to find the optimal values of  $\theta$ .

## Support Vector Machines

$$k'(X_i, X_j) = [X_i \cdot X_j]^d = [X_i \cdot X_j]^{24} \quad (6.2)$$

In Equation 6.2,  $k'$  is the kernel function and  $d$  is the degree of the polynomial. The degree of the polynomial is 24.

$$k(X_i, X_j) = \frac{k'(X_i, X_j)}{\sqrt{k'(X_i, X_i) \cdot k'(X_j, X_j)}} \quad (6.3)$$

In Equation 6.3,  $k$  is the normalised kernel function.

Sequential Minimal Optimization (SMO) [105] computes the co-efficients of the SVM algorithm.

1. The value of the slack variable  $\epsilon_i$  is  $1 \times 10^{-12}$ .
2. The penalty constant  $C$  is 1.

### **k-Nearest Neighbour**

1. The similarity measure is Euclidean distance.
2. The number of nearest neighbours is 3.

For the k-NN algorithm, the addition of weights to the voting method did not change the results.

In all the above algorithms (except k-NN), 10-fold cross-validation with stratification is used to create the training set and the testing set. In each fold, the data set is chosen randomly. Before application of each fold, the data set is randomized again.

### **6.3.3 Performance Analysis**

In this section, each of the chosen machine learning algorithms is tested with the data set to check its performance. The parameters were already chosen in the previous section. The confusion matrix is modified as in Table 6.2.

Table 6.2: Confusion Matrix

	<b>Predicted: Severe Attack</b>	<b>Predicted: Not-So-Severe Attack</b>
<b>Actual: Severe Attack</b>	TS	FNS
<b>Actual: Not-So-Severe Attack</b>	FS	TNS

In Table 6.2, TS, FS, TNS, and FNS represents True Severe Attack, False Severe Attack, True Not-So-Severe Attack, and False Not-So-Severe Attack, respectively.

The confusion matrix for all the previously chosen algorithms is shown in tabular form in Table 6.3.

The following performance metrics were used to best analyze the classification and the classifier.

Table 6.3: Confusion Matrix for ML algorithms

Algorithm	TS	FS	TNS	FNS
ZeroR	0	0	10918	14
SVM	8	3	10915	6
kNN	8	1	10917	6
OneR	9	1	10917	5
LR	10	4	10914	4
NB	12	5	10913	2
PART	13	0	10918	1
J48	13	0	10918	1

$$Accuracy = \frac{TS + TNS}{TS + TNS + FS + FNS} \quad (6.4)$$

$$Sensitivity = \frac{TS}{TS + FNS} \quad (6.5)$$

$$Precision = \frac{TS}{TS + FS} \quad (6.6)$$

$$F - score = \frac{(1 + \beta^2)(prec. \times sens.)}{(\beta^2 prec.) + sens.} \quad (6.7)$$

Since the data set is unbalanced, “Accuracy” is not the best metric to decide on the goodness of the classification. Hence, “Sensitivity” was chosen to know how well the severe attacks were classified. “Precision” was taken to know the effectiveness of the classifier. “F-score” gave a balanced score between “Sensitivity” and “Precision”.

Table 6.4: Performance Evaluation

<b>Algorithm</b>	<b>Sensitivity</b>	<b>Precision</b>	<b><math>F_2</math> score</b>
SVM	0.5714	0.7273	0.5970
kNN	0.5714	0.8889	0.6154
OneR	0.6429	0.9000	0.6818
LR	0.7143	0.7143	0.7143
NB	0.8571	0.7059	0.8219
PART	0.9286	1.0000	0.9420
J48	0.9286	1.0000	0.9420

Table 6.4 provides the performance metrics (Sensitivity, Precision,  $F_2$ -score) for the seven machine learning algorithms. The accuracy of ZeroR is 0.9987. All algorithms had accuracy greater than ZeroR (greater than 0.9990). The sensitivity of ZeroR is 0. ZeroR had no precision and  $F_2$ -score.

Both J48 and PART had the highest and the same performance metric for the data set. Naive Bayes Algorithm is comparatively better than the Support Vector Machine and k-NN algorithm, which were able to detect only 57.14% of all the severe attacks.

The classifiers J48 and PART made 100% correct predictions (precision score) on the severe attacks. The results of k-NN and OneR are comparatively better than Support Vector Machines, Logistic Regression, and Naive Bayes algorithm. The precision of Support Vector Machines, Logistic Regression, and the Naive Bayes algorithm are very similar. Support Vector Machine, k-Nearest Neighbour, and One R had low sensitivity but a high precision, whereas, Naive Bayes algorithm

had high sensitivity but slightly lower precision.

F-score gives a value combining sensitivity and precision. In this work, the value of  $\beta$  is taken as 2. Hence, sensitivity is given more importance than precision. This is done to give more importance to classification rather than the performance of a classifier. Since, J48 and PART had the highest sensitivity and precision, the  $F_2$  score was also highest. The  $F_2$  score of Support Vector Machine, k-Nearest Neighbour, and OneR are slightly higher than its sensitivity. Logistic Regression's  $F_2$  score was the same as its sensitivity and precision. The  $F_2$  score of Naive Bayes algorithm is slightly lower than its sensitivity.

In comparison with all algorithms, J48 and PART did considerably well. J48 chooses the best features with the help of information gain quantity in each level of a decision tree. It is similar to applying feature selection technique in each level. Hence, the intuition was to apply feature selection to find the best set of features.

### 6.3.4 Feature Selection

Exhaustive search is the technique used to find the best set of features from the original set. It is not an heuristics-based approach and guaranteed to produce the correct results at all times. The dimension of the data set was small, and the feature selection took considerably less time to do the computation. The performance evaluation algorithm used in the exhaustive search is the Naive Bayes algorithm.

Figure 6.1 shows the performance metrics (sensitivity, precision,  $F_2$ -score) for each row of feature sets computed in exhaustive search. The last element in the x-axis of Figure 6.1 gives the best feature set with high values for all the different performance measures.

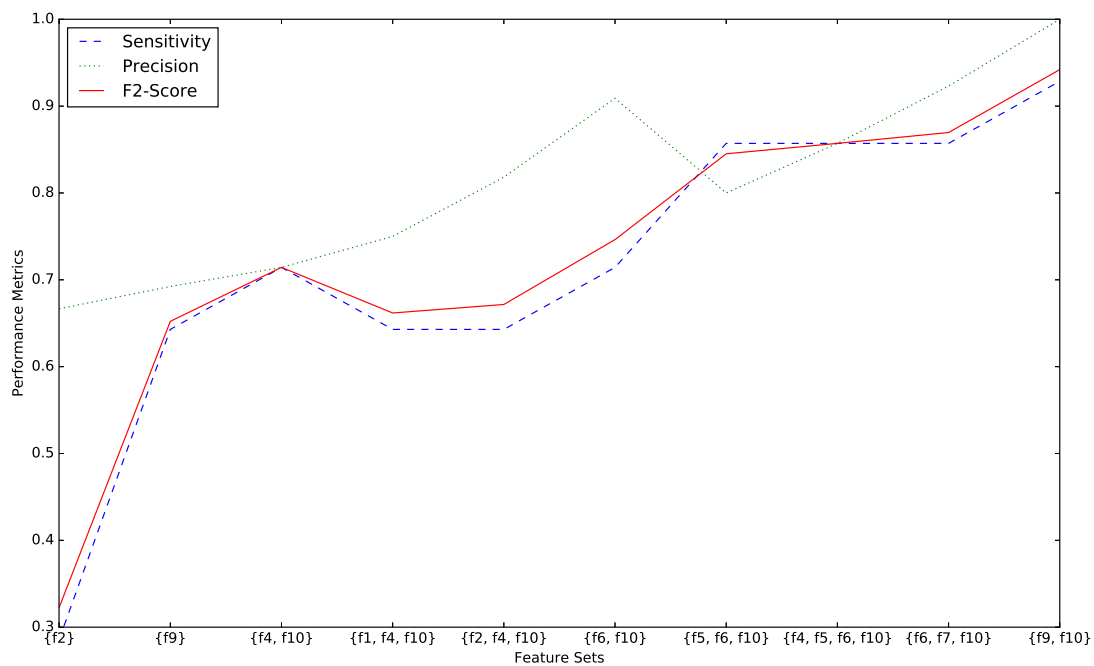


Figure 6.1: Best Feature Set Performance

Table 6.5: Performance with Best Features

Algorithm	Sensitivity	Precision	$F_2$ score
SVM	1.0000	0.5600	0.8642
3-NN	0.9286	0.8667	0.9155
OneR	0.6429	0.9000	0.6818
LR	0.9286	0.8667	0.9155



The algorithms that worked poorly in Table 6.4 are re-run with the features f9 and f10. Table 6.5 shows the performance of the algorithms Support Vector Machine, 3-Nearest Neighbour, OneR, and Logistic Regression. All three performance measures improved significantly for Logistic Regression and 3-Nearest Neighbour. There was no change in the performance of OneR algorithm. The precision of the SVM algorithm slightly decreased because of the misclassification of 14 different not-so-severe attacks. Otherwise, SVM classified all the severe attacks correctly leading to a sensitivity of 100%.

It can be inferred from this analysis that J48 performs well for this problem. Hence, the machine learning model employed J48 to do the classification.

## 6.4 Machine Learning Model

The machine learning model to segregate severe attacks from not-so-severe attacks is shown in Figure 6.2. The first two blocks (“Honeypot Captured pcap files” and “Pre-process to remove unwanted traffic”) were described in Chapter 3. The third and fourth block (“Split the pcap file into individual TCP flows”, “Extract features from each TCP flow”) were explained in the Section 6.2 and the Section 6.3. The next block “Normalize the data set” is briefed in the following sub-section.

### 6.4.1 Normalize the Data Set

Each of the different features was in a different range. There are a few chosen machine learning algorithms that are sensitive to different range of the input features. For example, “Logistic Regression”, “Support Vector Machine”, and “k-NN” perform differently with and without normalization [106]. These algorithms perform

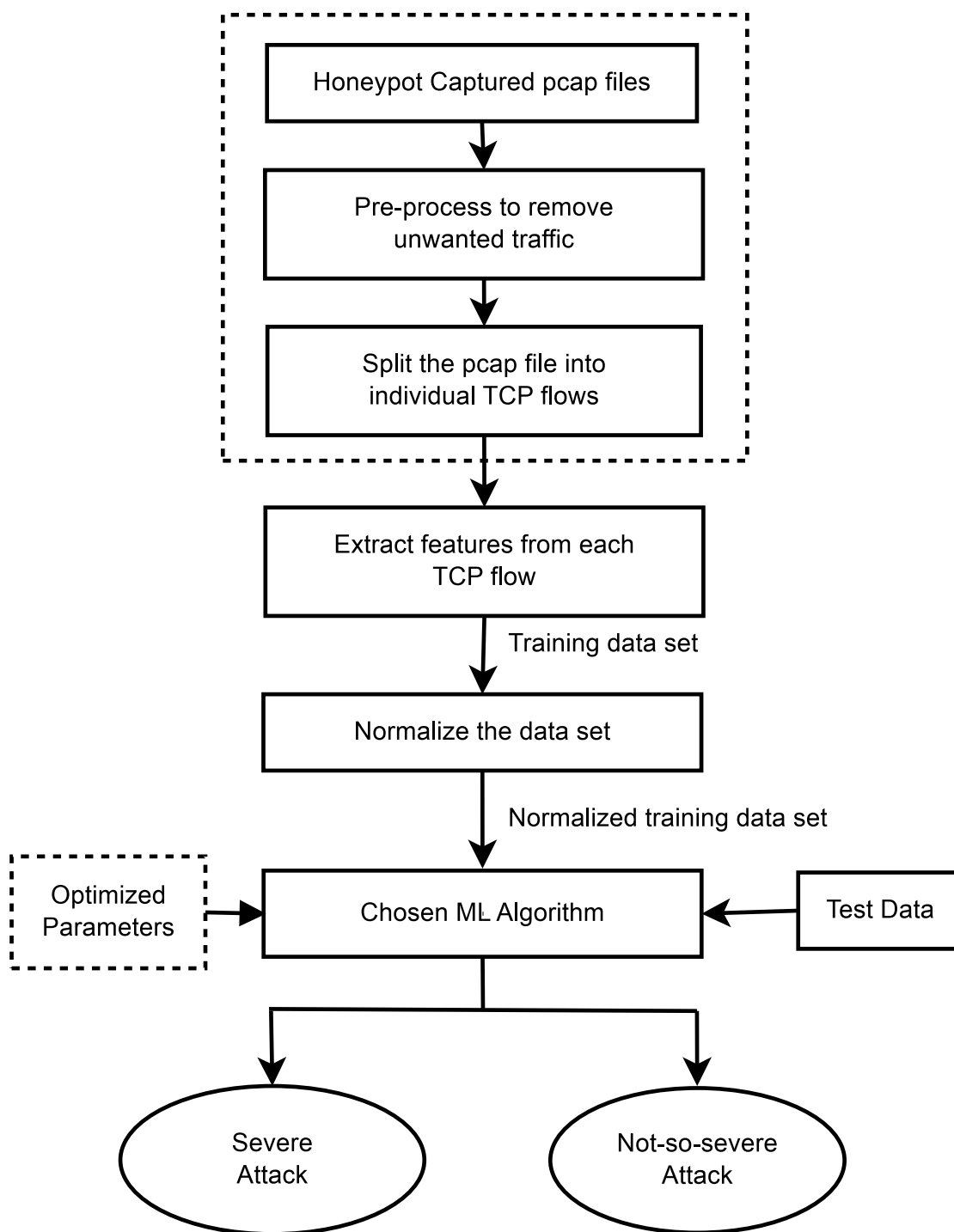


Figure 6.2: Machine Learning Model

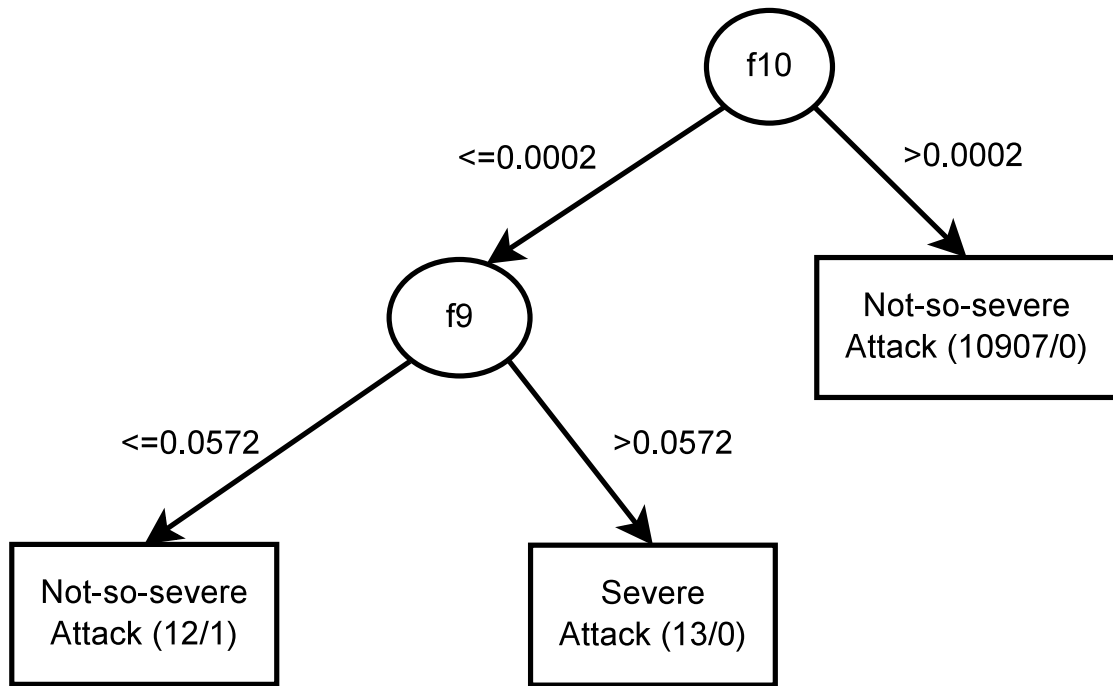


Figure 6.3: J48 Decision Tree

much better on normalized data. In order to preserve the structure of the data, “min-max normalization” was chosen to scale the data.

### 6.4.2 Chosen Machine Learning Algorithm

The selected machine learning algorithm for this work is the J48 decision tree algorithm. The optimized parameters were obtained from the experiment conducted in Section 6.3.

In the J48 algorithm, both the pruned and unpruned version produced the same results. The pruned decision tree obtained using the J48 algorithm is shown in Figure 6.3. The feature f10 is the mean inter-arrival time between the received packets of the server. The feature f9 is the sum of all sent SSH application payload bytes by the server.

Feature f10 did justice in separating most of the not-so-severe attacks (10907 attacks). Feature f9 segregated most of the severe attacks (13 attacks). If the mean inter-arrival time of received packets (Feature f10) is greater than 1651 milliseconds, then it is most likely a not-so-severe attack. If the quantity of sent payload bytes (Feature f9) is more than 3216 bytes, then it is highly likely a severe attack.

## 6.5 Real-Time Detection of Severe Attacks

The purpose of this section is to provide a mechanism to detect severe malicious activities in real-time.

The flowchart for detecting SSH severe attacks in real-time is shown in Figure 6.4, Figure 6.5 and Figure 6.6. The flowchart was implemented in Java language with the libraries: jnetpcap [101], commons-math3-3.5 [102]. The Java application is multi-threaded using shared resources among the threads. The shared resources are locked and unlocked correctly using synchronized method calls.

### 6.5.1 Network Packet Capture

The first thread is involved in the capturing of the network packets. The library jnetpcap had the necessary methods to capture packets from a live network. In order to expedite the processing of the packets, every time a packet is received by this thread, it is inserted into a queue. The capacity of the queue is 100,000 packets.

### 6.5.2 Segregation of Packets

The following steps are followed by the second thread for processing the packets.

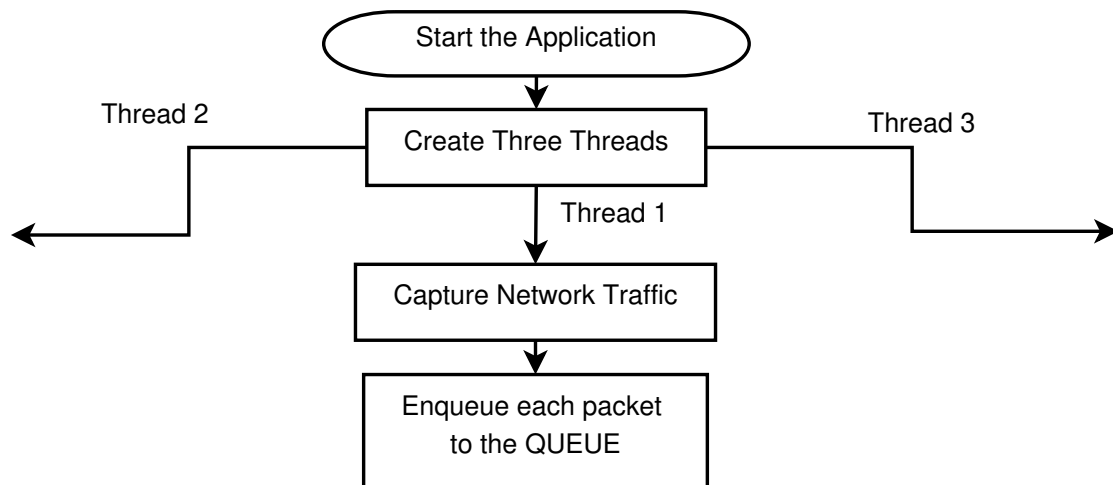


Figure 6.4: Flowchart for Real-Time Detection of SSH Severe Attacks - Thread 1

1. A packet is dequeued, and checked if it contains both TCP header and IPv4 header. If the TCP header is missing or IPv4 header is missing, then the packet is discarded.
2. If the packet has SYN flag set (the first packet of a TCP connection), then a new flow file (i.e., an empty pcap file) is created. The filename is the concatenation of source IP address, destination IP address, source port number, and destination port number of the packet. This filename is used to identify further packets from the same TCP flow. The packet is stored in the pcap file.
3. If a packet arrives without RST flag and FIN flag set, then the packet is immediately stored in the correct pcap file.
4. If a packet has RST flag or FIN flag set, then the corresponding variable (i.e., two boolean variables for each TCP flow) associated with the flow filename is set. These variables are tied to the file and stored in a separate location.

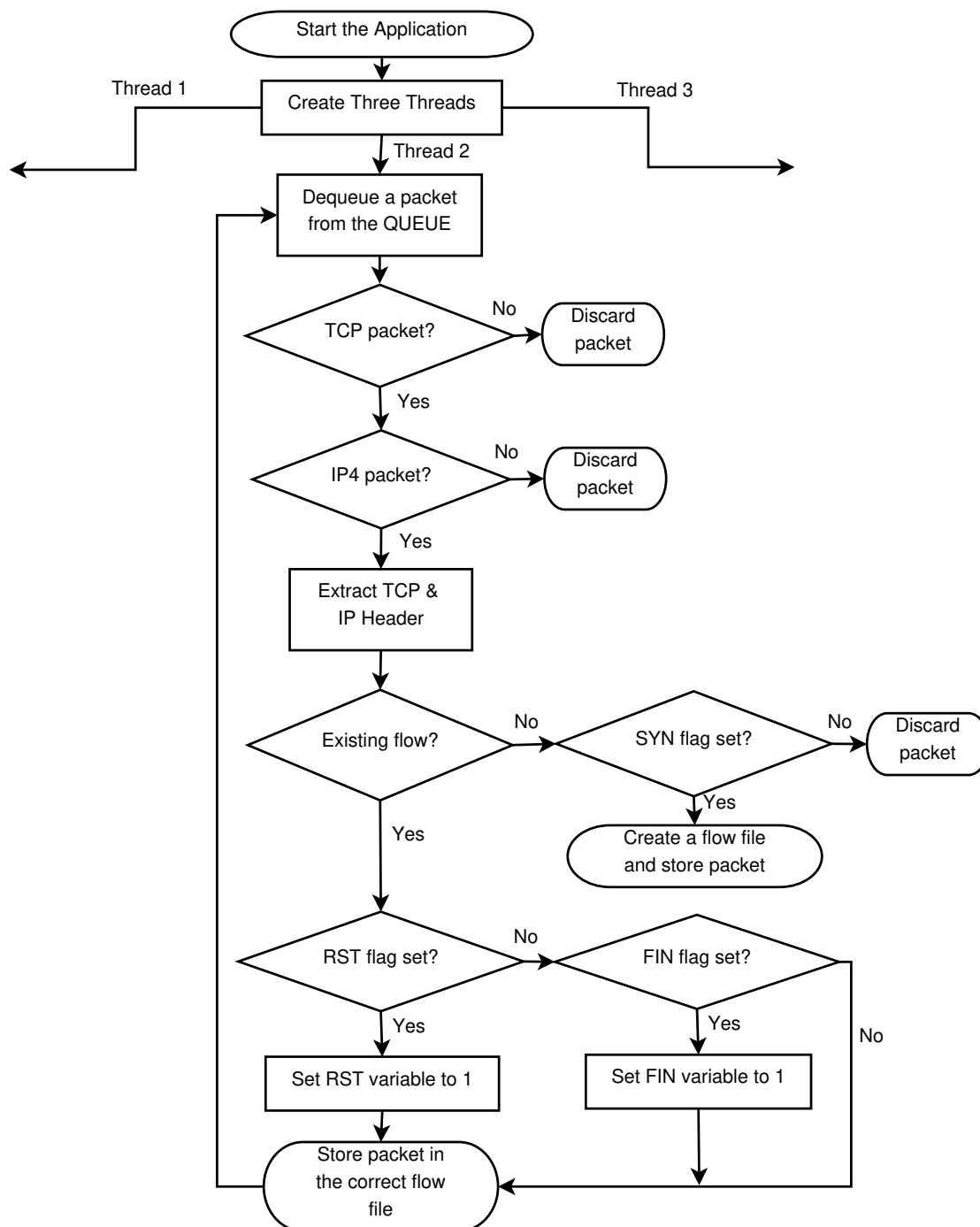


Figure 6.5: Flowchart for Real-Time Detection of SSH Severe Attacks - Thread 2

Every TCP flow allocates space for these two variables. Finally, the packet is stored in the correct pcap file.

### 6.5.3 Creation of TCP Flows

The third thread reads through all the TCP flows and finds the ones that are completed. Those completed flows proceed to final classification. Three different conditions are checked before closing a TCP flow. These conditions are

1. The appearance of an RST packet for a particular TCP flow indicates that the flow is immediately terminated by the client. Hence, this flow is completed and makes room for classification.
2. In Operating Systems, a KeepAlive timer is used to prevent long idle TCP connections. If a TCP connection is idle beyond the timeout (usually 120 minutes), then the connection is closed. For the testing purpose, the timeout is set to 10 minutes. If the last packet in a TCP flow is older than KeepAlive time (i.e., 10 minutes), then the TCP flow is considered completed or closed, and ready for final classification.
3. The machine (client or server) that performs “Active Close” (TCP Connection State) waits to receive FIN packet from the other side. Once a FIN message is received, it immediately sends an ACK and waits for 2MSL (Maximum Segment Lifetime) before closing the connection. So, after receiving two FIN packets in the TCP flow, the connection is waited for 2MSL (set to 60 seconds) before terminating the connection.

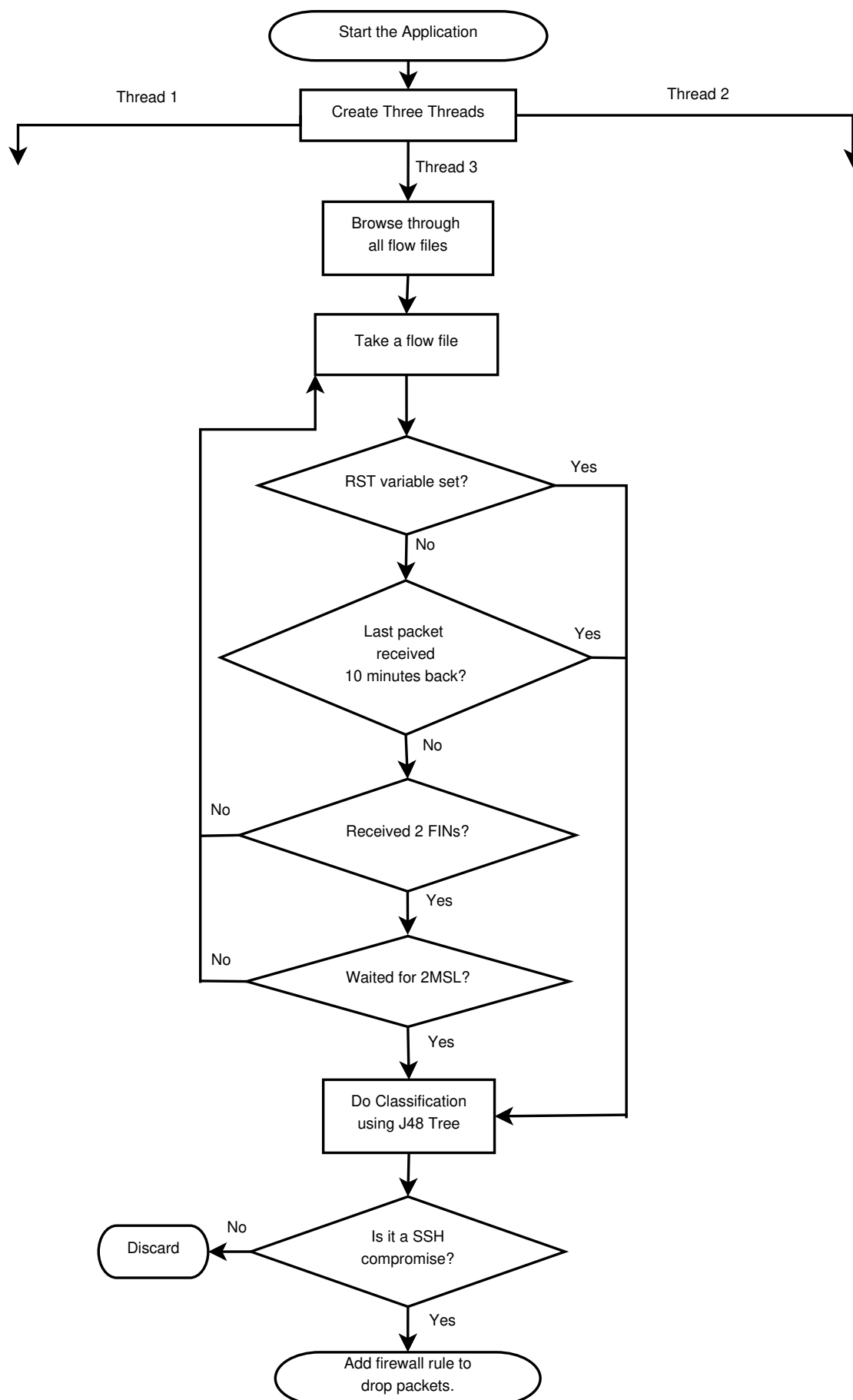


Figure 6.6: Flowchart for Real-Time Detection of SSH Severe Attacks - Thread 3



#### 6.5.4 Classification

The classification is done using the J48 decision tree algorithm as shown in Figure 6.3. If the result of the classification is a severe attack (SSH compromise), then a firewall rule is added to iptables, to block further attacks. The alerting mechanism can be changed to sending a message to the administrator when there are several such compromises. If there are several compromises, then the server password can also be changed.

#### 6.5.5 Performance of the Classifier

The time taken to build the model using J48 is 0.26 seconds. The total time to classify a TCP flow is based on two factors:

1. Time for completing a TCP flow. Assume  $n$  as the average number of TCP packets in a flow. Then the total duration of a flow is determined by the value of  $n$ . The Big-Oh notation is  $O(n)$ .
2. Time to classify an attack is negligible and can be discarded. The Big-Oh notation is  $O(1)$ .

Hence, the overall time to detect an attack would be  $O(n)$ .

#### 6.5.6 Limitations

1. The algorithm does not do the real-time classification of the attacks. The attack has to get completed (the TCP flow has to end) before the classification step.
2. The implementation had a limitation. Two TCP flows cannot have the same source IP address and the same source port number. (It is a valid assumption

rather than a limitation. The probability of an attack originating from the same source machine with a different source port number is high.)

## 6.6 Conclusion

The problem of detecting SSH compromises is discussed in this chapter. Conventionally, HIDS detects these compromises using a simple algorithm working on log files. Unfortunately, it is not scalable. The approach by Satoh et al. [80] uses Ward clustering to train their model (to detect SSH compromises). Hofstede [81] uses a simple algorithm (without machine learning) to detect SSH compromises. We used conventional machine learning (decision tree) algorithm, with a data set having complete TCP flows, to build the model.

The previous section explains a dedicated real-time algorithm, with advantages and disadvantages. Satoh et al. [80] do not discuss real-time detection of SSH compromises. Hofstede [81] uses Cisco’s NetFlow information (obtained from the routers) as the input to their algorithm. Our mechanism (and Satoh et al. [80] method) use TCP flow details extracted from regular network traffic. Both (our model and Hofstede et al. [81] method) discuss the closure of a flow after a successful login attempt. The flow data (used by Hofstede et al. [81]) consumes less memory space compared to network traffic (packet-based). However, there are more practical difficulties in detecting SSH compromises (Hofstede et al. [107]).

In the presence of SSH compromises, the TCP flow has comparatively more payload bytes (i.e., the sum of all sent payload bytes) and less mean inter-arrival time. In 2-dimensional space, several “not-so-severe” attacks aggregated together, which made the J48 algorithm to segregate them easily.

The data set is obtained from a honeynet. The SSH traffic can be obtained

from both legitimate and malicious entities. Under this scenario, it would be more challenging to detect SSH compromises. The author would like to work on this challenging task.

Dumont et al. [108] used n-gram features in machine learning algorithms to detect malicious SSH commands. The log files store the SSH commands executed by a user (legitimate/malicious). Using, n-gram their model segregates malicious commands from legitimate commands. In our model, the data set is obtained from honeypot traffic (only malicious commands). In future, the author would propose new techniques to improve the accuracy of Dumont et al. method.

# Chapter 7

## Conclusions and Future Scope of Work

This thesis proposed machine learning mechanisms that detect malicious SSH activities (password guessing attacks, compromises) in honeypot traffic. The past research on detecting SSH password guessing attacks and detecting SSH compromises was explained in Chapter 2. The honeynet architecture to collect malicious activities was explained in Chapter 3. An approach to classify different stealthy SSH password guessing attacks (based on the source) was explained in Chapter 4. The method to identify stealthy single-source SSH password guessing attacks in honeypot traffic was detailed in Chapter 5. Finally, the detection of SSH compromises in honeypot network traffic was presented with results (Chapter 6).

## 7.1 Conclusions and Summary of Research Contributions

In Chapter 4, the different instances of stealthy SSH password guessing are segregated based on the attack origin (source of the attack). The approach is simplistic and most of the information is obtained from honeypot log files. Proper reasoning was provided for the model parameters. In the field of network forensics, it had always been a challenge to identify the attackers, by scrutinizing the log files. The merging technique would help in identifying the group of attackers (or single attacker) involved in stealthy SSH password guessing attacks.

The main contribution of Chapter 4 is the identification of the fields (SSH client version, SSH outgoing encryption algorithm, SSH outgoing hash algorithm, SSH outgoing compression algorithm, Operating System, Usernames, Passwords) that help in segregating the password guessing instances.

Chapter 4 also does traffic analysis on stealthy SSH password guessing attacks. Analysis was done on the login credentials, SSH client library, and the login attempts in each TCP flow. The analysis helps in building the model to segregate stealthy SSH password guessing attack instances.

The nature of the attackers is also scrutinized in a different method in Chapter 4. The characteristics of source IP address and source port number provide the nature of the attackers. Most of the IP addresses belonged to China. The block IP addresses were analysed using classless addressing scheme. In all the data sets, only a few source port numbers appeared frequently. It conveys that the source port number is hard-coded in the attacker's source code (or executable).

In Chapter 5, the two different types of stealthy SSH password guessing at-

tacks were examined in detail. These two types of stealthy attacks appear very similar and had been a challenge to segregate them. We identified features that differentiate these two stealthy attacks: number of login attempts, repetition of login attempts, duration between port scanning and the password guessing attack, the aggressive nature of password guessing by a single-source. These features help in building models (with or without machine learning) that detects malicious activities originating because of stealthy attacks.

We used a clustering approach to segregate the attacks. Each and every parameter in the algorithm was finely tuned based on theoretical reasoning and empirical validation. The reason behind the use of a clustering approach as the model is also explained. All the stealthy SSH password guessing by bots (of a single botnet instance) formed a cluster. The outliers of the clustering result were the stealthy single-source SSH password guessing attacks. Among the different clustering algorithms, DBSCAN was selected based on proper reasoning.

The model can be applied for any authentication mechanism that involves the user to remember information (like PIN, a set of images, passphrase, and so on). All the features described are applicable to the above authentication mechanisms. The proposed technique can also be applied for any network protocol (Telnet, FTP, HTTP, MySQL, MS-SQL) that uses password-authentication. All these network protocols provide the necessary features to classify the two different stealthy password guessing attacks.

The clustering model was also applied on Hansteen's data set [98] which had only the features  $f1$  and  $f4$ . Clusters and outliers were also formed using this data set. The clusters represented stealthy distributed SSH password guessing attacks and the outliers represented stealthy single-source SSH password guessing attacks.

Detecting stealthy SSH password guessing in real-time network traffic (containing both legitimate and malicious traffic) has been a challenge. Some approaches were presented which had a few limitations and cannot be implemented practically. The work done in Chapter 5 will pave the way for better security approaches to detect stealthy SSH password guessing attacks. On a general view, the approaches can be designed to detect any stealthy password guessing attack (using FTP protocol, Telnet protocol, and so on).

Chapter 6 explores the problem of detecting SSH compromises in the network traffic. Initially, suitable machine learning algorithms (ZeroR, OneR, PART, Decision Tree, Naive Bayes algorithm, Logistic Regression, Support Vector Machine, k-Nearest Neighbour) were tested with different parameters using the data set. Decision Tree performed well with and without feature selection. Finally, J48 algorithm was used to detect the SSH compromises. The overall time to build the model was quite less (0.26 seconds).

The significant features (f9: the sum of all sent payload bytes, f10: mean inter-arrival time between received packets) help differentiate SSH password guessing attacks from SSH compromises. These features were obtained through feature selection mechanism. Exhaustive search is the feature selection method and the evaluation algorithm was Naive Bayes algorithm.

The features provide the nature of SSH compromises. SSH compromises have less mean inter-arrival time between the received packets by the server. It happens because of the hacker's activity in the SSH connection protocol. The sent payload bytes in the SSH application layer content is also high because of the packets in the SSH connection protocol. These insights help in building security tools that detect SSH compromises in the presence of real-time traffic (both legitimate traffic

and malicious traffic).

Four different performance metrics: accuracy, sensitivity, precision,  $F_2$ -score, played an important role in evaluating the data set.  $F_2$ -score was chosen to give more preference to sensitivity compared to precision. The data set was randomly shuffled during each iteration of the 10-fold cross validation. 10-fold cross-validation created training and testing data sets needed to check the performance of each machine learning algorithm explained earlier.

Chapter 6 also discusses a real-time detection technique to effectively gather SSH compromises. A multi-threaded (three threads) Java application helped store the network packets, process them, and classify based on J48 rules. The total time (Big-Oh notation) was upper bounded by  $O(n)$ .

## 7.2 Future Scope of Work

The following items can be explored in the near future.

1. The proposed models use only honeypot traffic (only malicious traffic). On the Internet, both legitimate traffic and malicious traffic appear together. In future, both stealthy single-source SSH password guessing attacks and SSH compromises can be detected in the presence of legitimate traffic.
2. A bigger machine learning data set (containing several 1000s of rows) can be collected and tested using the models proposed in this work. Even though the models are justified theoretically, a big data set would help know the performance of the proposed methods.
3. State-of-the-art deep-learning techniques can be explored in the future. Several deep-learning algorithms are available for classification of the data. With



a big data set, deep-learning models would validate the effect of the features used in this work.

4. There are separate methods to detect stealthy SSH password guessing attacks and SSH compromises. A unified approach can be designed and implemented in an IDS system.
5. There are two types of stealthy distributed SSH password guessing attacks: co-ordinated and uncoordinated. It would be necessary to classify these two types of attacks. Moreover, in order to improve the IDS, differentiation of all three types of stealthy SSH password guessing attacks (single-source, distributed coordinated, distributed uncoordinated) will pose an interesting research problem.

# Bibliography

- [1] N. Hoque, Monowar H. Bhuyan, R.C. Baishya, D.K. Bhattacharyya, and J.K. Kalita. Network attacks: Taxonomy, tools and systems. *Journal of Network and Computer Applications*, 40:307 – 324, 2014. (Cited on page 2.)
- [2] Adrien de Beaupre. Distributed ssh brute force attempts on the rise again. <https://isc.sans.edu/diary/Distributed+SSH+Brute+Force+Attempts+on+the+rise+again/9031>, 06 2010. (Cited on page 2.)
- [3] Shawn Davenport. Slow brute force attack. <https://github.blog/2013-11-20-weak-passwords-brute-forced/>, 11 2013. (Cited on pages 2 and 15.)
- [4] McAfee. McAfee labs threats report, 9 2017. <https://www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-sept-2017.pdf>. (Cited on page 2.)
- [5] Daniel Cid. Ssh brute force compromises leading to ddos. <https://blog.sucuri.net/2016/09/ssh-brute-force-compromises-leading-to-ddos.html>, 09 2016. (Cited on page 2.)

- [6] Lance Spitzner. The honeynet project: Trapping the hackers. *IEEE Security and Privacy*, 1(2):15–23, March 2003. (Cited on pages 3 and 43.)
- [7] phibos (Github Profile Name). Dionaea tool. <https://github.com/DinoTools/dionaea>, 2016. (Cited on pages 3 and 45.)
- [8] Lukas Rist. Glastopf tool. <http://glastopf.org>, 2012. (Cited on pages 3 and 45.)
- [9] Niels Provos. A virtual honeypot framework. In *Proceedings of the 13th Conference on USENIX Security Symposium*, volume 13 of *SSYM'04*, Berkeley, CA, USA, 2004. USENIX Association. (Cited on page 3.)
- [10] desaster (Github Profile Name). Kippo ssh honeypot. <https://github.com/desaster/kippo>, 2018. (Cited on pages 3 and 46.)
- [11] Niels Provos and Thorsten Holz. *Virtual Honeypots: From Botnet Tracking to Intrusion Detection*. Pearson Education, 2007. (Cited on page 4.)
- [12] Wikipedia. Secure shell. [https://en.wikipedia.org/wiki/Secure\\_Shell](https://en.wikipedia.org/wiki/Secure_Shell), 01 2020. (Cited on page 5.)
- [13] T. Ylonen and C. Lonvick. The secure shell (ssh) transport layer protocol. RFC 4253, RFC Editor, January 2006. <http://www.rfc-editor.org/rfc/rfc4253.txt>. (Cited on pages 5 and 78.)
- [14] T. Ylonen and C. Lonvick. The secure shell (ssh) authentication protocol. RFC 4252, RFC Editor, January 2006. <http://www.rfc-editor.org/rfc/rfc4252.txt>. (Cited on pages 5, 66, and 78.)

- [15] T. Ylonen and C. Lonvick. The secure shell (ssh) connection protocol. RFC 4254, RFC Editor, January 2006. <http://www.rfc-editor.org/rfc/rfc4254.txt>. (Cited on pages 5 and 78.)
- [16] Behrouz Forouzan. “Private Addresses” in *TCP/IP Protocol Suite*, chapter 5, page 148. McGraw-Hill, Inc., USA, 4th edition, 2009. (Cited on page 7.)
- [17] Behrouz Forouzan. “NAT” in *TCP/IP Protocol Suite*, chapter 5, pages 149–152. McGraw-Hill, Inc., USA, 4th edition, 2009. (Cited on page 7.)
- [18] Eric Alata. *Observation, characterization and modeling of attack processes on the Internet*. PhD thesis, INSA of Toulouse, Dec 2007. [https://tel.archives-ouvertes.fr/tel-00280126/file/THESE\\_ERIC\\_ALATA\\_TSF.pdf](https://tel.archives-ouvertes.fr/tel-00280126/file/THESE_ERIC_ALATA_TSF.pdf). (Cited on pages 8, 79, and 80.)
- [19] Vincent Nicomette, Mohamed Kaâniche, Eric Alata, and Matthieu Herrb. Set-up and deployment of a high-interaction honeypot: experiment and lessons learned. *Journal in Computer Virology*, 7(2):143–157, May 2011. (Cited on pages 9, 31, and 87.)
- [20] Cyril Jaquier. Fail2ban. <http://www.fail2ban.org>, 2015. (Cited on page 9.)
- [21] Phil Schwartz. Denyhosts. <http://www.denyhosts.net>, 2008. (Cited on page 9.)
- [22] Kagan MacTane. Sshblock. <http://kagan.mactane.org/software/sshblock>, 2009. (Cited on page 9.)
- [23] Michele Mazzucchi, Jones, and Kevin Zheng. Sshguard. <https://www.sshguard.net>, 2017. (Cited on page 9.)

- [24] R.Gregory. sshdfilter. <http://abatis.org.uk/sshdfilter/>, 2010. (Cited on page 9.)
- [25] Daniel Gerzo. bruteforceblocker. <http://danger.rulez.sk/projects/bruteforceblocker/>, 2005. (Cited on page 9.)
- [26] Avinash Chopde. Blockhosts. <https://www.aczoom.com/tools/blockhosts/blockhosts.html>, 2005. (Cited on page 9.)
- [27] JoMo-Kun. Medusa. <https://github.com/jmk-foofus/medusa>, 2016. (Cited on page 12.)
- [28] lanjelot (GitHub Profile Name). Patator. <https://github.com/lanjelot/patator>, 2018. (Cited on page 12.)
- [29] Van Hauser. The hydra. <https://github.com/vanhauser-thc/thc-hydra>, 2018. (Cited on page 12.)
- [30] Fotis Chantzis. Ncrack. <https://github.com/nmap/ncrack>, 2019. (Cited on page 12.)
- [31] Nico. phrasendrescher. <http://www.leidecker.info/projects/phrasendrescher/index.shtml>, 2009. (Cited on page 12.)
- [32] Wikipedia. Botnet. <https://en.wikipedia.org/wiki/Botnet>, 01 2020. (Cited on page 12.)
- [33] Jerry Gamblin. Source code of mirai botnet. <https://github.com/jgamblin/Mirai-Source-Code>, 2017. (Cited on pages 13, 86, and 90.)

- [34] Michael Rash. A new ssh password guessing botnet. [http://cIPHERdyne.org/blog/2010/08/a-new-ssh-password-guessing-botnet-dd\\_ssh.html](http://cIPHERdyne.org/blog/2010/08/a-new-ssh-password-guessing-botnet-dd_ssh.html), 8 2010. (Cited on page 13.)
- [35] Peter Kalnai and Michal Malik. New linux/rakos threat: devices and servers under ssh scan (again). <https://www.welivesecurity.com/2016/12/20/new-linuxrakos-threat-devices-servers-ssh-scan/>, 12 2016. (Cited on page 13.)
- [36] Daniel Goldberg and Ofri Ziv. Bread and butter attacks. <https://www.guardicore.com/2018/11/butter-brute-force-ssh-attack-tool-evolution>, 11 2018. (Cited on page 14.)
- [37] ESET Research and Michal Malik. Linux shishiga malware using lua scripts. <https://www.welivesecurity.com/2017/04/25/linux-shishiga-malware-using-lua-scripts>, 4 2017. (Cited on page 14.)
- [38] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. Understanding the mirai botnet. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1093–1110, Vancouver, BC, 2017. USENIX Association. (Cited on pages 14, 85, 86, and 90.)
- [39] John E Dunn. Poorly secured ssh servers targeted by chalubo botnet. <https://nakedsecurity.sophos.com/2018/10/24/>

[poorly-secured-ssh-servers-targeted-by-chalubo-botnet/](#), 10  
2018. (Cited on page 14.)

- [40] Peter Hansteen. A low intensity, distributed brute-force attempt. <http://bsdly.blogspot.in/2008/12/low-intensity-distributed-bruteforce.html>, 12 2008. (Cited on page 14.)
- [41] Tom M. Mitchell. *Machine Learning*, chapter 1, page 2. McGraw-Hill, 1997. (Cited on page 16.)
- [42] Wikipedia. Supervised learning. [https://en.wikipedia.org/wiki/Supervised\\_learning](https://en.wikipedia.org/wiki/Supervised_learning), 2 2020. (Cited on page 17.)
- [43] Saed Sayad. Zeror. <http://saedsayad.com/zeror.htm>. (Cited on page 17.)
- [44] Robert C. Holte. Very simple classification rules perform well on most commonly used datasets. *Mach. Learn.*, 11(1):63–90, April 1993. (Cited on page 18.)
- [45] Tom M. Mitchell. “*Decision Tree Representation*”, in *Machine Learning*, chapter 3, pages 52–53. McGraw-Hill, 1997. (Cited on page 18.)
- [46] Eibe Frank and Ian H. Witten. Generating accurate rule sets without global optimization. In *ICML*, 1998. (Cited on page 18.)
- [47] P.-N.Tan, Michael Steinbach, and Vipin Kumar. “*Naive Bayes Classifier*” in *Introduction to Data Mining*, chapter 5, page 229. Pearson India Education Services Pvt. Ltd., India, 7 edition, 2019. (Cited on page 18.)

- [48] P.-N.Tan, Michael Steinbach, and Vipin Kumar. “*Support Vector Machine (SVM)*” in *Introduction to Data Mining*, chapter 5, page 254. Pearson India Education Services Pvt. Ltd., India, 7 edition, 2019. (Cited on page [19](#).)
- [49] P.-N.Tan, Michael Steinbach, and Vipin Kumar. “*Nearest-Neighbor classifiers*” in *Introduction to Data Mining*, chapter 5, page 221. Pearson India Education Services Pvt. Ltd., India, 7 edition, 2019. (Cited on page [20](#).)
- [50] P.-N.Tan, Michael Steinbach, and Vipin Kumar. “*Feature Subset Selection*” in *Introduction to Data Mining*, chapter 2, page 52. Pearson India Education Services Pvt. Ltd., India, 7 edition, 2019. (Cited on page [21](#).)
- [51] P.-N.Tan, Michael Steinbach, and Vipin Kumar. “*DBSCAN*” in *Introduction to Data Mining*, chapter 8, page 518. Pearson India Education Services Pvt. Ltd., India, 7 edition, 2019. (Cited on page [23](#).)
- [52] P.-N.Tan, Michael Steinbach, and Vipin Kumar. “*The Silhouette Coefficient*” in *Introduction to Data Mining*, chapter 8, page 533. Pearson India Education Services Pvt. Ltd., India, 7 edition, 2019. (Cited on pages [24](#) and [102](#).)
- [53] Mobin Javed and Vern Paxson. Detecting stealthy, distributed ssh brute-forcing. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 85–96. ACM, 2013. (Cited on pages [27](#), [35](#), [39](#), [41](#), [42](#), [59](#), [75](#), [77](#), [82](#), and [83](#).)
- [54] Jim Owens and Jeanna Matthews. A study of passwords and methods used in brute-force ssh attacks. Technical report, Clarkson University, 2008. (Cited on page [30](#).)



- [55] Yu. Adachi and Y. Oyama. Malware analysis system using process-level virtualization. In *Computers and Communications, 2009. ISCC 2009. IEEE Symposium on*, pages 550–556, July 2009. (Cited on page 31.)
- [56] John P John, Fang Yu, Yinglian Xie, Arvind Krishnamurthy, and Martín Abadi. Heat-seeking honeypots : Design and experience. In *World Wide Web (WWW), 2011 International Conference on*, pages 207–216, 2011. (Cited on page 31.)
- [57] John P. John, Fang Yu, Yinglian Xie, Martín Abadi, and Arvind Krishnamurthy. Searching the searchers with searchaudit. In *Proceedings of the 19th USENIX Conference on Security*, USENIX Security’10, pages 9–9, Berkeley, CA, USA, 2010. USENIX Association. (Cited on page 31.)
- [58] Romain Bezut and Vivien Bernet-Rollande. Study of dictionary attacks on ssh. Technical report, Universite de Technologie Compiègne, 2010. [https://files.xdec.net/TX\\_EN\\_Bezut\\_Bernet-Rollande\\_BruteForce\\_SSH.pdf](https://files.xdec.net/TX_EN_Bezut_Bernet-Rollande_BruteForce_SSH.pdf). (Cited on pages 31 and 87.)
- [59] Saurabh Chamotra, J. S. Bhatia, Raj Kamal, and A. K. Ramani. Deployment of a low interaction honeypot in an organizational private network. In *Emerging Trends in Networks and Computer Communications (ETNCC), 2011 International Conference on*, pages 130–135. IEEE, April 2011. (Cited on pages 31 and 58.)
- [60] Sanjeev Kumar, Paramdeep Singh, Rakesh Sehgal, and J. S. Bhatia. Distributed honeynet system using gen iii virtual honeynet. *International Journal of Computer Theory and Engineering*, 4(4):537–541, August 2012. (Cited on pages 32 and 58.)

- [61] AbdelRahman Abdou, David Barrera, and Paul C. van Oorschot. *What Lies Beneath? Analyzing Automated SSH Brute-force Attacks*, pages 72–91. Springer International Publishing, 2016. (Cited on pages [32](#), [39](#), [59](#), [64](#), and [87](#).)
- [62] Stewart Sentanoe, Benjamin Taubmann, and Hans P. Reiser. Virtual machine introspection based ssh honeypot. In *Proceedings of the 4th Workshop on Security in Highly Connected IT Systems, SHCIS '17*, page 13–18, New York, NY, USA, 2017. Association for Computing Machinery. (Cited on page [32](#).)
- [63] M. Kumagai, Y. Musashi, D. A. L. Romana, K. Takemori, S. Kubota, and K. Sugitani. Ssh dictionary attack and dns reverse resolution traffic in campus network. In *Intelligent Networks and Intelligent Systems (ICINIS), 2010 3rd International Conference on*, pages 645–648, Nov 2010. (Cited on pages [33](#) and [41](#).)
- [64] M.M. Najafabadi, T.M. Khoshgoftaar, C. Kemp, N. Seliya, and R. Zuech. Machine learning for detecting brute force attacks at the network level. In *Bioinformatics and Bioengineering (BIBE), 2014 IEEE International Conference on*, pages 379–385, Nov 2014. (Cited on pages [33](#) and [41](#).)
- [65] Anna Sperotto, Ramin Sadre, Pieter-Tjerk Boer, and Aiko Pras. Hidden markov model modeling of ssh brute-force attacks. In *Proceedings of the 20th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management: Integrated Management of Systems, Services, Processes and People in IT, DSOM '09*, pages 164–176, Berlin, Heidelberg, 2009. Springer-Verlag. (Cited on page [33](#).)

- [66] Laurens Hellemons, Luuk Hendriks, Rick Hofstede, Anna Sperotto, Ramin Sadre, and Aiko Pras. Sshcure: a flow-based ssh intrusion detection system. In *Dependable Networks and Services*, volume 7279 of *Lecture Notes in Computer Science*, pages 86–97, Berlin, Germany, June 2012. Springer Verlag. (Cited on pages [34](#) and [41](#).)
- [67] phaag (Sourceforge Username). Nfsen. <http://nfsen.sourceforge.net/>, 2011. (Cited on page [34](#).)
- [68] drike and Rick Hofstede. Sshcure. <http://sshcure.sf.net>, 2016. (Cited on page [34](#).)
- [69] M. Jonker, R. Hofstede, A. Sperotto, and A. Pras. Unveiling flat traffic on the internet: An ssh attack case study. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 270–278, May 2015. (Cited on page [34](#).)
- [70] Erwan Le Malecot, Yoshiaki Hori, Kouichi Sakurai, Jae-Cheol Ryou, and Heejo Lee. (visually) tracking distributed ssh brute force attacks? In *Proceedings of the 3rd International Joint Workshop on Information Security and Its Applications (IJWISA 2008)*, pages 1–8, 2008. (Cited on pages [34](#), [39](#), [41](#), [59](#), and [77](#).)
- [71] Satomi Honda, Masahiko Takenaka, Yuki Unno, Koji Maruhashi, and Satoru Torii. Detection of novel-type brute force attacks used ephemeral springboard ips as camouflage. volume 2, 04 2014. (Cited on pages [35](#), [41](#), and [77](#).)
- [72] Satomi Saito, Koji Maruhashi, Masahiko Takenaka, and Satoru Torii. Topase: Detection and prevention of brute force attacks with disciplined

- ips from ids logs. *Journal of Information Processing*, 24(2):217–226, 2016. (Cited on pages [35](#), [36](#), [41](#), and [77](#).)
- [73] Gregory Conti and Kulsoom Abdullah. Passive visual fingerprinting of network attack tools. In *Proceedings of the CCS Workshop on Visualization and Data Mining for Computer Security, VizSEC/DMSEC '04*, pages 45–54, New York, NY, USA, 2004. ACM. (Cited on pages [36](#), [37](#), [41](#), [108](#), and [109](#).)
- [74] F. Pouget and M. Dacier. Honeypot-based forensics. In *In AusCERT Asia Pacific Information Technology Security Conference 2004 (AusCERT2004, Brisbane, AUSTRALIA, 5 2004*. (Cited on pages [37](#), [41](#), [108](#), and [109](#).)
- [75] Olivier Thonnard and Marc Dacier. A framework for attack patterns' discovery in honeynet data. *Digital Investigation*, 5:S128 – S139, 9 2008. The Proceedings of the Eighth Annual DFRWS Conference. (Cited on pages [37](#), [41](#), [108](#), and [109](#).)
- [76] Hyunsang Choi, Heejo Lee, and Hyogon Kim. Fast detection and visualization of network attacks on parallel coordinates. *Computers and Security*, 28(5):276 – 288, 2009. (Cited on pages [37](#), [41](#), [106](#), and [109](#).)
- [77] Mohammed H. Sqalli, Syed Naeem Firdous, Khaled Salah, and Marwan Abu-Amara. Classifying malicious activities in honeynets using entropy and volume-based thresholds. *Security and Communication Networks*, 6(5):567–583, 2013. (Cited on pages [38](#), [41](#), [108](#), and [109](#).)
- [78] Abdallah Ghourabi, Tarek Abbes, and Adel Bouhoula. *Behavior Analysis of Web Service Attacks*, volume 428, pages 366–379. Springer, Berlin, Heidelberg, 2014. (Cited on pages [38](#) and [42](#).)

- [79] Katerina Goseva-Popstojanova, Goce Anastasovski, Ana Dimitrijevikj, Risto Pantev, and Brandon Miller. Characterization and classification of malicious web traffic. *Computers & Security*, 42(0):92 – 115, 2014. (Cited on pages 38 and 42.)
- [80] A. Satoh, Y. Nakamura, and T. Ikenaga. Ssh dictionary attack detection based on flow analysis. In *Applications and the Internet (SAINT), 2012 IEEE/IPSJ 12th International Symposium on*, pages 51–59, July 2012. (Cited on pages 40, 42, 112, and 135.)
- [81] Rick Hofstede, Luuk Hendriks, Anna Sperotto, and Aiko Pras. Ssh compromise detection using netflow/ipfix. *SIGCOMM Comput. Commun. Rev.*, 44(5):20–26, October 2014. (Cited on pages 40, 42, 67, 112, and 135.)
- [82] Bojan Zdrnja. Second-generation (gen ii) honeypots, 2004. (Cited on page 43.)
- [83] Know your enemy: Honeywall cdrom roo. <http://old.honeynet.org/papers/cdrom/>, 2005. (Cited on page 44.)
- [84] Pavol Sokol, Jakub Míšek, and Martin Husák. Honeypots and honeynets: issues of privacy. *EURASIP Journal on Information Security*, 2017, 12 2017. (Cited on page 44.)
- [85] Paul Baecher, Markus Koetter, Thorsten Holz, Maximillian Dornseif, and Felix Freiling. The nepenthes platform: An efficient approach to collect malware. In Diego Zamboni and Christopher Kruegel, editors, *Proceedings of the 9th International Conference on Recent Advances in Intrusion Detection*,

- volume 4219 of *Lecture Notes in Computer Science*, pages 165–184, Berlin, Heidelberg, 2006. Springer-Verlag Berlin Heidelberg. (Cited on page 45.)
- [86] Lukas Rist. Snare tool. <http://mushmush.org>, 2017. (Cited on page 45.)
- [87] Lance Spitzner. Open source honeypots: Learning with honeyd. <http://www.symantec.com/connect/articles/open-source-honeypots-learning-honeyd>, 2010. (Cited on page 47.)
- [88] Yuqing Mai, R. Upadrashta, and Xiao Su. J-honeypot: a java-based network deception tool with monitoring and intrusion detection. In *Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004. International Conference on*, volume 1, pages 804–808 Vol.1, April 2004. (Cited on page 47.)
- [89] Daniel P. Berrangé. Virtual machine manager. <https://virt-manager.org>, 2013. (Cited on page 50.)
- [90] Kamil Koltys. An in-depth look at kippo: an integration perspective. <https://www.cert.pl/en/news/single/in-depth-look-at-kippo-an-integration-perspective/>, 2013. (Cited on page 63.)
- [91] Geolocation utilities. <https://www.maxmind.com>, 2017. (Cited on page 63.)
- [92] Iana source port. <https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml>, 2017. (Cited on page 65.)
- [93] Michal Zalewski. p0f v3 (version 3.09b). <http://lcamtuf.coredump.cx/p0f3/>, 2014. (Cited on page 71.)

- [94] Wikipedia. Entropy. [https://en.wikipedia.org/wiki/Entropy\\_\(information\\_theory\)](https://en.wikipedia.org/wiki/Entropy_(information_theory)), 2018. (Cited on page 88.)
- [95] P.-N.Tan, Michael Steinbach, and Vipin Kumar. “Variable Transformation”, in *Introduction to Data Mining*, chapter 2, pages 63–65. Pearson India Education Services Pvt. Ltd., India, 7 edition, 2019. (Cited on page 97.)
- [96] P.-N.Tan, Michael Steinbach, and Vipin Kumar. “Issues in Proximity Calculation”, in *Introduction to Data Mining*, chapter 2, page 83. Pearson India Education Services Pvt. Ltd., India, 7 edition, 2019. (Cited on page 98.)
- [97] P.-N.Tan, Michael Steinbach, and Vipin Kumar. “Missing Values”, in *Introduction to Data Mining*, chapter 2, pages 40–41. Pearson India Education Services Pvt. Ltd., India, 7 edition, 2019. (Cited on page 99.)
- [98] Peter Hansteen. The hail mary cloud and the lessons learned. <https://home.nuug.no/~peter/hailmary2013/thenumbers.html>, 2008. (Cited on pages 104 and 139.)
- [99] Y. Zhu and W. X. Zheng. Observer-based control for cyber-physical systems with periodic dos attacks via a cyclic switching strategy. *IEEE Transactions on Automatic Control*, pages 1–1, 2019. (Cited on page 110.)
- [100] Netresec. Splitcap tool. <http://www.netresec.com/?page=SplitCap>, 2013. (Cited on page 112.)
- [101] Sly Technologies. jnetpcap library. <http://jnetpcap.com>, 2016. (Cited on pages 117 and 129.)
- [102] Apache. Commons math. <https://archive.apache.org/dist/commons/math/RELEASE-NOTES.txt>, 2016. (Cited on pages 117 and 129.)

- [103] Eibe Frank. Zeror. <https://weka.sourceforge.io/doc.stable-3-8/weka/classifiers/rules/ZeroR.html>, 2017. (Cited on page 117.)
- [104] Tom M. Mitchell. “Alternative Measures for Selecting Attributes”, in *Machine Learning*, chapter 3, pages 73–74. McGraw-Hill, 1997. (Cited on page 118.)
- [105] John Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical Report MSR-TR-98-14, Microsoft Research, April 1998. (Cited on page 120.)
- [106] Wikipedia. Feature scaling. [https://en.wikipedia.org/wiki/Feature\\_scaling](https://en.wikipedia.org/wiki/Feature_scaling), 2019. (Cited on page 126.)
- [107] Rick Hofstede, Aiko Pras, Anna Sperotto, and Gabi Rodosek. Flow-based compromise detection: Lessons learned. *IEEE Security & Privacy*, 16:82–89, 01 2018. (Cited on page 135.)
- [108] P. Dumont, R. Meier, D. Gugelmann, and V. Lenders. Detection of malicious remote shell sessions. In *2019 11th International Conference on Cyber Conflict (CyCon)*, volume 900, pages 1–20, 2019. (Cited on page 136.)



# Publications

## Journal Papers

1. Gokul Kannan Sadasivam, Chittaranjan Hota, and Anand Bhojan. Detection of severe ssh attacks using honeypot servers and machine learning techniques. *Journal of Software Networking*, 2017(1):79–100, 01 2017.
2. Gokul Kannan Sadasivam, Chittaranjan Hota, and Anand Bhojan. Detection of stealthy single-source ssh password guessing attacks. *Journal of Evolving Systems [Accepted]*

## Conference Papers

1. Gokul Kannan Sadasivam and Chittaranjan Hota. Efficient detection of malwares using low-interaction honeypots. In *Intelligent systems, Computing and Information Technology (NCICIT), 2014 National Conference on*, Apr 2014
2. Gokul Kannan Sadasivam and Chittaranjan Hota. Scalable honeypot architecture for identifying malicious network activities. In *Emerging Information Technology and Engineering Solutions (EITES), 2015 International Confer-*

*ence on*, pages 27–31, Feb 2015

3. Gokul Kannan Sadasivam, Chittaranjan Hota, and Anand Bhojan. Classification of ssh attacks using machine learning algorithms. In *2016 Sixth International Conference on IT Convergence and Security (ICITCS)*, pages 1–6, Sept 2016
4. Gokul Kannan Sadasivam, Chittaranjan Hota, and Anand Bhojan. Honeynet data analysis and distributed ssh brute-force attacks. In *The International Conference Towards Extensible and Adaptable Methods in Computing, TEAMC 2018*, Netaji Subhas Institute of Technology, New Delhi, India, March 2018. 26-28 March

## Book Chapters

1. Gokul Kannan Sadasivam, Chittaranjan Hota, and Bhojan Anand. *Honeynet Data Analysis and Distributed SSH Brute-Force Attacks*, pages 107–118. Springer Singapore, Singapore, 2018

## Posters

1. Gokul Kannan Sadasivam and Chittaranjan Hota. Honeypot Challenges and Countermeasures, *Symposium on Information Security*, Hyderabad, Nov. 2013

# Biographies

## Brief Biography of the Candidate

Gokul Kannan Sadasivam is a Ph.D. scholar at the Department of Computer Science and Information Systems of Birla Institute of Technology and Science - Pilani, Hyderabad Campus. Prior to pursuing Ph.D., he received the B.E. degree from College of Engineering Guindy (Chennai, Tamil Nadu, India) in 2004 and the M.Sc. degree in from the National University of Singapore (Singapore) in 2007. He completed his second M.Sc. degree from the Northwestern Polytechnic University (Fremont, California, USA) in 2012. From 2012 to 2018, he was a Lecturer with the BITS, Hyderabad. His research interest includes the development of machine learning models to classify and cluster SSH password guessing attacks. Mr. Sadasivam was awarded the NPU Scholarship Award and a member of ACM (Association of Computing Machinery).

## Brief Biography of the Supervisor

Dr. Chittaranjan Hota, Professor of Computer Science and Engineering, completed his Ph.D from BITS, Pilani and has over thirty years of academic, research and administrative experience in Indian and universities abroad. His Ph.D work was

on QoS assurances in IP-Virtual Private Networks, a significant part of which was carried out at University of New South Wales, Sydney, Australia during his visit to Network Research Laboratory at UNSW under Indo-Australian joint research program. Prior to PhD, he had earned B.E in Computer Engineering, and M.E in Computer Sc. and Engineering.

Dr. Hota's research interests include developing algorithms and models for building systems and applications in areas like Big-data analytics, Internet of Things, and Cyber security in which he has more than 115 publications, in various journals and conferences. He is currently working on developing self-verifying codes for IoT devices under Indo-Dutch joint research programme with funding support from MeitY (GoI) and NWO (NL). In the past, he has executed funded projects with support from DeitY, UGC, and TCS in the areas of Network threat monitoring, and building secure Peer-to-Peer overlays using Artificial Intelligence, and Machine learning techniques.

Prof. Hota has also been involved in academic administration at BITS, Pilani over the past one and half decades. He is currently responsible for managing the entrance test (BITSAT) for admitting students into various degree programmes at BITS, Pilani and its campuses as its Professor In-Charge (PIC).

He had managed admissions to various degree programmes at BITS, Pilani Hyderabad Campus over the past five years as an Associate DEAN of Admissions. He had also managed admissions under International student admissions into three campuses of BITS, Pilani over the past five years. He was the founding HEAD of Dept. of Computer Science and Engineering (HOD) at BITS, Hyderabad for four years. He had also led a team of Network engineers in designing and commissioning the Campus Network and building other IT services at BITS, Hyderabad over

the past ten years as Faculty-In-Charge (FIC) of Information Processing Centre. He had shouldered the responsibilities of Unit Chief of Computer assisted house-keeping unit for 2 years in the formative years of BITS, Hyderabad where he was responsible for supporting institute wide payroll processing, and other HR functions. Dr. Hota was instrumental in formulating the ERP Roadmap for BITS, Pilani as its Task force leader under the University administration thrust area of BITS, Pilani's Mission 2012 project.

He has been a visiting researcher and visiting professor at University of New South Wales, Sydney; University of Cagliari, Italy; Aalto University, Finland; University of Tartu, Estonia; City University, London and Vrije University, Amsterdam over the past two decades. He is recipient of Australian Vice Chancellors' Committee award, twice recipient of Erasmus Mundus fellowship from European commission, and recipient of Certificate of Excellence from Kris Ramachandran Faculty Excellence award from BITS, Pilani. He is a Senior Member of IEEE. He is also a member of ACM, CSI, IE and Life Member of ISTE.

## **Brief Biography of the Co-Supervisor**

Dr Bhojan Anand, Senior Lecturer in School of Computing, National University of Singapore. He has received PhD in Computer Science & Engineering from National University of Singapore. He has received several awards including Research achievement award, University first rank, State rank scholarship. His thesis was nominated for the best PhD thesis award of the University. He was a mentor and visiting scholar at MIT Gambit lab (2008), USA. His research interests center on wireless networks, robotics and interactive virtual and augmented reality environments. He has published on premier networking and multimedia conferences

including IEEE-INFOCOM, ACM-SIGCOM, ACM-Mobisys, ACM-Netgames and ACM-Multimedia on a wide range of topics. He has published five books on mobile computing and networks. He has adapted international edition of the best selling book ‘Computer Networks – A top down approach / Kurose and Ross’. He has been serving as Organising Chair and Programme Chair of several International conferences and in the Program Committees of several International conferences. He is an invited speaker in international conferences and forums. He has received best-invited speaker award in M2M conference, organised by SIAA, Singapore. He is an Associate Editor of Computers and Electrical Engineering Journal, Elsevier. He has more than 20 years of teaching experience at various levels. Currently, at NUS he is teaching courses on computer networks, game development, virtual and augmented reality. He is a founding member Anuflora Systems, Virtual and Augmented Reality labs, and the Vice President of International Researchers club, Singapore. He is the faculty advisor for NUS Game Development Club, NUS eGaming Society, Singapore Medical Grand Challenge and School of Computing Technology Showcase & Entrepreneurship.