SSH Bruteforce Attack Detection with Network Flow Analysis

MS (Research) Thesis

By

Namrata Tiwari

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

JUNE 2021

SSH Bruteforce Attack Detection with Network Flow Analysis

A THESIS

submitted to the

INDIAN INSTITUTE OF TECHNOLOGY INDORE

in partial fulfillment of the requirements for

the award of the degree of

Master of Science (Research)

By

Namrata Tiwari

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

JUNE 2021

INDIAN INSTITUTE OF TECHNOLOGY INDORE

CANDIDATE'S DECLARATION

I hereby certify that the work which is being presented in the thesis entitled **SSH Bruteforce Attack Detection with Network Flow Analysis** in the fulfillment of the requirements for the award of the degree of **MASTER OF SCIENCE (RESEARCH)** and submitted in the **Department of Computer Science and Engineering, Indian Institute of Technology Indore,** is an authentic record of my own work carried out during the time period from July 2019 to June 2021 under the supervision of Dr. Neminath Hubballi, Associate Professor, Indian Institute of Technology Indore, Indore, India.

The matter presented in this thesis has not been submitted by me for the award of any other degree of this or any other institute.

Namrata. 25/06/2021
Signature of the Student with Date

(Namrata Tiwari)

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

25-06-2021
Signature of Thesis Supervisor with Date

(Dr. Neminath Hubballi)

Namrata Tiwari has successfully given her MS (Research) Oral Examination held on 25/10/2021

October 25, 2021
(Dr. Shanmugam Dhinakaran)
Signature of Chairperson, OEB

Date:

Signature of Convener, DPGC

Date: 25-Oct-2021

Signature of Thesis Supervisor

Date: 25-Oct-2021

Somnath Dey
Signature of Head of Department

Date: 25/10/2021

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my gratitude to people who, in one or the other way, contributed by making this time learnable, enjoyable, and bearable. At first, I would like to express my thankfulness towards my supervisor **Dr. Neminath Hubballi**, who was a constant source of inspiration during my work. This research work could not have been completed without his constant guidance, research directions, patience, and immense knowledge. His continuous support and encouragement have motivated me to remain streamlined in my research work. I am also grateful to **Dr. Somnath Dey**, HOD of Computer Science, for all his help and support.

I am thankful to **Dr. Bodhisatwa Mazumdar** and **Dr. Manish Kumar Goyal**, my research progress committee members, for taking out some valuable time to evaluate my progress all these years. Their valuable comments and suggestions helped me to improve my work at various stages.

My sincere acknowledgment and respect to **Prof. Neelesh Kumar Jain**, Director, Indian Institute of Technology Indore, for providing me the opportunity to explore my research capabilities at the Indian Institute of Technology Indore.

I would also like to thank **Pratibha Khandait**, my senior, for discussions and help during my thesis work.

I want to express my heartfelt respect to my parents, my brother for the constant love, care, and support they have provided to me throughout my life.

Namrata Tiwari

To you as a reader

ABSTRACT

Secure Socket Shell (SSH) exposes an interface for remote login. This provides convenience to system administrators for managing systems remotely and for other users it facilitates remote access to servers and applications over an unsecured network. SSH requires users to authenticate before they are allowed to access system and this can be done either through a key or by using passwords depending on the configuration. Using key based authentication is secure but it requires maintaining public key for every user and poses serious scalability issues. Thus, password based authentication methods are predominantly used in practice.

SSH servers which use password based authentication method are vulnerable to password guessing attacks known as bruteforcing where an adversary tries many passwords to gain access. These login attempts are recorded in a log file by the operating systems. We perform a log analysis case study with logs collected from a production level SSH server in our university campus network. With analysis, we find different types of failed logins, origin of attacks, common usernames used, etc. Our analysis showed that many sources persistently try to login showing recurrent attempts across weeks.

Owing to the scalability issues of log analysis for attack detection, we propose network based bruteforce attack detection methods. In the first place, our proposed methods separate network flows corresponding to failed and successful login cases. This is done using few statistical features derived from network flows. We propose two bruteforce attack detection methods using network flows. First one models the failed login cases generated due to users forgetting their passwords or misspelling as a probability distribution and detects unusually low probability events as bruteforcing attacks. In the second method, we build a Petri-Net based model to detect and classify the bruteforcing attacks. The classification marks the attack as either single source, single domain, and/or distributed attack. We evaluate our proposed model with data generated from a testbed setup and also form production level servers.

Contents

Abstract **iv**

List of Figures **iv**

List of Tables **vii**

1 Introduction **1**

1.1 Motivation 3

1.2 Thesis Contribution 4

1.3 Organisation of Thesis 6

2 Literature Survey **7**

2.1 Bruteforcing Attack Detection Tools 7

2.2 Coordinated Password Guessing Detection Techniques 10

2.3 Statistical and Machine Learning based Methods 12

2.4 Complementary Approaches 17

2.5 Log Analysis and Case Studies 20

2.6 Discussion and Conclusion 24

3 Secure Socket Shell Authentication Log Analysis **25**

3.1 Introduction 25

3.2 Prior Work 26

3.3 Network Architecture and Dataset Collection 27

3.4 Authentication Log Analysis 30

3.4.1 Analysis on Weekly Dataset 31
3.4.2 Analysis on Entire Dataset 35
3.5 Identifying Failed SSH Connections 39
3.6 Conclusion 45

4 SSH Bruteforce Attack Detection with Probability Distribution of Failed Logins 46
4.1 Introduction 46
4.2 Prior Work 47
4.3 SSH Bruteforce Attack Detection 48
4.3.1 Identifying Flows Corresponding to SSH Connections 49
4.3.2 Detecting Failed Logins 51
4.3.3 Detecting Bruteforcing Attacks 53
4.4 Experiments 54
4.4.1 Datasets 55
4.4.2 Detection Performance 56
4.4.3 Sensitivity Analysis 58
4.5 Conclusion 60

5 SSH Bruteforce Attack Detection and Classification with Petri-Net Modeling 61
5.1 Introduction 61
5.2 Prior Work 62
5.3 Proposed Bruteforce Attack Detection and Classification Method . . . 63
5.3.1 Problem Formulation 63
5.3.2 Bruteforcing Detection with Timed Colored Petri-Net Model . . 64
5.3.3 Setting Threshold Values for the Model 72
5.4 Experiments and Evaluation 75
5.4.1 Dataset Description: 75
5.4.2 Characteristics of Failed SSH Connections 77
5.4.3 Implementation and Evaluation: 79

5.4.4 Sensitivity Analysis 82
5.5 Detecting Stealth Bruteforce Attacks 83
5.6 Conclusion 86

6 Conclusion and Future Work **88**
6.1 Thesis Contribution 89
6.1.1 Secure Socket Shell Authentication Log Analysis 89
6.1.2 SSH Bruteforce Attack Detection with Probability Distribution Model 90
6.1.3 SSH Bruteforce Attack Detection and Classification with Pertri-Net Modeling 91
6.2 Future Work 92

List of Figures

3.1 Representative Network Architecture 28
3.2 Sample Log Entry 30
3.3 Geographic Sources of Failed Login Attempts Due to Invalid Usernames 36
3.4 Distinct IP Addresses of Failed Login Attempts with Guessed Usernames 37
3.5 Recurrent IP Addresses of Failed Login Attempts with Guessed Usernames 38
3.6 Common IP Prefixes of Failed Login Attempts with Guessed Usernames 38
3.7 Number of Packets Per Flow for Successful and Failed SSH Connections 41
3.8 Number of Bytes Per Flow for Successful and Failed SSH Connections . 41
3.9 Time Duration Per Flow for Successful and Failed SSH Connections . . 42

4.1 Sample Payload of a Packet in a SSH Flow 50
4.2 Poisson Probability Distribution with λ=5 53
4.3 Poisson Probability Distribution with λ=5 55
4.4 Probability Distribution of Failed Logins 58
4.5 Bruteforce Attack Detection Rate for Different Values of Threshold τ_1 . 59
4.6 Bruteforce Attack Detection Rate for Different Values of Threshold τ_2 . 60

5.1 Single Source based Bruteforcing 64
5.2 Single Domain based Bruteforcing 65
5.3 Distributed Bruteforcing 66
5.4 Proposed CPN Model Part-1 68
5.5 Proposed CPN Model Part-2 69

5.6 Number of Packets Per Flow for Bruteforce Attack and Successful SSH Connections 78
5.7 Number of Bytes Per Flow for Bruteforce Attack and Successful SSH Connections 78
5.8 Time Duration Per Flow for Bruteforce Attack and Successful SSH Connections 78
5.9 Bruteforce Attack Detection Rate for Different Values of Threshold τ_2 . 83
5.10 Bruteforce Attack Detection Rate for Different Values of Threshold τ_3 . 84
5.11 Bruteforce Attack Detection Rate for Different Values of Threshold τ_4 . 84

List of Tables

2.1 Literature for SSH Bruteforce Attack Detection 18
2.2 Literature for SSH Authentication Log Analysis 23

3.1 Weekly Log Data 30
3.2 Summary of Dataset Statistics 31
3.3 Summary of Invalid Usernames in SSH Login Attempts 31
3.4 Summary of Maximum Authentication Attempts Exceeded by Invalid User 33
3.5 Summary of Maximum Authentication Attempts Exceeded by Valid User 34
3.6 Summary of Connections Terminated due to Password Based Authentication 34
3.7 Top 10 Usernames used in Failed Login Attempts 35
3.8 Flow Level Features 40
3.9 Dataset Characteristics 42
3.10 Classification Performance of KNN Classifier 44
3.11 Classification Performance of Naive Bayes Classifier 45
3.12 Classification Performance of Random Forest Classifier 45
3.13 Classification Performance of Random Tree Classifier 45

4.1 Dataset Statistics of Normal Intervals 56
4.2 Dataset Statistics of SSH Bruteforce Attacks 56
4.3 Bruteforce Attack Detection Performance of Proposed Approach 58

5.1 Dataset Statistics of Normal Intervals 80

5.2 Dataset Statistics of SSH Bruteforce Attacks 81
5.3 Bruteforce Attack Detection 82
5.4 Prefix Generated from the Program 82
5.5 Dataset Statistics of SSH Stealthy Bruteforce Attack 85
5.6 Stealthy Bruteforce Attack Detection 85

1 Arcs in the Petri-Net model 104

Chapter 1

Introduction

Secure Socket Shell (SSH) is a cryptography based network protocol allowing remote users (usually system administrators) to access the system over an unsecured network. It is considered as a secured alternative to the Telnet as the communication session between the clients and server is encrypted using a session key. SSH exposes a `shell` to remote user, thus the user can execute any commands on the server and perform operations such as the secure transmission of files between the machines, all types of file operations, running applications, creation of backups at the server, management activities, and maintenance of remote machines, etc.

SSH provides two ways of remote login authentication. One is using a username and password and the other is using public-private key pair.

1. Password based Authentication: In password based authentication, a user authenticates herself by providing a username and password[1]. User supplied password is verified with the stored password in the machine.

2. Key based Authentication: In public-key based authentication, client and server machines possess public-private key pair. The public key is deployed in the server and private key is held by the client machine. For authentication, the server encrypts a *challenge* using the stored public key corresponding to the username and sends it to the client. The client machine then decrypts this *challenge* using its own private key and sends it back to the server. If the *challenge* received by the server

[1]Password is transmitted over a secure channel

matches with the original challenge, authentication is successful. Many SSH clients like Putty [1], have inbuilt facilities to generate such key pairs.

In general key based authentication is considered as secure, as only clients possessing the private key can decrypt the challenge. However, it has the operational issues of key generation and deployment. Moreover this exercise has to be repeated every time the user looses her private key or it is compromised. Thus password based authentication is practiced. Password-based authentication is prone to password guessing attacks as an adversary can guess the login credentials and gain access to the system, particularly if the password is not strong enough. Such password guessing attacks are common in web applications[2].

Exposing the interfaces of critical systems to the Internet [3] via SSH makes them vulnerable to possible attacks. Attackers can scan the network space looking for machines running SSH daemons (port number 22). After locating these machines, an attacker can try to gain unauthorized access by guessing the login credentials. Ill managed systems having user accounts with weak passwords, systems having default usernames and passwords become easier preys. This will compromise the system's security which can be catastrophic for an organization. Even if the user account passwords are changed, they may try guessing passwords or use stolen passwords for gaining illegal access [4]. Worst case scenario is to systematically try different guessed passwords on the target in the hope that one of them will work. This is generally known as *bruteforcing*. A bruteforcing attack is very hard to defend as there is little one can do in this case. A report published by Ponemon Institute [5] suggest that more than 50% of the organizations experienced such attacks against their servers. In general there are two ways bruteforce attacks can be detected.

1. Using Authentication Logs: Many Operating Systems like Linux, Unix maintain authentication logs. These logs will have details corresponding to all logins in the system. Details like username, time, date, the remote IP address from which user login was attempted, status of login (whether the login was successful or failed) are maintained as part of these authentication logs. As bruteforcing requires adversary to try many passwords, it results in many failed attempts being logged. Such system

logs can be consulted to find if there are excessive number of failed logins from any particular source. There are tools like Fail2ban [6] which process such logs and block those IP addresses involved in bruteforcing.

2. Using Network Traffic/Flow Details: Every connection/login attempt to the server results in exchanging of several packets between the client and server. These stream of packets are considered as network flows. Another way of detecting the brute-force attacks is using these network flows and some characteristics of such flows. For e.g. a repeated login attempt results in flows between SSH server and client.

Rest of this chapter is organized as follows. The motivation behind our work is presented in Section 1.1. In Section 1.2, we discuss the summary of thesis contributions and outline of the rest of thesis is described in Section 1.3.

1.1 Motivation

Protecting SSH servers, against bruteforce attacks is required for safe computing. These protections come in the form of filtering the connections originating from known offenders. This is typically done by adding a rule in firewall to filter/deny connections originating from such sources. This method will be effective if the source of attacks can be reliably identified. Authenticating logs maintained by systems are one of reliable sources of such information. Tools like SSHGaurd [7], Fail2ban [6] analyze such logs and put filters to deny connections from attack sources. However, this requires these tools to be installed in every machine running a SSH daemon. In a large network with hundreds of systems, installing and maintaining these tools may quickly become unmanageable. For e.g. the National Center for Super-computing Applications at University of Illinois has a network with a very large number of such SSH servers [8] which is used by many users for high performance computing. It is not uncommon to have such large pool of servers in a similar setup. In these environments, the authentication log based detection methods pose significant challenge of scalability. Further in a dynamic environment where such systems are added and removed frequently, keeping track of these changes is another major challenge.

As mentioned above, an alternative method for detecting bruteforce attacks is to use network traffic or flow level information. These network flows and traffic is collected either at a router/switch or at a perimeter security device itself. As these devices can see the traffic of every device inside the network, attacks against all SSH servers can be detected by deploying detection tools at this level. As these methods are scalable, recent works [9] [10] [11] have proposed to use network flow level information for detecting SSH bruteforce attacks.

Although network based detection methods are recent and scalable, they are limited to identifying whether bruteforcing is attempted or not against a server. However, an attacker can intelligently distribute the password guessing task across many sources[2] so that no single source generates password guesses aggressively. If the attack sources are distributed, the IP address and port parameters will change. These evasive techniques further harden the task of attack detection. The work found in the literature is limited to detecting only SSH bruteforce attacks. We aim at not only detecting attacks but also identifying any correlation between them. In specific we set the following two as objectives for our work.

1. Developing a method to detect the SSH bruteforce attacks using network packet/flow level information accurately.

2. As a second level activity, classify these attacks into categories based on possible correlation among sources of attacks.

1.2 Thesis Contribution

In this thesis we describe new SSH bruteforce attack detection methods and attack categorization to understand the possible correlation between the contributing sources. Before we present our attack detection methods, we conducted a study of SSH authentications logs to understand the attack patterns. Following are our thesis contributions:

[2]A compromised machine can be used to generate attacks against other machines.

I. SSH Authentication Log Analysis Case Study: Bruteforce attacks are common against SSH servers. As a first contribution, we report on a case study using logs of an SSH server deployed in a production environment. This study helped us to gain insights into the attack patterns, common sources of attacks, their frequency and distribution, etc. Our analysis on a large scale authentication log dataset reveals that attackers attempt various methods to break into the system, there are few common usernames which were tried persistently, origin of attacks are well spread and more than a handful number of sources make repeated attempts to break into the system spanning weeks.

II. SSH Bruteforce Attack Detection: As mentioned earlier, network traffic/flow based bruteforce attack detection methods are scalable and can be deployed at the perimeter of the network. In our second contribution, we propose a method for SSH bruteforce attack detection using network flows. This detection method is a two phase activity where in the first phase, flows belonging to SSH connections are identified. Subsequently the flows corresponding to failed login connections are detected. For the first task of identifying SSH flows, we use content similarity between flows using Deep Packet Inspection (DPI)[12]. SSH client and servers exchange a set of supported protocol suit as part of their handshaking to negotiate and agree on a particular protocol suit. This exchange is typically done in plain text form which allows to us to use DPI. For the second task of attributing a SSH flow to a failed login connection, we use few flow parameters which help in separating the connections of failed login attempts from successful login cases.

We subsequently proposed a method for detecting SSH bruteforce attacks. This is done by modeling the failed login connections as a probability distribution and we identify rare probability events corresponding to unusual number of failed logins in a chosen interval time period as bruteforce attacks. We experiment with network traffic collected from a production level SSH server and from testbed setup to evaluate the proposed method.

III. SSH Bruteforce Attack Detection and Classification: As discussed in previous section, one of the objectives of our work is to classify the SSH bruteforce attacks. We identify that these attacks can be of three possible categories as originating from single source, a domain or purely distributed. In this work, we extend our previous model of detecting attacks into both detection and classification method. In particular, we propose a Petri-Net based state transition model to keep track of various events and use it to not only detect the attacks but also to identify its category. As in the previous case, we use SSH network traffic collected from the production level SSH server and from testbed setup to evaluate the proposed method.

1.3 Organisation of Thesis

We organize the rest of this thesis as follows:

Chapter 2: In this chapter, we discuss the related work on SSH bruteforce attack detection, the mitigation tools available and also the case studies done using the SSH authentication logs.

Chapter 3: In this chapter, we present a case study on SSH Authentication logs from the production-level server, which is deployed in our university network. Through this study, we aim to get insights into the attack patterns.

Chapter 4: In this chapter, we describe a method for detecting SSH bruteforce attacks. We model the failed login attempts as a Poisson probability distribution and identify rare events as bruteforce attacks.

Chapter 5: In this chapter, we present a Petri-Net based model for the detection and classification of SSH bruteforce attacks. Our proposed model classifies the bruteforce attacks into one of the categories as originating from a single source, single domain, or distributed attacks.

Chapter 6: In this chapter, we conclude the work presented in the thesis with pointers to the future directions for followup work in this area.

Chapter 2

Literature Survey

Secure Socket Shell (SSH) is vulnerable to bruteforcing attack where an adversary attempts multiple passwords. These passwords come either from a dictionary or systematically generated. As compromised SSH servers allow users to access data in the target machine and may also allow users to navigate to other parts of the internal network, it is imperative that these are detected. The attack detection and prevention methods are of two types as host based and network based. In the host based approaches system logs are used for detecting attacks and in the network based detection methods network traffic or flows are used. We cover the different SSH bruteforce attack prevention and detection methods and related case studies in the next five subsections.

2.1 Bruteforcing Attack Detection Tools

There are few host based SSH bruteforce prevention tools. These tools block the IP addresses after a threshold number of failed logins in a time period. We elaborate on the working methodology and few features of these tools in the next few paragraphs.

1. `Fail2Ban` [6]: Fail2ban is an open-source software available for free. It parses the authentication log file and finds patterns of malicious behavior. It bans the IP address, which shows malicious behavior. Malicious behavior can be events such as too many authentication failures or exploit querying. It works with a

computer firewall by updating its rules and block a particular IP address for a certain period of time. If a pattern is observed for a particular IP address more than x number of times, it then bans that IP address. The value of x can be configured accordingly. Fail2ban is available for various distributions like Ubuntu, Fedora, Mac OS X, FreeBSD. It can be configured according to the user's needs. For example, permanently banning an IP address using 'Admin' as a username to login to the server.

2. `SSHGuard` [7]: SSHGuard is another open-source tool that protects the system from bruteforce attacks. It does so by parsing the authentication log files at the host and blocking the IPs with the help of a firewall such as iptables, pf, and ipfw. The IP address is unblocked after a certain amount of time. It also has an option of semi-permanently banning the IP address using blacklist. Multiple log files can be monitored at once using SSHGuard. It scans the logs and searches for attack patterns. This pattern is similar to the previous case where a x number of failed login attempts in a few seconds. It can be run on distributions such as Ubuntu, Debian, Mac OS X, OpenBSD, Fedora. SSHGuard can help prevent attack against various services such as OpenSSH, Exim, UWimap, Sendmail.

3. `BruteForceBlocker`[13]: BruteForeBlocker is a script written in perl language, which blocks traffic suspected to be of SSH bruteforce attack. It works with pf and iptables based firewall. BruteForeBlocker reads sshd log file via syslog and scans for possible failed logins. It then counts the number of such tries from the given IP address. If the count reaches a certain limit (which can be configured), it blocks all the traffic from that particular IP address, by adding an entry to the pf table corresponding to that IP address. New version of BruteForeBlocker 1.2 also allows sharing the list of such IP addresses with other users.

4. `DenyHosts`[14]: DenyHosts is a script that helps prevent SSH bruteforce attacks, exclusively for Linux servers. It scans the ssh logs line by line and filters out the login attempts which are failed and successful. It then stores the users with unsuccessful logins based on whether they are a root, valid(with an existing

system account), or invalid(with a non-existing system account). DenyHosts set a different threshold on the maximum number of times a valid user, a root user, and an invalid user can have failed login attempts. Whenever these thresholds are crossed, the user is blocked by adding the user to the */etc/hosts.deny* file. It also keeps a record of the host from which a user has logged-in successfully after many failed attempts, denoting them as suspicious logins.

5. `pam_abl` [15]: PAM Auto Blacklist Module or in short pam_abl, is a command-line tool configured to blacklist the users who perform repeated failed login attempts. Hosts are blacklisted when the number of failed authentication attempts in a particular interval exceeds a threshold. When this happens, even if the host now provides correct credentials, the authentication will fail. This is a precautionary measure to prevent an adversary gaining access after trying some guessed passwords. After a certain period of time, if the host stops authenticating itself, it is removed from the blacklist. The time period for cooling off period is configurable.

6. `BlockHosts` [16]: BlockHosts is a script that keeps track of the number of times a service is probed. It parses the logs from the sshd and checks if there are failed authentication attempts. BlockHosts keep track of the number of times a connection source (an IP address) made a failed login attempt. Whenever this value exceeds a predefined threshold value(can be configured), it adds the IP address to */etc/hosts.allow* and adds a deny flag with it. Subsequently, if that source (IP address in question) to connect, a refused connection message will be sent. Alternatively, all kinds of communication can be blocked from that IP address by using packet filtering.

All the tools mentioned above run at host level and need access to the host level log files. Although these have better visibility into the activities of the server, in a large network setup with hundereds of SSH servers they quickly become unmanageable requiring system administrator to individually manage systems.

2.2 Coordinated Password Guessing Detection Techniques

An attacker can intelligently coordinate while executing a SSH bruteforce attack so as to keep a low profile. Coordinated attacks are those attacks that are carefully orchestrated such that they do not originate from a single source. Multiple machines are involved, each contributing to the attack. These sources are often geographically distributed and are specially designed to evade detection.

Author Javed and Paxon in [17] gave an approach to detect such stealthy SSH distributed attacks where individual attackers work in a low-profile manner. They used statistical change point detection method to detect attacks. They built a model for legitimate users' failed authentication attempts which are either due to users failing to recollect passwords or misspelling. The model was used as a detector for bruteforcing. The detector was tuned with a trade-off between the detected number of false positives and detection time. The authors used an 8-year SSH login dataset (authentication logs) from Lawrence Berkeley National Laboratory, comprising thousands of SSH servers with nearly 4000 users. The detection approach is divided into two steps; the first is to monitor when an attack occurs. The second is to determine the participant involved in the attack. In the first phase of detection, Aggregate Analyzer checks the probability distribution of the parameter for the extreme change. It then flags such changes as the attack case. In the second phase, Attack Participants Classifier aims to identify the attacker for which attack was flagged in the first phase. For each host which appears to be an attacker, a decision is made whether it is a legitimate user, a singleton attacker(working individually), or an attacker working collectively with other attackers so as to generate coordinated attack. To perform this classification, authors presented two steps. First one was to classify on the basis of past histroy. For e.g. if the host had successfully authenticated itself in the past, it is considered as a genuine misspelling. Next step for the remaining users was to find commonality between them in the set of systems or username they try to authenticate. Authors assume this would be effective because distributed attackers aim to

collectively accomplish a particular goal. Also, all of them are expected to run from the same automated software; therefore would have similar number of authentication trials. Work of javed and Paxon [17] is based on authentication log analysis and hence it also shares the same scalability issue as in other host based methods. Unlike this, we propose methods for detecting SSH bruteforce attacks and classification techniques using network flow level data. Network level detection techniques have the advantage of making real-time detections.

Such coordinated attacks are found in other security areas as well. For e.g. in port scanning, the goal is to scan a set of ports to check which ports are open. Like in the bruteforcing, this can be done from a single source or a bunch of machines can be employed.
Carry Gates in [18] presented a similar problem of coordinated port scan detection. Her work was to detect distributed port scanning. In coordinated port scanning, attacker operates with multiple systems, where open ports are scanned by these systems, each scanning some section of network so as to cover the whole target network. Intrusion detection systems are capable of identifying port scan activity from a single source. However, if the activity is collaborated by attacker via multiple sources, it goes unrecognised. The detection problem was formulated as a set covering problem. In `Set Cover Problem`, the goal is to find subsets with minimum total costs such that all the elements in subsets cover the universal set. The assumption is that the attacker distributes the task in such a way that maximizes the reach and reducing the overlap of work. Author generated footprint pattern based on the port scan by an attacker, for e.g. the IP/Port pair observed by the attacker. The information from the footprint pattern was used to create the problem approach. Since a coordinated scan comprises of many single-source scans, the author first detected single-source scans. The input set comprised of all scans identified throughout a given time period. With the given attacker information from the footprint pattern and the set of scans, the problem was reduced to observing a subset of the scans such that a footprint pattern was identified. For experiments, coordinated scans were implemented in the isolated network, and the network traffic generated was captured. Total 87 data sets were

generated for validation, out of which 61 were recognized by the detector(70%). The proposed method has a limitation; in that, it uses single-source port scans as an input in the algorithm. Therefore, if an attacker is able to bypass the single-source scan detection, then coordinated scan detection can also be bypassed. Although her work is not on SSH attacks, the idea of launching coordinated attacks can be adapted to SSH login case too. A similar technique can be used for the detection of distributed SSH Bruteforce attacks.

2.3 Statistical and Machine Learning based Methods

These methods use some form of statistics/features derived using network packets or flows to detect SSH bruteforcing attacks. There are two sub-categories of works which are found in the literature as below.

I. Statistical based Methods: Sperotto et al. [19] proposed a Hidden Markov-based time-series model of SSH bruteforce attacks. They examined the attack characteristics from several bruteforce attack tools and determined some set of behavior to differentiate these attacks. The developed model was used to imitate characteristics of bruteforce attacks by creating the flow-based time series of the network traffic. Three attack phases were identified by the authors as

i) Scanning- Initially, the attacker executes a sequential SSH scan encompassing the whole network address space. This stage was referred to as the scanning phase (during the first 1000 seconds).The attacker gets information on which hosts are running a vulnerable SSH service during this phase. Only a few users are affected by this attack phase.

ii) Bruteforcing- Following this phase, the attacker launches an attack of user/password guessing, which is referred to as the bruteforce phase. Only a small portion of the network's hosts are involved in this phase. They measured that, this

phase undergoes a high level of communication between the victim and the attacker machine. This involves 1000 seconds of the second block. The bruteforce phase concludes after around 2000 seconds since the attack began.

iii) Compromising- Once the attack is successful, the host machines which are compromised tend to communicate with the attacker, which generated extra network traffic.

iv) Die-Off- The final phase is referred to as the die-off phase.

To evaluate the observations made, the authors used two datasets collected from the University of Twente network. They show that after being trained on the actual data, the model mimics the network behavior of bruteforce attacks on SSH servers within 10% relative error. Synthetic traces from the model resemble the mean, standard deviation, and correlation of flow, packet, and byte time series.

In a follow-up work, Sperotto et al. developed a tool named SSHCure [20] using the observations made above and combining some heuristics on packet frequency observed in four phases of bruteforce attack. SSHCure [20] is a flow-based open-source tool to detect real-time SSH bruteforce attacks. SSHCure uses metrics such as Packets-Per-Flow(PPF) and number of flow records in a time interval to identify different phases of attack. The observations of flows in four phases are as follows

i) Scanning Phase: SSHCure identifies the scanning phase when a maximum of 2 PPF and 200 flow records per 1 minute time interval are seen.

ii) Bruteforcing: A PPF value of 8-14 with a threshold value of 20 flow records per minute (for an individual attacker) will detect the bruteforce phase.

iii) Compromise Phase: An idle connection between an attacker and a victim target is indicated by traffic with a PPF less than 8; whereas a PPF Value greater than 14 may indicate that an attacker is actively using a victim.

iv) Die-Off Phase: The die-off phase is identified when there is a significant change in characteristics from the bruteforce phase. For this phase, PPF values greater than 8 less than 14 are considered.

For validation, two data sets were used to test the suggested detection technique.

The first set comprised of one week of data, also used in [19]. The second data set was similar to the first one, collected at the University of Twente network; however, it was collected in February 2012. Out of 29 total attack cases in the first data set, 17 attacks were moved to the bruteforce phase meeting the thresholds and 16 to the die-off phase. For the second dataset, out of 101 attack cases, 58 were identified as bruteforce attacks, of which 25 entered the die-off phase. The algorithm reported no False positives, with the identification of two false negatives.

Authors of [9] used Netflow records to improve their detection technique of SSH Attacks given in [20]. In this they updated the range of PPF for the bruteforce attack phase, with the new range as [11,51]. The minimum number of packets needed for authentication was claimed to be 11, and 51 was the maximum number of observed PPF in bruteforce traffic. The most frequently used number for PPF was taken as a baseline from the traffic, and deviations with this baseline were observed for the particular attack. After defining the baseline, the flow data was observed between the attacker and victim. Whenever two or more consecutive flows with the same value of PPF were observed, it was detected as an attack. They identified six possible attack scenarios. These scenarios were the actions that could be seen after a compromise. The next phase of compromised detection consists of two steps.

1. Matching traffic against scenarios: When a bruteforce phase between attacker and target is recognized, this phase seeks to detect one scenario out of the six scenarios initially generated. A compromise is identified if there is a match.

2. Identification of mitigation mechanisms: Following the discovery of a matching scenario, the traffic is examined for indicators of mitigating techniques that have been activated. When this happens, one or more of the following scenarios may occur: i) Due to retransmissions between the attacker and the target, mid-connection mitigation can result in a higher number of PPF than the established baseline, ii) TCP SYN is all that is required for new connections. Typically, a SYN flag set consists of three packets: a retry count and a commonly used

metric for establishing TCP connections. It is critical to identify mitigation methods, as it would generate false-positive compromise detections(those with instant logout.)

The authors gathered two datasets from the University of Twente's campus network, both consisting of data for one month. The dataset consists of the log files from honeynet (namely dataset D1) and workstation servers(namely Dataset D2), having publically accessible SSH service. The dataset is also comprised of the SSH flow data flowing in and out of the campus network. To validate the algorithm, implementation was done as a part of SSHCure v2.4[21]. Dataset D1 showed an accuracy of 84% with a 30% False-negative rate. Dataset D2 showed accuracy close to 100%, with 0.3% of the attacks incorrectly classified.

II. Machine Learning based Methods: Machine learning algorithms are extensively used in detecting various kinds of cyber attacks like DDoS attacks, SYN flood attacks, Port Scanning, etc. These machine learning algorithms are used as anomaly detectors by generating a model(s) of the normal behavior or both normal and abnormal behaviors. There are two types of algorithms as supervised and unsupervised. Both types of algorithms are used as anomaly detectors.

i) Supervised Machine Learning methods Supervised machine learning is a type of machine learning where labeled instances are used to train the model. The labeled data indicates that the input data instances have already been marked with the appropriate class. These machine learning algorithms can be used for SSH attack detection by generating a model with appropriate network flow features. In order to detect attacks, the supervised machine learning algorithms are trained with data which is labeled with ground truth (e.g. successful or a failed flow), based on which, they predict the class label for future network flows.

Najafabadi et al. [10] used netflow record based features to train four machine learning algorithms to detect bruteforce attacks. K-Nearest Neighbor, two types

of C4.5 decision trees (C4.5D and C4.5N), and Naive Bayes (NB) algorithms were used to generate predictive models using 8 features. These models were evaluated on a dataset collected from a campus network in 24 hour duration. They used Area Under the Receiver Operating Characteristic Curve (AUC) as a metric to show that these mdoels catch SSH bruteforce attacks with good accuracy. In a subsequent followup work, Najafabadi et al. [22] extended their work with 17 flow based features with same four algorithms. They evaluated these models with a dataset collected from a campus network for a period of 8 days again using Area Under the Receiver Operating Characteristic Curve (AUC) as a metric.

Hynek et al.. [23] did feature engineering on IP network flows and identified 11 features to be useful to train machine learning algorithms. They tested their features with five machine learning algorithms namely Ada-Boosted tree, Naive Bayes, K-NN, C4.5 Decision tree, and Random forest and showed good detection rate on SSH bruteforce attacks.

Shmagin [11] also used netflow based features with machine learning algorithms to detect SSH bruteforce attacks. Hossain et al. [24] proposed a method to detect brutefroce attacks. They used both conventional supervised machine learning algorithms and also Long Short-Term Memory (LSTM) based deep learning method for attack detection. Evaluation on a dataset containing bruteforce attacks against SSH and FTP showed good accuracy. A similar work by sadasivam et al. [25] also evaluated four machine learning algorithms using 14 features extracted from network flows.

ii)Unsupervised Machine Learning methods: Unsupervised learning is another type of machine learning in which models are trained on unlabeled data. The model then interprets the data based on some patterns. With a suitable algorithm, the model outputs the data in the form of clusters, with similar data in one cluster. This approach can be used for the detection of SSH attacks. Because of the difference in the flow features of normal and malicious traffic, unsupervised machine learning

algorithms can be used to divide or split the network traffic into attack or malicious and normal clusters using suitable features.

FLOWHACKER [26] is a network anomaly detector which detects attacks like DDoS, Botnet command and control and also SSH bruteforce attacks. It uses a bunch of features extracted from network flows and estimate the similarity among flows to group them using unsupervised machine learning algorithms. Once the flows are clustered, they are labelled either as belonging to normal traffic or attacks (which includes SSH bruteforcing).

He et al. [27] proposed a method to capture attacks at the source rather than at the target. They used a bunch of statistical features and explored both supervised and unsupervised methods to detect if any of the virtual machines in a cloud are contributing to DDoS and bruteforce attacks. SSH bruteforce attack is one of the attack that they considered in evaluation.
We present a summary of the important works for SSH Attack Detection in Table 2.1. The table shows various methods used for the detection of SSH attacks, their key findings, and the limitation associated with them.

2.4 Complementary Approaches

In this section, we describe some of the complementarry approaches, which can be used with other detection methods described earlier.
Identifying Keystroke (password) and Non-Keystroke Authentication: As mentioned earlier there are two ways in which an user authenticates for SSH login - Keystroke and Non-Keystroke.
i)Keystroke: Human-input character strings are used in Keystroke based authentication. This method of authentication is also used when offering a remote login, file transfer, and TCP/IP forwarding to users. Stored passwords are compared with the user entered password and if they match user is authenticated. These passwords are

Table 2.1: Literature for SSH Bruteforce Attack Detection

Method	Discussions	Limitations
Detection of Stealthy Distributed SSH Bruteforce Attacks [17]	Authors proposed the statistical technique of change-point detection for the detection of distributed malicious attacks.	Authors used authentication logs for analysis and hence is subjected to scalability issue.
Hidden Markov Model modeling of SSH brute-force attacks [19]	Authors proposed a time-series model to imitate the characteristics of bruteforce attacks.	The model fails in approximating the autocorrelation resulting in high randomness in the generated attack trace.
Flow-based SSH Intrusion Detection System [20]	Authors created SSHCure, a flow-based open-source tool to detect real-time SSH bruteforce attacks, using the observations made from [19].	The proposed algorithm cannot identify Distributed SSH dictionary attacks.
SSH Bruteforce Attack Detection using NetFlow/IPFIX [9]	Authors used Netflow records to improve their detection technique of SSH Attacks given in [20].	The proposed method cannot detect stealthy, distributed SSH attacks.
Brute Force Attack Detection using Machine Learning [10]	Authors trained four machine learning algorithms, namely, 5-Nearest Neighbor (5-NN), two forms of C4.5 Decision Trees (C4.5D and C4.5N), and Naive Bayes (NB) to detect bruteforce attacks.	The model cannot identify Distributed SSH Brute Force attacks.

not stored in plain-text format but in hashed format.

ii)Non-Keystroke: To avoid prompting a human, non-keystroke based authentication uses information other than a character string. Administrators can use SSH to automate operation functions with this authentication. Authentication via host and Public key-based are some examples of this method.

Satoh et al. [28], presented a new method for detecting user authentication methods on SSH connections and removing non-keystroke-based authentication connections. This strategy is based on two observations

i) An SSH dictionary attack is targeted on a host which uses the keystroke authentication.

ii) Tasks that are automated through SSH works with non-keystroke authentication. Authors aim was to identify the methods used for authentication then subsequently remove connections that exercise non-keystroke-based authentication. SSH protocol provides confidentiality and versatility by using a various MAC, cipher, and compression methods so as to create secure connections. Therefore, this anonymity of the SSH protocol makes authentication identification difficult by limiting direct packet examination. Therefore they used flow behaviors to identify the type of authentication. This is because: (i) Flow behaviors can be observed without performing a direct packet examination; (ii) Flow behaviors vary depending on the type of user authentication technique used. Various combinations of compression algorithms, cipher algorithms, MAC algorithms, and user authentication techniques were used to create the datasets. Traffic traces for both the successful and failed authentication were separated and used for validation. The flows corresponding to public-key based authentication were given accurate labels with a true positive value of 0.995; whereas, flows using keystroke based authentication were also labeled with a true positive of 0.997. Furthermore, in their experiments, they did not observer any false positives.

2.5 Log Analysis and Case Studies

Log analysis studies are an important direction of work in SSH related attack detection and cyber security research in general. These case studies give insights about the trends, patterns of attacks and behaviors of attackers which can be used as lessons to secure systems. Future detection/mitigation methods can take clues from these studies. We find broadly two classes of case studies related to SSH authentication logs as one taken from honeypot networks and second one collected from real servers. We elaborate studies of these two types below.

I. Case Studies using Honeynets and Honeypots: A honeypot is an alternative server which is setup mimicking a real server to lure attackers. Here an attacker gets a feeling that, she is communicating with the real server or system while she is only communicating with a dummy system. These systems deliberately expose vulnerable services to lure attackers and capture their footprint. Honeynet is a network of honeypots containing multiple such servers. There are several case studies done with logs collected by deploying SSH servers in the honeypots and honeynets.

2008: Owens and Matthews [29] deployed honeypot based SSH servers in three different types of networks namely i) a small business network ii) residential network and iii) a university campus network. From these servers they collected logs and analyzed multiple parameters. These logs correspond to bruteforce attacks against SSH servers deployed in Linux systems. Their study revealed that there are many similarities between the attack patterns among the three servers. Their study also showed that, there are many common usernames and passwords used against all the three servers indicating that the common tools were used to generate attacks[1]. Their study also showed slow rate and distributed attacks too.

2010: Romain and Vivien [30] built a whole network of systems and servers deploying several applications as part of a honeynet. In order to capture logs and activities of attackers, they wrote several custom scripts and patches. The logs collected were

[1]Such tools are often widely publicized and circulated

exported to a central LogServer and processed. Their study on SSH authentication log data collected over a period of four months revealed that many attacks recur over a period of time.

2013: Valli et al. [31] conducted authentication log analysis case study with logs collected from three Kippo honeypot systems. The logs were collected over a period of 75 days. The analysis revealed that nearly 50% of the attacks originated from China. They also found the commonly executed commands by the attackers after they login.

2018: Joshua Faust [32] also reported log analysis case study by deploying a honeynet. His study focuses on finding the country of origin of attacks, number of attempts made, etc. These logs were collected by deploying SSH servers at five different places namely Bangalore, Frankfurt, London, Singapore and San-Francisco. All the logs were aggregated at a central server and used for analysis.

2019: The paper [33] reports a 463-day log analysis of CAUDIT honeypot at the University of Illinois' National Center for Supercomputing Applications (NCSA). CAUDIT is a completely automated system that detects and eliminates hosts vulnerable to SSH bruteforce attacks. SSH attack attempts were blocked using the honeypot's recorded IP addresses.

II. Case Studies using Logs of Real SSH Servers: Many studies used authentication logs collected from real SSH servers and systems deployed in networks. These systems also generate similar logs as in the previous case. Unlike the previous case, as these are real servers used by intended users, and they do not have vulnerabilities in them.

2010: Sharma et al. [34] presented a SSH log analysis case study from the SSH servers of National Center for Supercomputing Applications at the University of Illinois. They analyzed logs collected over a period of five years in 5000 machines. Their study covered mechanisms used by attackers to gain access to the servers. In particular they observed that, attackers initially try exploiting a vulnerable application and subsequently escalate the privileges to root and even replace the

SSH/SSHD applications with a trojaned versions to permanently retain access.

2015: Abdou et al. [35] collected SSH logs by deploying servers in five different locations. They collected 17 million log records in a span of one year and gathered the commonly used passwords and usernames. Their study revealed that 'admin' and 'root' are the most commonly used usernames in the attacks and were seen across the SSH servers. They also found several attacks distribute usernames and passwords (dictionary entries) among a set of sources.

2019: CAUDIT proposed by Cao et al. [33] has a second component which replays the attacks observed in honeypot servers over the real servers to find the vulnerable systems using which necessary remedial action can be taken. Using the IP addresses of attack sources as seen in the honeypot servers, they setup filtering at a Black Hole Router (BHR) which helps in protecting the real SSH servers. CAUDIT also has a coordination mechanism where it can exchange the observed attack sources to many peers. This intelligence sharing mechanism helps in protecting these servers from future attacks and the authors claim that, it is able to detect/mitigate millions of such attacks on daily basis in their network.

2020: Another study by Wu. et al. [8] reports SSH bruteforce attempts against an operational system at the National Center for Supercomputing Applications with 11 billion number of such attacks. Their study revealed that several stolen SSH keys are used for authentication, and these keys were widely circulated. This measurement also unearthed that very diverse versions of SSH clients are used for login. There were attacks where activity patterns that showed human participation in the attack through botnets.

We present a summary of the important works related to SSH log analysis in Table 2.2. The table lists different log analysis case studies, crisp summary of their key findings, and the limitation associated with them.

Table 2.2: Literature for SSH Authentication Log Analysis

Method	Discussions	Limitations
A Study of Passwords and Methods Used in Brute-Force SSH Attacks [29]	Authors studied logs collected from honeypot servers in three different networks indicating the use of common tools for generating attacks.	Authors claim that the aggregation and analysis of the collected data is labor intensive and can be automated by software tools.
Study of Dictionary Attacks on SSH [30]	Authors designed the architecture with SSH servers on virtual machines to analyze attacks from the log data, revealing recurrent use of username password by attacker.	Authors used VPN for the virtual machine, which can cause scaling issues.
Distributed Analysis of SSH Brute Force and Dictionary Based Attacks [32]	The author deployed total 6 servers (one master, 5 honeypots) to collect and analyse the authentication log data.	Since only 5 locations were taken into consideration, the collected data did not represented the global view, instead a limited geography.
Continuous Auditing of SSH Servers to Mitigate Brute-force Attacks [33]	Authors proposed CAUDIT, an automated system to identify the hosts that are vulnerable to SSH attacks, and based on the data collected from honeypot, blocks the attempts from the recorded IP addresses.	The proposed system is not able to generate coordinated alerts among the sites, for the detection of coordinated attacks across different sites.

2.6 Discussion and Conclusion

SSH bruteforcing is an attack which needs system administrator's attention. It is important that these attacks are detected at early stage to prevent damage. While log analysis is effective and can identify attacks reliably, they have scalability issues. Several case studies using logs revealed interesting patterns and behavioral trends. Network based bruteforce detection methods are recent and can scale well. Taking motivation from this, we design network flow based SSH bruteforce attack detection methods which we describe in the subsequent chapters.

Chapter 3

Secure Socket Shell Authentication Log Analysis

3.1 Introduction

Computing and networked servers often allow users to remotely connect through Internet by exposing a Secure Socket Shell (SSH) interface [33]. Although remote access gives the convenience to legitimate users, often these systems are not well managed and many user accounts may have weak passwords which are guessed easily. The implications of a system which is compromised can be very severe. It may allow an attacker to navigate to other internal systems of an organization and may harvest sensitive data [36]. In general, an attacker can use multiple types of password guessing attacks and combine her computational resources to generate coordinated and well organized attacks [18, 34, 17]. Operating systems usually keep logs of such remote SSH connection requests as authentication logs. These logs have rich information and are one of the sources of information for detecting the attacks against SSH servers.

In this chapter we present a study conducted using SSH authentication logs collected from a production level system deployed in our university network. In particular our study focuses on gaining insights about attack patterns. In the second part we use the network traffic or flow level data corresponding to these logs and show that the flows corresponding to successful and failed SSH connections exhibit differ-

ent characteristics. These flow classification methods help to detect SSH attacks like bruteforcing. Summary of our study and observations are as below.

- We identify common usernames attackers tried to use for gaining access and observe that few of them are very common (top 10 usernames are used in 12.8% of break-in attempts).
- We mapped sources of attacks to their country-wise geographical location to find that malicious login attempts come from vary diverse sources and locations.
- We study the distribution of attacks for common IP addresses and prefixes, and observe that few of them are recurring and persistent across weeks.
- We propose a flow based detection method to identify successful and failed login attempts using few parameters taken from network flows.

We organize rest of this chapter as follows. In Section 3.2 we discuss the related work on SSH attack detection and prevention. In Section 3.3 we provide the generic architecture of network where SSH server is deployed and log dataset used for analysis. In Section 3.4 we furnish the details of our log analysis case study. Section 3.5 has the details of failed login identification method and its evaluation with few machine learning algorithms. Finally we conclude this chapter in Section 3.6.

3.2 Prior Work

As described in Chapter 2, there are few prior works which have studied the SSH authentication logs. These studies are of two types.

I. Logs Collected from Honeypots: Honeypots are systems primarily deployed to attract the attackers. These systems may deliberately expose vulnerable services and collect evidence and the behavior of attackers which are subsequently used to secure real systems. Few studies used authentication logs collected from the SSH servers deployed in honeypot networks to understand the attackers' behavioral patterns and sources of such attacks. Authors of [30] collected a dataset of SSH bruteforce attacks

using honeypots network and observe that many attacks recur over a period of time. Another measurement study [29] also reported recurrent attack behavior with analysis on data collected from honeynet system. A recent case study [32] also collected SSH logs by deploying a honeynet. This analysis involved various parameters like country of origin, number of attempts made, etc. CAUDIT [33] uses honeypot deployed SSH servers to attract intruders and using the IP addresses of such sources, filter the packets originating from these servers from a higher level router.

II. Logs Collected from Real Servers: These studies use authentication logs collected from production level servers. There are number of such case studies using real SSH server logs [35, 33, 37, 38, 8]. Wu et al. [8] presented a case study of logs collected from National Center for Supercomputing Applications at University of Illinois. Their study revealed that, several stolen keys and botnets are used to login to SSH servers. They also observed that attackers use several variants of SSH clients for login.

Drawing motivation from the above works, we conduct a log analysis case study with authentication logs collected from a SSH server deployed in our university network. Our observations and findings complement the above works by observing similar behavioral patterns.

3.3 Network Architecture and Dataset Collection

In this section we give an overview of network architecture in which SSH server is deployed and the details of how the log dataset is generated.

Network Architecture: Our university computer network architecture follows a popularly used network design of separating inside Local Area Network (LAN) with external Wide Area Network (WAN) through a firewall. There is also a Demilitarized Zone (DMZ) as shown in the Figure 3.1. Different parts of this network structure are elaborated below.

1. *Demilitarized Zone*: This name has its origin from the military background. In that context, a zone marked as DMZ prevents military installations from

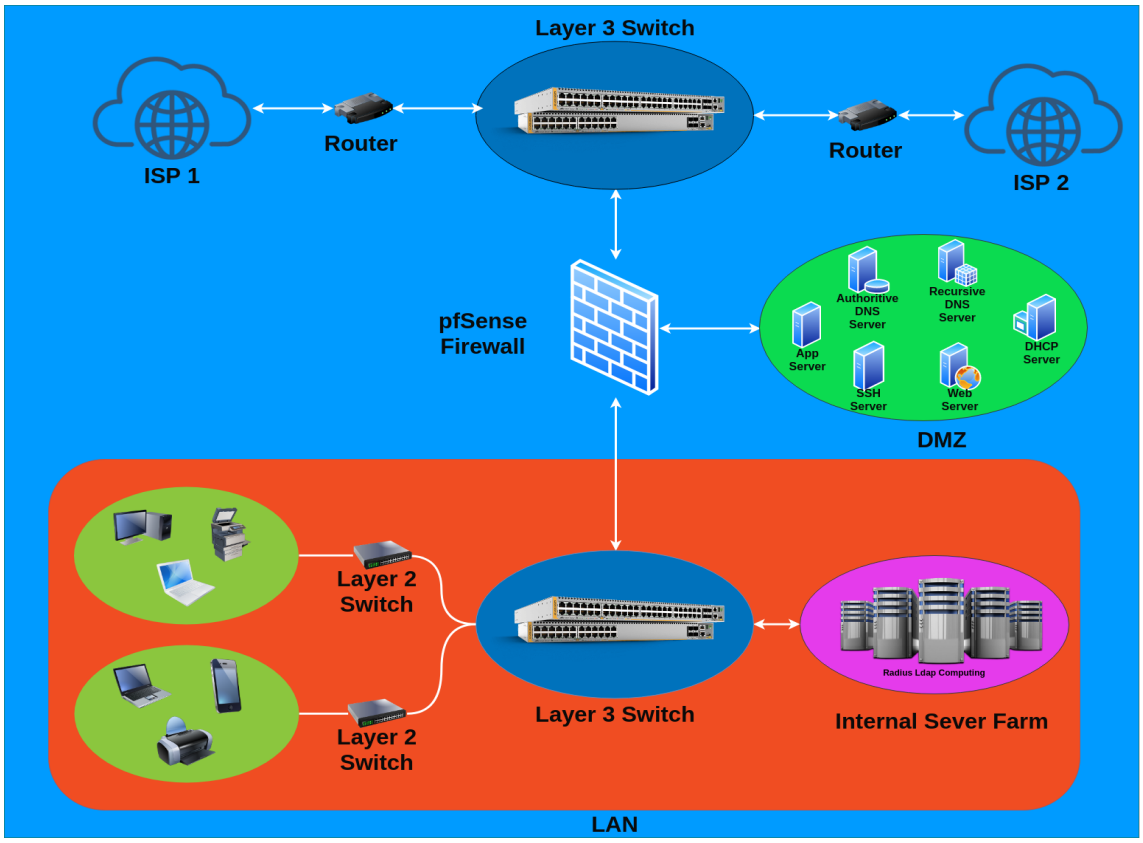

Figure 3.1: Representative Network Architecture

two neighboring countries. In the network context, this is used to separate the deployment of various services and servers. DMZ controls the access to services for users located in different areas. In our network, this DMZ has various servers like web server, authoritative name server, DNS resolvers, etc. The web server also exposes an SSH interface for remote access.

2. *Server Farm*: There is also a server farm (a collection of servers), which is primarily used for High Performance Computing (HPC) purpose by the internal users and also few external users. Some of these servers have also exposed SSH interfaces to Internet.

3. *Desktop and Internal Users*: These are end users who are part of the local network and they use Internet for day-to-day web access, education and entertainment purposes.

4. *Layer 3 Switch and ISP*: Our campus network has services of two Internet Service Providers (ISPs) whose connections terminate on a layer 3 switch as shown in Figure 3.1.

There is a software firewall *pfSence* [39] separating different zones of the network. This firewall screens both incoming and outgoing traffic and does filtering based on rules. This firewall also has an integrated intrusion detection and prevention system Suricata. Our university network infrastructure currently serves approximately 1800 users. Going with a conservative estimate of each user has on an average 1.5 devices, it provides connectivity to 2700 devices. The web server deployed in DMZ hosts our institute website and also runs SSH server which allows authorized users to remotely connect to the server. This will help users to access the server and also facilitates working on updates of the website remotely. This SSH server has many accounts for different types of users (faculty, students, staff, administrators) who primarily login to maintain and update their homepages and also the institute website. This SSH server allows only public key based user authentication and hence all password based login attempts are rejected. We collected two types of datasets from this network setup for our study. The first dataset is the authentication log dataset and second one is the network traffic dataset. We elaborate the details of these two in the next two paragraphs.

Dataset-I: As mentioned above, the web server located in the DMZ exposes an SSH interface to the remote users. Hence all the connections from external users are captured as part of the authentication logs (by default every user login attempt is recorded in the auth.log file). We collected authentication logs related to SSH connections (filtered from file auth.log) from the web server as our first dataset. This data was collected for a period of one month starting from 8 September 2019 to 6 October 2019. The log dataset we collected consists of 1,04,821 number of login attempt entries. Few sample log file entries are shown in Figure 3.2. Each line has a timestamp indicating when the event took place, details of the event and source IP address which was involved in the event. The three lines snippet in the Figure 3.2 show that username "admin" was tried from a client having IP address 92.XX.YY.90 and was found

to be invalid. Subsequently connection request was closed.

```
Sep 8 09:49:16 Server sshd[17901]: Invalid user admin from 92.XX.YY.90 port
53108
Sep 8 09:49:16 Server sshd[17901]: input_userauth_request: invalid user admin
[preauth]
Sep 8 09:49:16 Server sshd[17901]: Connection closed by 92.XX.YY.90 port
53108 [preauth]
```

Figure 3.2: Sample Log Entry

Dataset-II: The second dataset we collected is of network traffic from the SSH server. We collected this traffic by deploying tcpdump [40] tool and setting a filter for port number 22 (SSH runs on this port number). This data was collected from 01-01-2020 to 30-01-2020. We use this datset to show that network traffic can be used instead of log data to identify failed SSH connections. We also make use of authentication log data for validation and ground truth. This will help in cross verifying the results with authentication logs for accuracy.

3.4 Authentication Log Analysis

We used the Dataset-I generated (as described above) to study the trends and patterns of login attempts from different perspectives. We perform two sets of experiments to understand various trends and patterns. First study is on weekly data and

Table 3.1: Weekly Log Data

Week	**From**	**To**
Week-1	08-sept-2019	15-sept-2019
Week-2	16-sept-2019	22-sept-2019
Week-3	23-sept-2019	29-sept-2019
Week-4	30-sept-2019	06-oct-2019
Week-5	07-oct-2019	09-oct-2019

second one is the entire dataset. For the first set of experiments we divided the entire dataset into five parts as one log file per week as shown in Table 3.1. This table shows the starting and ending date for each week. Table 3.2 summarizes the number of log entries and size of log file on per week basis.

Table 3.2: Summary of Dataset Statistics

Period	Number of Log Entries	Size of Log File (in MB)
Week-1	7271	0.693
Week-2	7758	0.739
Week-3	78668	7.9
Week-4	7105	0.691
Week-5	4019	0.393

For the second part, we used the entire dataset for deriving statistics and trends. In the next two subsections we elaborate on these two experiments.

3.4.1 Analysis on Weekly Dataset

For this experiment, we used the log files representing the weekly cases for deriving statistics and understanding trends of login attempts. We performed four types of analysis as elaborated below.

I. Usage of Invalid Usernames in Login Attempts: Our first analysis was to understand the usage of invalid usernames used in the login attempts. A user is invalid if she does not have a login account in the target machine. Here the attacker is guessing the usernames along with login credential.

Table 3.3: Summary of Invalid Usernames in SSH Login Attempts

Period	Distinct Remote Clients	Distinct IP Address with Failed Logins	Number of Invalid User Login Attempts	Average Attempts Per Attack IP Address	Most Common Username	Number of Trials
Week-1	1126	725	947	1.30	admin	234
Week-2	1186	854	1073	1.25	admin	225
Week-3	1145	821	12160	14.81	admin	450
Week-4	1014	541	633	1.17	admin	99
Week-5	516	64	95	1.48	admin	31

Table 3.3 summarizes the details of this study. In the table we have shown the time period of the log entries (on weekly basis), the number of distinct clients connecting to the server, number of clients involved in failed login attempts, distinct number of

invalid usernames used in the failed login attempts, average number of failed attempts per unique client, most commonly used username in failed login attempts and number of times that username has been tried for login. We derive the number of unique SSH clients by counting the total number of unique IP addresses seen in the dataset during that week. This includes both legitimate and successful login cases and also the failed attempts as well. The entries in the third column depicts the number of clients which had failed login attempts again identified by the unique IP addresses. Fourth column indicates the total number of login cases which were recorded as having invalid usernames in that week. Fifth column indicates the average number of failed login cases per IP address. Sixth column shows the most commonly used username and the last column indicates the number times the frequent username has appeared in the failed login cases. For e.g. the first row represents the log analysis for the Week-1 (between 08-Sep-2019 to 15-Sep-2019) having received login attempts from 1126 clients out of which 725 had atleast one failed login attempts. A total of 947 distinct invalid usernames were attempted during this period with an average number of 1.3 invalid usernames per distinct client. During this week, the most commonly used username was "admin" which was tried 234 times in total. We can also notice from this table that "admin" remains the most commonly used username across all the five weeks.

II. Maximum Authentication Attempts Exceeded by Invalid Users: In our second analysis we studied the authentication attempts exceeded cases by different clients. These are the instances where an invalid username has been attempted multiple times with private key being used to authenticate the user. SSH servers using key based authentication use a threshold known as `MaxAuthTries`. This threshold by default is set to 6 and if any client offers half of this value number of invalid keys over a connection, the server will log such cases. The SSH server will reset the connection after `MaxAuthTries` number of failed login attempts. Table 3.4 shows the summary of such maximum failed authentication cases in our weekly logs. The first two columns of this table are identical to the first two columns of Table 3.3, while third column shows the number of unique IP addresses from which such maximum authentication failure cases were seen. Fourth, fifth, sixth and seventh columns show the number of

Table 3.4: Summary of Maximum Authentication Attempts Exceeded by Invalid User

Period	Distinct Remote Clients	Distinct IP Addresses with Authentication Attempts Exceeded	Number of Maximum Authentication Attempts Exceeded	Average Attempts Per Attack IP Address	Most Common Username	Number of Trials
Week-1	1126	50	52	1.04	admin	41
Week-2	1186	54	55	1.01	admin	42
Week-3	1145	20	76	3.8	admin	39
Week-4	1014	10	11	1.1	admin	10
Week-5	516	2	2	1	admin, usuario	1,1

times maximum authentication attempts exceeded message was seen in the logs, average such connection attempts made by clients, most commonly used username against which maximum authentication failure cases were seen and number of times using that username authentication failure cases were seen in the entire week data respectively. For e.g. the first row of Week-1 data, there were 50 unique addresses which generated failed attempts by presenting invalid keys over a session, with a total of 52 attempts and an average of 1.04 such attempts per IP address. The most common username used for login (using wrong key) was "admin" which was used 41 times out of the total 51 such cases in that week. It is worth noting that the last row has two usernames each appearing single time.

III. Maximum Authentication Attempts Exceeded by Valid User Cases: In our third trend analysis, we studied another type of login failure instances where valid username being used but with the invalid key being attempted. Table 3.5 shows the summary of such failure cases on a weekly basis. In this case too the first two columns are identical to the first two columns of Table 3.3. Third column shows the distinct IP addresses which were the sources of such authentication failure instances. Fourth column has the total number of such failure instances seen in that week. Fifth, sixth and seventh column show the average number of authentication exceeded cases per IP address, most commonly used username and total number of times the popular username has been used in these failed attempts respectively. We can notice from the table that username "root" is being used in all the cases and it is a valid username in our system. Hence, fourth and seventh columns have identical numbers.

Table 3.5: Summary of Maximum Authentication Attempts Exceeded by Valid User

Period	Distinct Remote Clients	Distinct IP Addresses with Authentication Attempts Exceeded	Number of Maximum Authentication Attempts Exceeded	Average Attempts Per Attack IP Address	Most Common Username	#Trials
Week-1	1126	72	75	1.04	root	75
Week-2	1186	76	81	1.06	root	81
Week-3	1145	34	61	1.79	root	61
Week-4	1014	20	24	1.2	root	24
Week-5	516	9	11	1.22	root	11

IV. Connection Termination Due to Password Based Authentication: In our last analysis of weekly logs, we counted the sources of failed logins due to password based authentication being attempted. As our server does not allow password based authentication, all such connection requests were refused as "disconnection requests" from clients. Table 3.6 shows the statistics of such failures. With first two columns again being same as in Table 3.3, entries in the third column indicate the distinct source IP addresses from which these attempts were made. Fourth column shows the number of such requests in that week and last column showing the average number of requests per IP address.

Table 3.6: Summary of Connections Terminated due to Password Based Authentication

Period	Distinct Remote Clients	Distinct Source IP Addresses	Disconnects due to Password based Login	Average Attempts Per IP Address
Week-1	1126	132	1404	10.63
Week-2	1186	68	1259	18.51
Week-3	1145	106	4962	46.81
Week-4	1014	93	1721	18.5
Week-5	516	77	1376	17.87

3.4.2 Analysis on Entire Dataset

For our second set of experiments, we used the entire dataset to understand three important trends (i) distinct set of usernames (ii) geographical location of IP addresses from where login failure attempts were made and (iii) recurrence of attack sources in the dataset.

I. Distinct Usernames in Failed Login Attempts: From the entire dataset of a month, we counted the number of distinct usernames which appeared in the failed login cases. In all there were 14908 login failure instances (with few being reused) out of which 9044 were distinct usernames. In Table 3.7 we have listed the top 10 such usernames along with number of login attempts and fraction of the time that particular username being used in the total cases. We can notice from the table that "admin" being the most attempted username followed by "pi". In all, the top 10 usernames contribute 12.8% of the failed login attempts with the remaining 13009 usernames contributing 87.2% of the failed logins.

Table 3.7: Top 10 Usernames used in Failed Login Attempts

Usernames	**Failed Attempts**	**Fraction**
admin	1039	0.069
pi	174	0.011
user	146	0.009
test	119	0.007
ubnt	111	0.007
oracle	87	0.005
postgres	62	0.004
support	62	0.004
guest	50	0.003
mysql	49	0.003
Others	**13009**	**0.872**

II. Geographical Location of Failed Login Attempts: In our second trend analysis, we studied the geographical location of failed login sources of different types. For the first case of login failure due to invalid usernames being attempted, there were total of 14908 number of failed instances logged over a period of five weeks. We

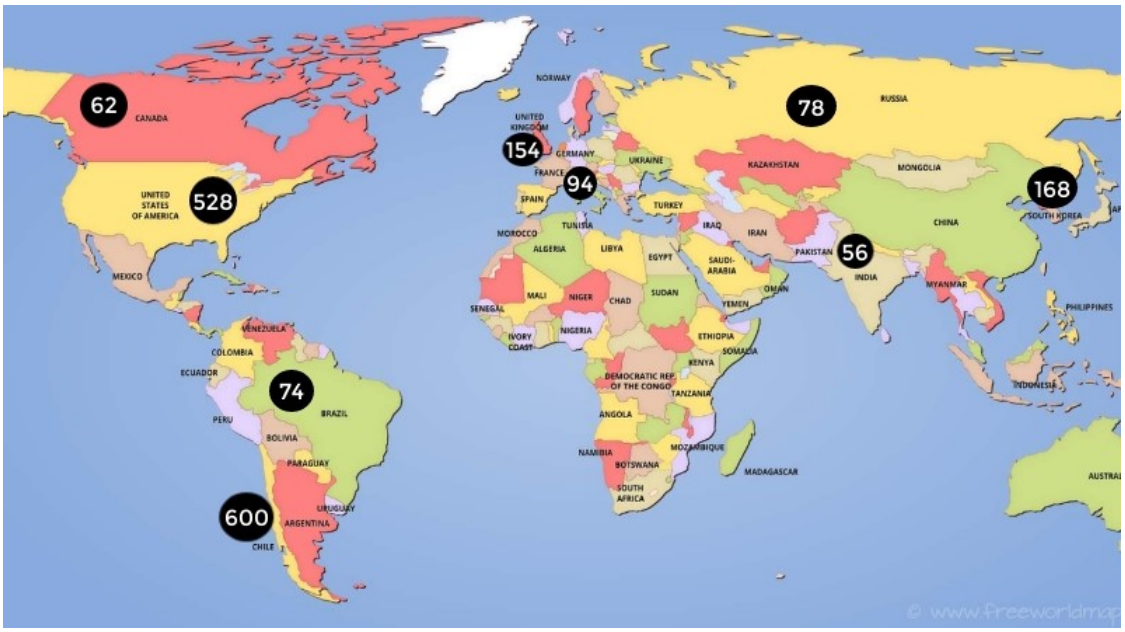

Figure 3.3: Geographic Sources of Failed Login Attempts Due to Invalid Usernames

found geographic location of these source IP addresses using Python-geoip package[1] and lookup(IpAddr) method. In this case (failed login attempts due to invalid usernames) the source IP addresses of failed login sources spanned 76 different countries. In Figure 3.3 we have shown the number of source IP addresses per country for the top 10 countries. We can notice that Chili with 600 source IP addresses tops the list followed by Unite State of America with 528 IP addresses and India being in the tenth position.

III. Recurrence of Failed Login Attempts In this study we find sources of attacks which were repeating over a period of time. We do this in two parts as below.

(i) Recurrence of IP Addresses: In the first case, we analyzed the number of repeating connections (from same IP address) over the period of 5 weeks. Figure 3.4 shows the distinct number of IP addresses per week from which failed login attempts were made using guessed usernames. From the figure it can be seen that in first week there were 725 IP addresses and in the second week there were 854 such IP addresses. Figure 3.5 shows the recurring set of IP addresses over the weeks. Each column in this graph denotes the number of repeating IP addresses which were seen in the sub-

[1]This package in turn uses Goelite2 database created by MaxMind as reference for finding the exact location

sequent weeks. The X-axis in this graph represents the week from which data is taken for intersection and each column indicates the subsequent week(s) data with which recurrence is calculated. For example there are 57 IP addresses from which failed login attempts were made using guessed usernames in Week-2 which were sources of such attacks in Week-1 too (denoted by first column in the Figure). Similarly the second column (green coloured with cross bars) with X-axis label Week-1 and legend Week-3 show there are 46 such overlapping IP addresses. It is easy to notice that similar trend is continued for the remaining weeks too except for the Week-5 which has only 3.5 days data. This study reveals that across the weeks few sources made persistent attempts to connect to the SSH server.

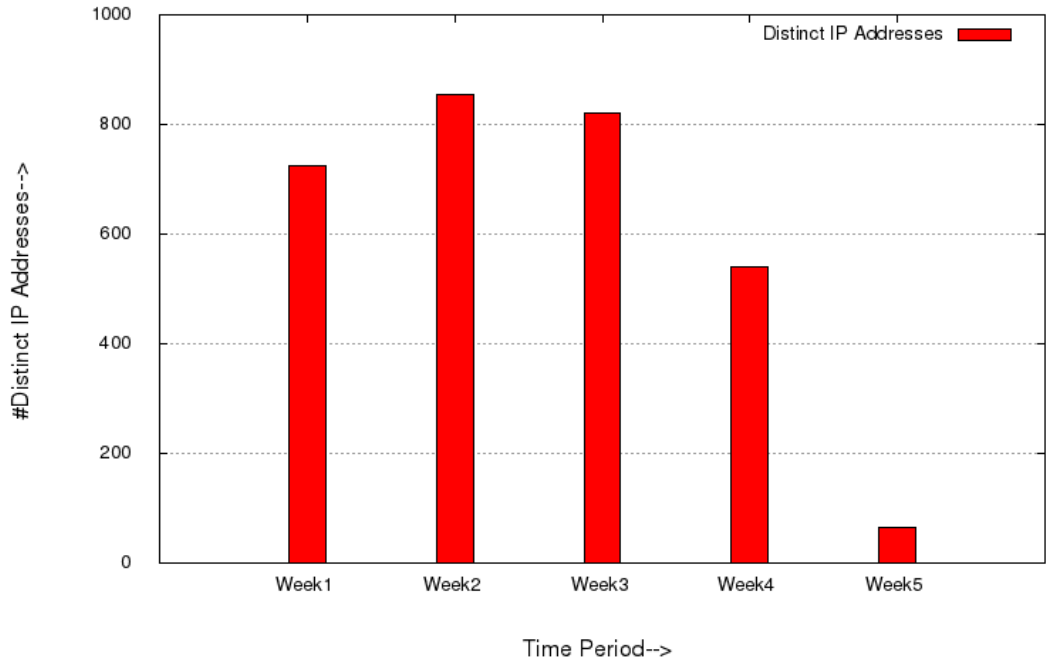

Figure 3.4: Distinct IP Addresses of Failed Login Attempts with Guessed Usernames

(ii) Recurrence of IP Prefixes: Similar to finding recurrent IP addresses, we also found the number of common IP prefixes over the weeks. For example if two IP addresses 10.10.10.7 and 10.10.12.7 are the sources of attacks, they share a common prefix of 16 bit address which in this case is 10.10. As IP addresses can be geographically located understanding the common prefixes will help identifying geographic location of attack sources.

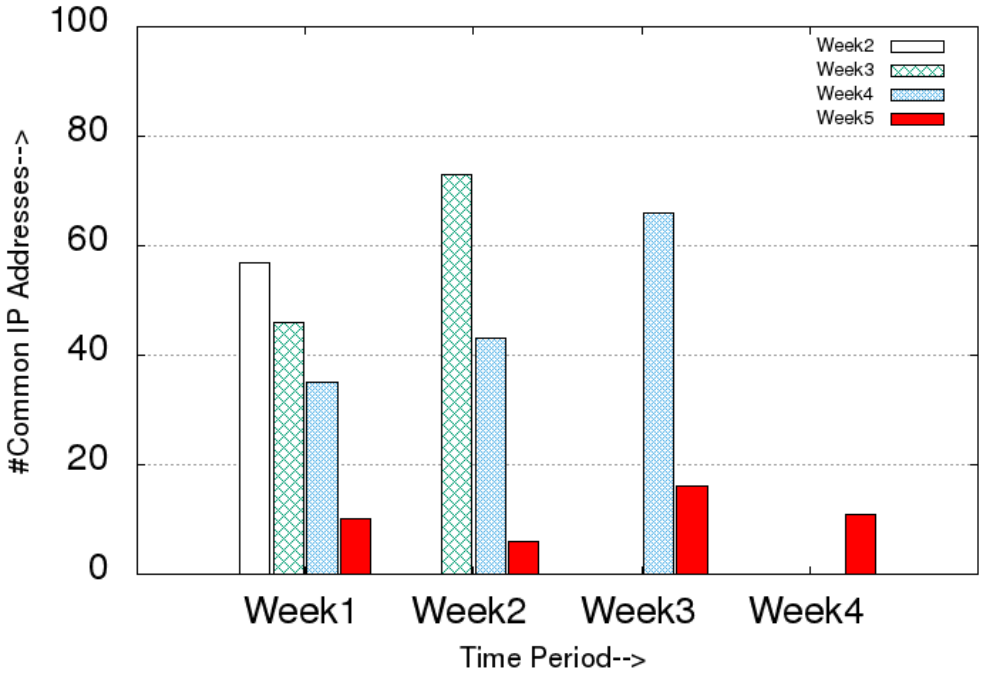

Figure 3.5: Recurrent IP Addresses of Failed Login Attempts with Guessed Usernames

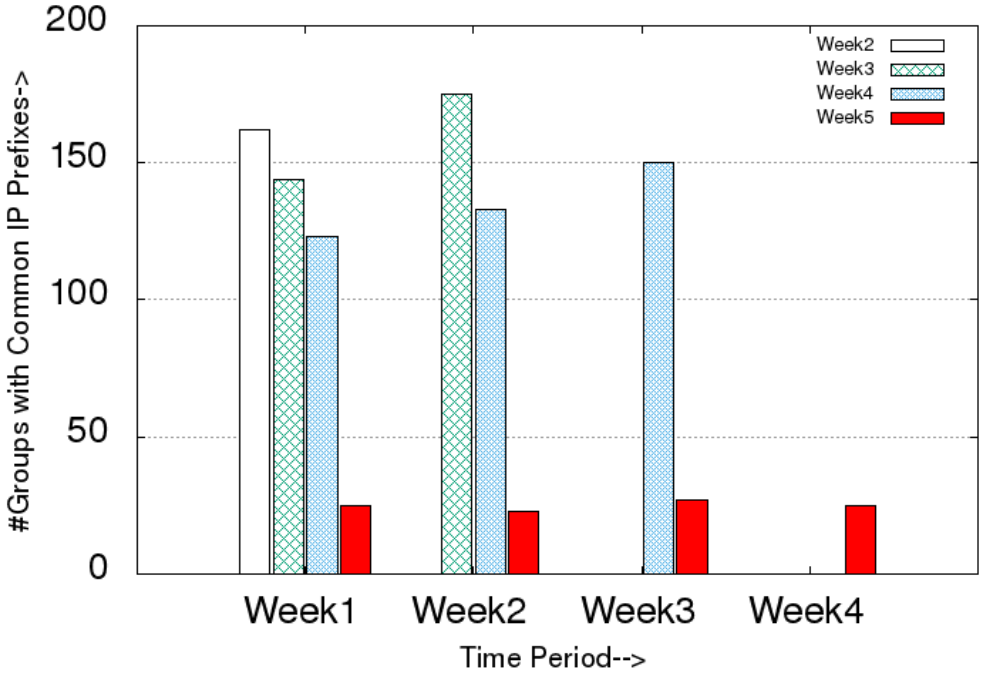

Figure 3.6: Common IP Prefixes of Failed Login Attempts with Guessed Usernames

Figure 3.6 shows the count of such IP prefixes over the period of one month. Labels on X axis represents week number and labels on Y axis represents number of common IP prefixes recurring over weeks. Each column represents the number of recurring groups between the week on X-axis and label legend. For e.g. the very first column (white/transparent bar) denotes the number of IP prefixes which are common between Week-1 (X-axis label) and Week2 (bar legend). Similar interpretation is done for other weeks too. We can notice that indeed there are significant number of such IP prefixes which are repeating across the weeks. In the second week there are 162 distinct 16 bit IP prefixes which were repeating from first week and it can be seen from the figure that similar trend is seen across the weeks.

From above two analysis, we can conclude that there are sources which indeed make persistent attempts to compromise the SSH server spanning several weeks. This trend may be used to secure the SSH server by placing adequate filtering mechanisms either in firewall or using tools which block hosts/prefixes after a threshold number of failed attempts.

3.5 Identifying Failed SSH Connections

The log analysis part presented in the last section revealed that attackers use several techniques to compromise the systems and attacks are prevalent against SSH server. As discussed in the Chapter 1 that detecting attacks using the logs is not scalabale as it requires access to log files from every system. The other alternative is to use network level data of packets and flows to detect such cases. In this section we describe a method to identify failed and successful login cases using network level data. Our detection method uses network flows which can be generated using traffic or can be in the form of records exported by routers. For e.g records can be netflow records. Few previous works [20, 28, 9, 41] have reported that flow based detection is not only effective; but it is also lightweight. In addition, as this is a network based detection, it does not require access to system level logs from individual machines. In

the next three subsections we elaborate on the details of Dataset-II, detection method, and results of experiments with our detection method.

I. Dataset for Experiments For our experiments with SSH flow classifier we use the Dataset-II which has raw network traffic collected with tcpdump tool [40]. For the ground truth, we cross verified the log files generated in the same period for timing and success of the login attempt.

II. Flow Characteristics of Failed Login Connections A flow is a bidirectional sequence of packets exchanged between the two end points. It can be a TCP flow or an UDP flow. In order to differentiate the flows corresponding to failed logins and successful login cases, we use the characteristics of network flows. In particular, we notice that if the server is using only non keystroke (public key) based authentication, any attempt to login with credentials are immediately rejected. Further key mismatch and username mismatch will also result into immediate closure of such connections. We can make following observations in this case.

(i) If the authentication is unsuccessful and connection is closed immediately, it results in very few bytes of data exchange between the SSH server and client.

(ii) Immediate connection closure also indicate that the flow has a very small duration. However, flows corresponding to successful connections will have large duration.

(iii) Small duration of connection and small number of bytes exchanged normally also results in small number of packets being exchanged.

We exploit these observations and identify three parameters listed in Table 3.8 to differentiate successful and unsuccessful SSH connections.

Table 3.8: Flow Level Features

Feature	**Description**
f_1	Number of Packets in the Flow
f_2	Number of Bytes Transmitted in the Flow
f_3	Duration of the Connection

In order to identify failed login cases we generate feature vectors with ground truth labels and use machine learning algorithms for classifying them. Detecting failed login attempts is the first step in detecting bruteforce attacks. A simple method

is to aggregate failure cases in an interval time period and use a threshold to declare the interval time has experienced any attack or not.

To begin with we show that, the three parameters chosen are useful for detecting the flows corresponding to failed SSH connections at the network level. For this we used 50 flows of successful and failed connections as a sample and draw comparison graphs. Figures 3.7, 3.8 and 3.9 show the different values of these parameters. In these three figures, the X axis show the flow number (1-50) and Y axis has the respective parameter values in Log scale. We can notice that, these figures confirm the observations made above.

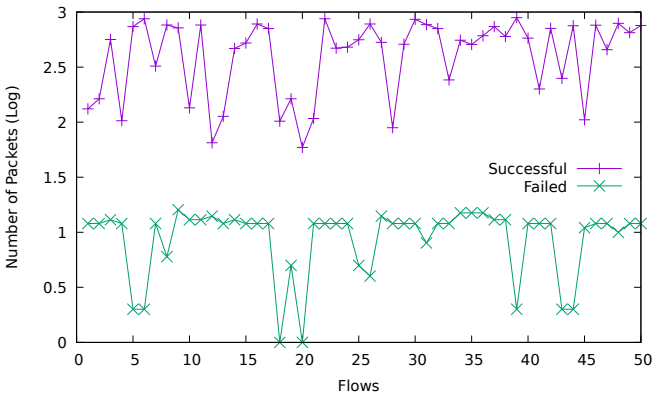

Figure 3.7: Number of Packets Per Flow for Successful and Failed SSH Connections

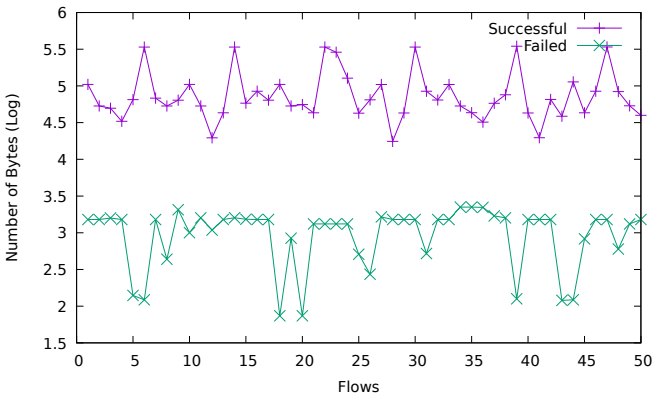

Figure 3.8: Number of Bytes Per Flow for Successful and Failed SSH Connections

III. Evaluation Results We processed the Dataset-II of network packets with a python script to reconstruct flows. What we observed is that the number of failed login cases in this period of one month was considerably larger than the number of

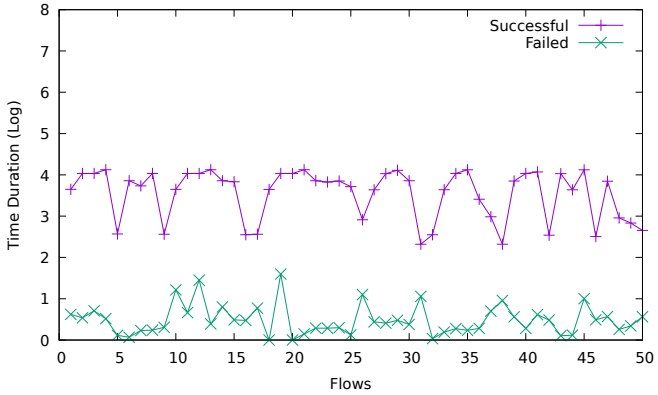

Figure 3.9: Time Duration Per Flow for Successful and Failed SSH Connections

successful login attempts. This results in uneven sample distribution when used for training with machine learning algorithms. This can adversely affect their learning mechanism. Thus we used all flows corresponding to successful logins (found with ground truth using authentication logs) for the entire month and approximately 4 times those many number of failed connection flows. Table 3.9 shows the details of the types of flows.

Table 3.9: Dataset Characteristics

Type of Flows	**Number of Flows**
Success	481
Failure	1914

In order to identify failed and successful login attempts, we begin with extracting per-flow based features to generate the feature vector. Each network flow is encoded into a feature vector containing the number of packets per flow, total bytes per-flow, and flow duration as mentioned previously. In the next step, these feature vectors are provided as an input to machine learning algorithms to classify SSH flows as successful or failed login attempts.

We experimented with a bunch of well known supervised machine learning algorithms namely K Nearest Neighbor, Naive Bayes, Random Forest and Random Tree. A brief description of these algorithms follows

(i) `K Nearest Neighbor`: K-nearest neighbor is a supervised machine learning algorithm used for classification and regression. The training dataset contains feature

vectors in multidimensional feature space, each assigned with a class label. Feature vectors, along with class labels, are called training samples. The training phase of the algorithm only involves storing the training samples. The next step and the algorithm's outcome depend on the problem the algorithm is applied for, i.e., classification or regression. The K-NN classification outcome is a class membership resulting in a discrete value called a class label. A test sample is assigned a class label closest among its K-neighbors, where K is the value defined by the user. The outcome of a K-NN regression is real number data representing the property value of the sample object. In K-NN regression, the result is the average value of its K-nearest neighbors.

(ii) `Naive Bayes`: Naive Bayes is also a supervised machine learning algorithm used for classification problems. Naive Bayes classifiers use a collection of classification algorithms developed using Bayes theorem hence named Naive Bayes. The algorithm works based on two common assumptions. First, each feature in a given vector is independent of the other given a class label. Second, each feature makes an equal contribution to the outcome. Thus, it assures that no feature is irrelevant and any combination of features less than n cannot predict the outcome correctly. For a problem instance with a vector of n features, the classifier predicts the class label from a given finite set of classes. Naive Bayes classifier is designed based on probabilistic models using conditional probabilities for outcome prediction. Using Bayes theorem, conditional probability is estimated as

$$p(Y_K|X) = \frac{p(Y_K).p(X|Y_K)}{p(X)} \tag{3.1}$$

Where a problem instance to be classified, is represend as a vector $X = (x_1, x_2, \cdots x_n)$ representing n independent features.

(iii) `Random Forest`: Random Forest is also a supervised machine learning algorithm that combines multiple decision trees for outcome prediction. A decision tree is the fundamental building block of the random forest classifier. A tree is built by deciding(based on a feature) at each node that splits the tree further down to a more accurate prediction eliminating the wrong possibilities at each step hence the

name decision tree. Each decision tree predicts a class label, and the outcome of the random forest classifier is estimated collectively based on the majority of the class label predicted by each decision tree. A problem instance is assigned a class label that wins by the majority of trees in a forest. Random forest classifiers perform well when the decision trees have a low correlation between them and the meaningful features are selected while building decision trees. Low correlation between the trees protects each other from their individual errors, minimizing the random forest's overall error, resulting in a more accurate prediction

(iv) `Random Tree`: The random tree is a machine learning algorithm that uses a random tree classifier for classification. The classifier works by splitting the dataset into a smaller subset with different features in each subgroup. This process creates different models of a given dataset. Next, decision trees are built using these sub-sampled data that predicts one class label. Subsequently, the outcome from each decision tree is aggregated to predict the outcome/class label for a given instance. Finally, a label to the problem instance is assigned by majority voting of the class label from all decision trees.

We used the feature vectors generated with three flow parameters and the Weka tool [42] for the evaluation. In all the experiments we used 10 fold cross validation method. The experimental result (confusion matrix) for each algorithm listed above is shown in Table 3.10, Table 3.11, Table 3.12 and Table 3.13 respectively. We can notice that most of the algorithms showed very minimal number of misclassifications. These experimental results demonstrate that flow-based features serve as a good candidates to differentiate successful and failed login instances.

Table 3.10: Classification Performance of KNN Classifier

	Success	**Failure**
Success	480	1
Failure	2	1912

Table 3.11: Classification Performance of Naive Bayes Classifier

	Success	**Failure**
Success	479	2
Failure	2	1912

Table 3.12: Classification Performance of Random Forest Classifier

	Success	**Failure**
Success	474	7
Failure	0	1914

Table 3.13: Classification Performance of Random Tree Classifier

	Success	**Failure**
Success	477	4
Failure	0	1914

3.6 Conclusion

In this chapter we presented a case study using SSH logs collected from a production level server to reveal that attackers use various methods to break into the system, a handful of usernames are attempted repeatedly and few sources make persistent attempts to break-in. Further the origin of attacks are geographically well spread with few countries having a significant share. We also presented a method to identify successful and failed login cases by analyzing network flow characteristics with machine learning algorithms. These algorithms showed very high accuracy of classification in our experiments.

Chapter 4

SSH Bruteforce Attack Detection with Probability Distribution of Failed Logins

4.1 Introduction

Bruteforcing is a common attack against SSH servers where attacker tries multiple combination of usernames and passwords in an attempt to login to a server. This was evident from the case study we presented in the previous chapter. These attacks are usually executed either with a dictionary of words or by systematically generating username and password combinations. Consequences of a compromised system can be severe. The detection methods usually appear in the form of x number of failed connections in y time from a source will signal an attack. These methods usually have thresholds to detect attacks. An attacker can respond to such detection methods by distributing the load to multiple systems [9]. If the attack sources are distributed, the IP address and port parameters will change and hence this poses serious challenge in detecting such attacks. As mentioned in Chapter 1 and evaluated in Chapter 3 network flow based parameters help in identifying failed connections which can be used to detect bruteforce attacks. The flow based statistics like packets per flow,

bytes per flow and connection duration will help minimize the false alarm rate by reliably estimating the connection status.

In this chapter we propose a method to detect SSH bruteforce attacks using network flow characteristics and representing the failed login attempts as probability distribution model. Our proposed method first identify a flow as originated from SSH application and subsequently classify it to be either successful login or failed login attempt. Our contributions in this chapter are as follows.

- We propose a method to identify SSH flows using flow content similarity. It uses few initial bytes extracted from the flows for this similarity estimation.
- We adopt the flow parameters described in Chapter 3 for detecting flows corresponding to failed SSH connections. This is done by maintaining simple statistics of connection parameters.
- We model the failed login connections as a probability distribution and identify events corresponding to low probability as bruteforce attacks.
- We experiment with an attack dataset and show that the proposed method is effective in detecting such SSH bruteforce attacks.

Remaining part of this chapter is organized as follows. In Section 4.2 we briefly revisit the related literature on detecting SSH bruteforce attacks. In Section 4.3, we describe the proposed flow based bruteforce detection method. In the Section 4.4 we elaborate on the experiments done to evaluate the proposed method. We conclude this chapter with a discussion in Section 4.5.

4.2 Prior Work

Existing literature related to SSH attacks detection are of following two types:
(i) Bruteforce Detection and Prevention: Threshold based detection methods are very commonly used to identify bruteforce attacks. Tools like DenyHosts [14], BruteForce-Blocker [13], fail2ban [6], and sshguard [7] belong to this category.

Another work by Tian et al. [43] slowly filters the number of connection attempts if a source generate more number of failed login attempts. Few recent works have used flow statistics [9, 41, 44] to detect SSH compromises. Hellemon et al. [20] and Hofstede et al. [9] use two features packets-per-flow (PPF) and the number of flow records per minute with thresholds to detect attacks. Hofstede et al. [44] use packet payload size within the flow as metric to identify successful compromises. Javed and Paxon [17] used statistical change point detection to identify distributed and stealthy SSH bruteforce attacks.

(ii) Case Studies on SSH Bruteforcing: Romain and Vivien [30] used honeypots to collect a SSH bruteforce attack dataset. Their observation with the collected dataset revealed that many attacks recur over a period of time. Another recent study [32] on SSH logs collected with honeynet found the country of origin of attack, number of attempts made, etc. CAUDIT [33] is another case study which used observations made in honeypot systems to set filter for the real SSH servers.

As mentioned in the Chapter 1 due to scalability problem, network based attack detection methods are usually used [45]. This is also evident from the above literature that network traffic monitoring techniques are the recent phenomenon. Taking motivation from these works we propose a method for detecting distributed SSH attacks in this chapter.

4.3 SSH Bruteforce Attack Detection

In this section we describe the proposed bruteforce attack detection method. Our detection method works in three phases as

(i) Identifying flows corresponding to SSH connections

(ii) Identifying failed SSH connections through flow parameters

(iii) Detecting attacks using probability distribution of failed login connections.

We describe these three phases in the next three subsections.

4.3.1 Identifying Flows Corresponding to SSH Connections

First phase of our detection method is to identify flows belonging to SSH application. There are two ways in which SSH flows can be identified.

(I) Using Port Number: A simple method to identify flows corresponding to SSH connections is to rely on port number. As SSH typically runs on port number 22, any flow communicating to port 22 is identified as of SSH application. Few standard applications have default port numbers. However it is possible to run these applications on different port numbers. For. e.g. many web servers do not run on port number 80 which is its default port number.

(II) Using Flow Content: Second method is to use flow contents in addition to port number. This is done by measuring the flow content similarity between a known reference SSH flow and subsequent flows having connection to the port number on which SSH is running. The rational behind this approach is that, SSH protocol has an initial handshaking phase where client and server negotiate a set of parameters. These include cipher suit to be used, message authentication (MAC) and compression algorithms to be used [28]. The cipher protocols may be AES-CBC, 3DES-CBC and MAC algorithms may be HMAC-MD5, HMAC-SHA1, etc. These algorithms are listed in the initial handshake in plain text format. Thus, the content from this portion of flow can be used to find the similarity between the flows in order to confirm that flow is indeed a genuine SSH connection negotiation. Figure 4.1. shows the contents of a packet in SSH negotiation as seen in the wireshark. We can notice that these protocol names are carried as part of the payload. SSH flow identification using content similarity estimation is based on Deep Packet Inspection technique and it involves three phases as below.

(i) Flow Reconstruction: First step is to group packets part of a flow. This is done by flow reconstruction by identifying packets between TCP connection establishment to termination, sharing common IP address and port number details. SSH uses TCP as transport layer protocol and a TCP connection begins with a three-way handshake as below.

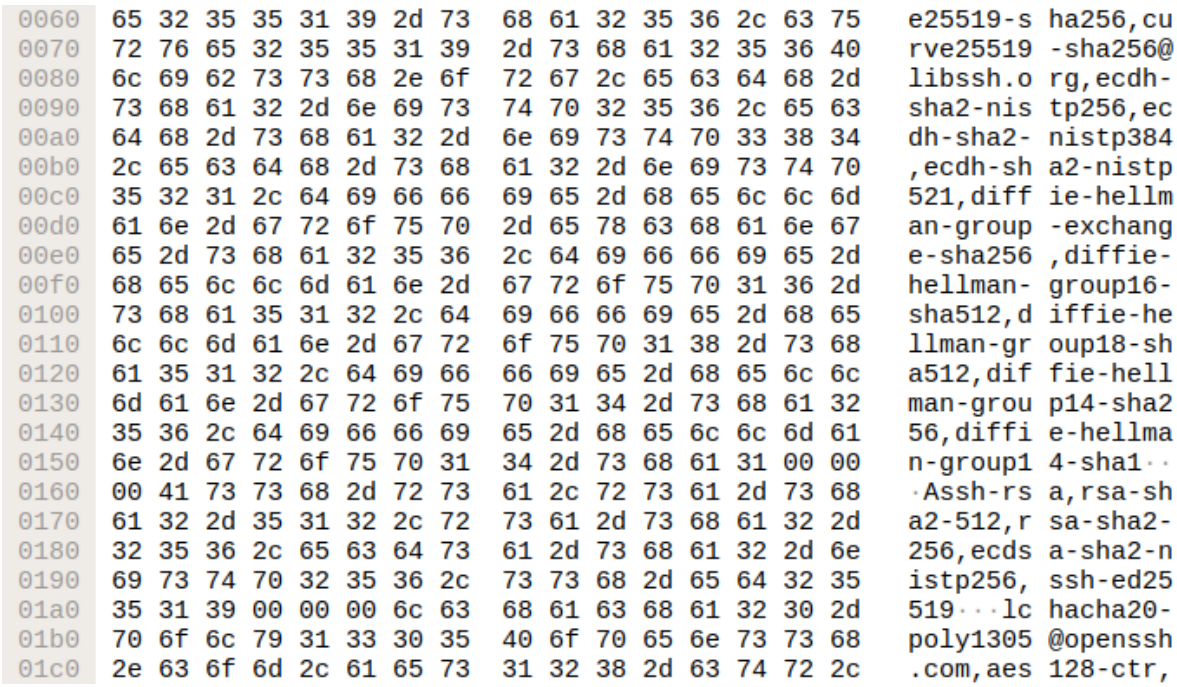

Figure 4.1: Sample Payload of a Packet in a SSH Flow

1) (X) → [SYN] → (Y)

2) (X) ← [SYN/ACK] ← (Y)

3) (X) → [ACK] → (Y)

The TCP connection closes with a fourway handshake as shown below

1) (X) → [FIN] → (Y)

2) (X) ← [ACK] ← (Y)

3) (X) ← [FIN] ← (Y)

4) (X) → [ACK] → (Y)

All packets between connection establishment to closure are typically part of one flow (there are variants exists). Subsequent to flow reconstruction, we process initial few bytes of flow content and parse it to generate tokens from the flow. Using these bytes we find the similarity between the flow in question and the reference flow. First few bytes are relevant to be used as handshaking happens only in the beginning. Subsequently when the client and server agree on using a crypto suit, they derive a session key. Once the session key is derived, all the messages exchanged between client and server are encrypted with that session key.

(ii) Token Generation: Once the bytes which are in plain text format and necessary for comparison are extracted, we generate tokens from this payload. We use few characters (like , /, whitespace, -, etc) as delimiters to generate tokens. From every flow payload (concatenation of all packet payload within a flow) a set of tokens are generated as $T = \{t_1, t_2, \cdots, t_n\}$. Few example tokens generated from the example Figure 4.1 are "sha2", "sha256".

(iii) Flow Similarity Estimation: Once the tokens are generated from the payload under consideration, next step is to estimate the similarity between the reference SSH flow and the flow in question. For this comparison, we take the reference SSH flow and use tokens from this flow. The tokens generated from the flow in question are compared with the tokens of reference flow to find the similarity. Flows which are found to be similar are labeled as belonging to SSH application. Flow similarity is estimated by finding the overlap between the tokens. Let $T_R = \{tr_1, tr_2, \cdots, tr_n\}$ be the set of tokens from reference SSH flow and $T_Q = \{tq_1, tq_2, \cdots, tq_m\}$ be the tokens from flow in question, then similarity between these two flows are estimated as in Equation 4.1. Equation 4.1 is the ratio of number of tokens common between them to the sum of the token in them minus the size of common token set.

$$SIM(T_R, T_Q) = \frac{|T_R \cap T_Q|}{|T_R| + |T_Q| - |T_R \cap T_Q|} \tag{4.1}$$

The flow in question T_Q is declared as SSH flow if the $SIM(T_R, T_Q) \geqslant \delta$.

4.3.2 Detecting Failed Logins

We consider the flows which are found to be similar to the reference SSH flow for further verification as belonging to either successful or failure SSH connections. As shown in the Chapter 3, the characteristics of flows representing unsuccessful and successful login cases are different. In general, flows belonging to unsuccessful login attempts have less number of packets exchanged, less number of bytes transmitted and are short in duration. The three features f_1, f_2 and f_3 correspond to these parameters. Using these parameters, we can determine the flows belonging to successful

or unsuccessful SSH connections.

(i) Using Machine Learning Algorithms: As shown in the Chapter 3, we can train the machine learning algorithms with the feature vectors generated with these parameter values along with labels to classify unknown flows. However, machine learning algorithms can be often expensive to train and implement. Here we define another simpler way to identify the flows corresponding to failed SSH connections.

(ii) Using Weighted Average of Parameters: In order to identify failed logins, we use the weighted average of parameters chosen i.e., average number of packets in a flow (f_1), average number of bytes transmitted in a flow (f_2), and connection duration (f_3) as shown in Equation 4.2.

$$WtAvg = 0.33 \times f_1 + 0.33 \times f_2 + 0.34 \times f_3 \tag{4.2}$$

Let F_1, F_2,... ,F_n be the flows of successful SSH connections. We calculate the weighted averages $WtAvg_{F_1}$, $WtAvg_{F_2}$, ... $WtAvg_{F_n}$ for these flows. Using these n weighted averages of flows corresponding to successful connections, we calculate the mean and standard deviations as in Equation 4.3 and Equation 4.4 respectively.

$$FlowMean = \frac{\sum_{i=1}^{n} WtAvg_{F_i}}{n} \tag{4.3}$$

$$\sigma_1 = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(WtAvg_{F_i} - FlowMean)^2} \tag{4.4}$$

We set the threshold on these parameters as in Equation 4.5 and any flow which has a value smaller than this threshold on the weighted average of these parameters are considered as failure instances.

$$Threshold(\tau_1) = FlowMean + K \times \sigma_1 \tag{4.5}$$

Here K is a positive constant and σ_1 is the standard deviation on the number of failed logins across the intervals. Any flow F_l whose weighted flow parameter $WtAvg_{F_l}$ is lesser than this $Threshold$ is considered as failed login.

4.3.3 Detecting Bruteforcing Attacks

Login failures happen under two cases, (a) when a legitimate user forgets the password or mistypes the password (b) when bruteforcing is attempted. In order to separate the second class from first set of cases, we model the failure cases as a probability distribution. In particular, we consider the mean number of failed SSH connections in a time interval w and model it as a Poisson probability distribution. Let λ denote mean number of login failures in an interval from known non bruteforcing intervals, then probability distribution function of this is given by Equation 4.6.

$$f(n, \lambda) = \frac{\lambda^n \times e^{-\lambda}}{n!} \tag{4.6}$$

In this equation, n is a value of failed logins for which probability is being calculated and e is a constant. This function will have its peak value around the mean and on either side it decreases. An example graph with mean value $\lambda = 5$ is shown in Figure 4.2.

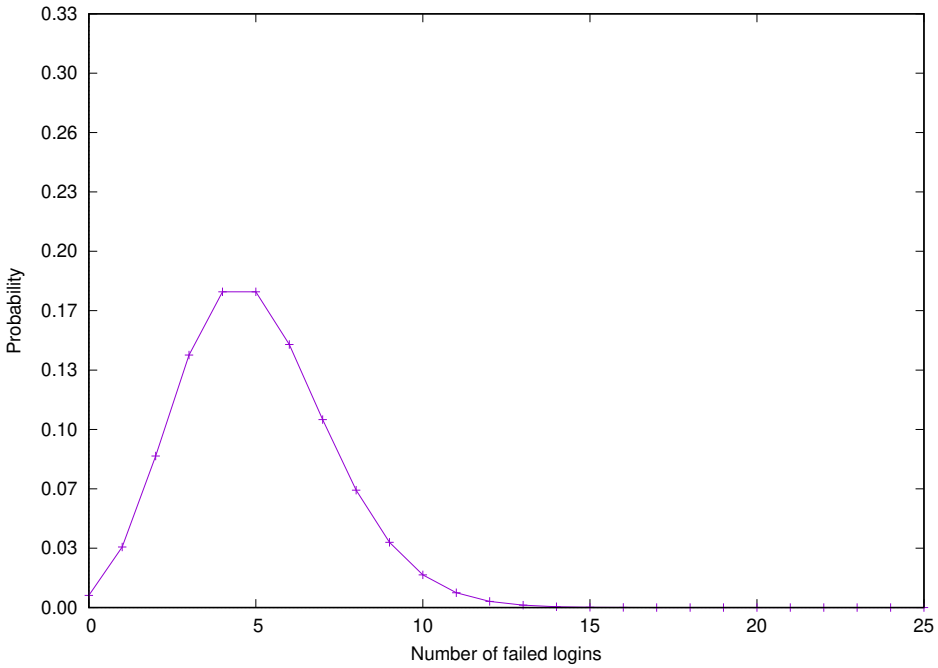

Figure 4.2: Poisson Probability Distribution with λ=5

Poisson probability distribution model assumes the independence of events. In

our context, it implies that there is no relationship/interdependence among the failed logins. As the model is generated using the mean number of failed login cases of known non bruteforce intervals, this assumption is valid. The only kind of failures are of first category where users either forgot their passwords or misspell them. In this case, one user misspelling or forgetting the password has no bearing on other users. Thus the independence of events required for Poisson probability distribution holds good.
In brutefrocing attacks this assumption will not hold as an adversary is generating many failed requests and all of them are coming from a source. This will increase the number of failed login cases. Thus, to detect bruteforcing attempts we identify low probability events in the distribution. To identify such low probability events, we use the one-sided Chebyshev inequality shown in Equation 4.7.

$$P(X \geqslant \lambda + c) \leqslant \frac{\sigma_2^2}{\sigma_2^2 + c^2} \tag{4.7}$$

In this equation X is a random variable with mean λ and standard deviation σ_2, and $c > 0$ is a positive constant. For Poisson distribution the value of σ_2 is estimated as in Equation 4.8.

$$\sigma_2 = \sqrt{\lambda} \tag{4.8}$$

The Equation 4.7 help us get an upper bound for the sum of probabilities for X greater than $\lambda + c$. A low sum of probabilities indicates collection of very rare events (likelihood of those many failed logins). A threshold (τ_2) on the number of failed login attempts in the tail of distribution will help detect the attacks. In our case we set this value to $\beta \times \sigma_2$, i.e., $P(X = \lambda + c) = \beta \times \sigma_2$. For the sample distribution shown in Figure 4.2 and for the values of c=12, $\beta = 7.602$, and $\sigma_2 = 2.236$ the threshold value is shown in Figure 4.3 with the vertical red line.

4.4 Experiments

In order to evaluate the performance of the proposed approach of detecting SSH bruteforce attacks we performed an experiment. The details of datasets used and

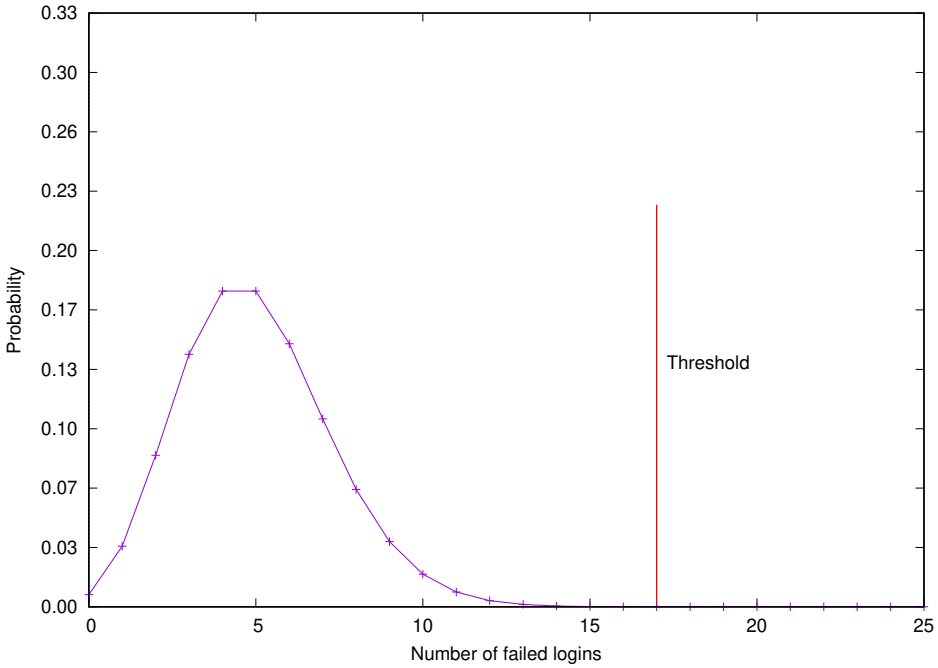

Figure 4.3: Poisson Probability Distribution with λ=5

detection performance of proposed method are described below.

4.4.1 Datasets

For this experiment we used two datasets, first one is taken from a SSH server deployed in our university network and second one was collected from testbud setup in our Lab.

(i) SSH Traffic Data from Production Level Server: Our university network has an SSH server used by many internal users. This server is primarily utilized for running high-performance computing applications. This server is located in the internal server farm of Figure 3.1. The idea is to use the failed login attempts from this server as a reference to detect bruteforce attacks by modeling it as a Poisson probability distribution. We collected SSH network traffic from this server for a period of 45 hours using tcpdump[40] tool. We used this dataset for setting the threshold for bruteforce detection.

(ii)SSH Traffic Data Generated from Testbed: We set up an SSH server on

Table 4.1: Dataset Statistics of Normal Intervals

Type	Interval	Total Flow Count	Successful Connection Flows				Failed Connection Flows			
			Flow Count	Avg Pkt Count	Avg Duration(s)	Avg Bytes	Flow Count	Avg Pkt Count	Avg Duration(s)	Avg Bytes
Institute-Part1	48	962	68	576.544	6219.899	106412.132	894	11.543	133.222	1691.709
Institute-Part2	42	546	42	483.904	2265.776	84531.666	504	10.521	75.481	1389.087

a Ubuntu Machine in our laboratory with 10 valid users. We generated bruteforce attacks against this server by generating requests using Python scripts. To generate bruteforce attack flows, we wrote a script that made successive login attempts using randomly generated text as passwords for valid usernames. We executed this attack for 36 hours continuously. Using tcpdump [40] tool, we collected login data at the server and used it for testing the attacks.

Table 4.2: Dataset Statistics of SSH Bruteforce Attacks

Type of Flows	Intervals	Flow Count	Avg Failed Login Count	Avg Duration(s)	Avg Pkt Count	Avg Bytes
Attack	72	55337	768.569	2.135	14.813	2114.36

4.4.2 Detection Performance

Deriving Parameters: We divided the datasets from known no bruteforce attack intervals into two parts for our experiments. The first part was used for setting the threshold for differentiating successful and failed SSH flows using the flow characteristics. The second portion of this data and the attack dataset are used for evaluating the detection performance. Table 4.1 shows the characteristics of flows collected from known no bruteforce attack intervals in terms of the number of flows of successful and failed login attempts, the average number of packets, bytes in the flow, and also the average duration of a connection. Though the Table 4.1 shows the statistics of no attack intervals, it can be seen that there are a significant number of failed login connections for institute server traffic. These failed login connections are from two sources. The first source of such cases is internal users, and the second category of requests is from external sources. The failed connections from internal users are primarily caused by genuine users who have forgotten their login credentials and attempt incorrect passwords, resulting in failed connection attempts. We manually examined the external connection source IP addresses, and found that

many of them had no similarity, with the exception of one IP series with the prefix 222.186.*.*. In the Institute-Part1 dataset, there were 34 such connection requests from this prefix with eight different IP addresses (5 connections in Institute-Part2). The number of packets and bytes transmitted were lower in these flows, but the connection duration was unusually large. These are the connections that influence the average connection length of failed login cases, as seen in Table 4.1 in the $10th$ column of the first and second rows. Similarly, the parameters of the bruteforce attack dataset are shown in Table 4.2. We used the known non attack data from production level server and calculated the mean number of failed login attempts in an interval window of 30 minutes and it turned out to be 19 (approximated to nearest integer) for a week data. The Poisson probability distribution with this failure rate is shown in Figure 4.4. We use this probability distribution graph as reference to detect rare probability events as attacks. We also calculated the normalized weighted average of all the three features (duration of flow, number of packets and number of bytes transmitted) of login attempts found in this dataset. Using these values of failed login attempts and threshold value (τ_1), we label a test flow as success or failure as in Equation 4.5.

Evaluation: We used the data generated from python script with bruteforce attack for testing purpose by dividing it into 30 minutes interval time resulting in total of 72 such intervals to be labeled. A reference SSH flow was used for classifying subsequent testing flows with a similarity score value of 0.8. With this similarity score, all test SSH flows were identified as SSH. We evaluated the detection performance of our proposed approach for different values of threshold τ_1 and τ_2. We empirically found the performance variance for these values and chose the best combination yielding a higher detection rate. Table 4.3 shows the performance results for the best combination of values (shown in the table) for the experiment where Institute-Part1 data was used for deriving thresholds and the Part2 along with the attack data used for testing. We can see that all the 72 intervals of test attack data were classified as attack cases, yielding a 100% detection rate for the given threshold values of $\tau 1$ and $\tau 2$. On the

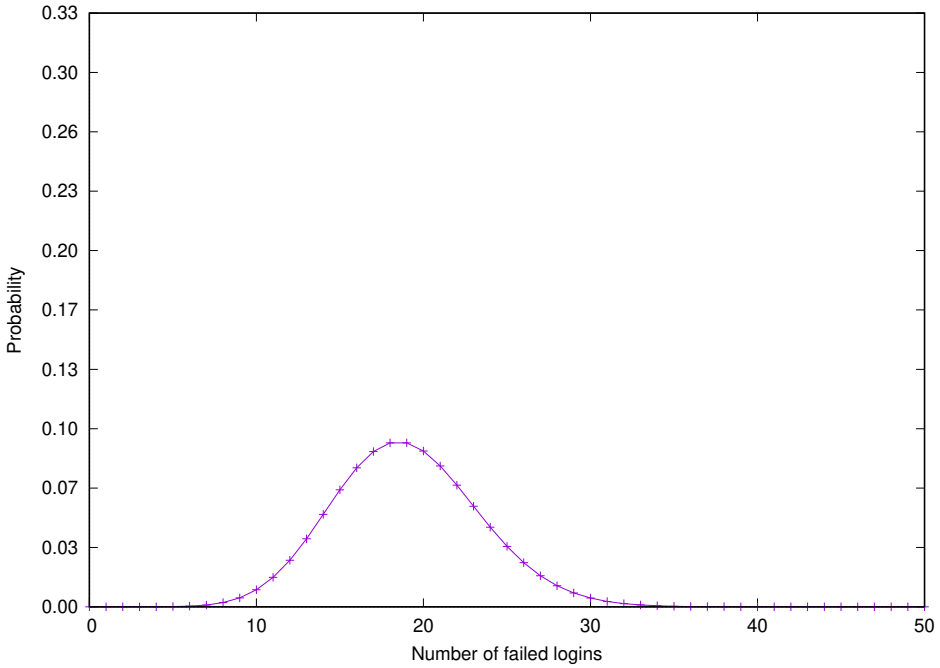

Figure 4.4: Probability Distribution of Failed Logins

Table 4.3: Bruteforce Attack Detection Performance of Proposed Approach

	Parameters τ_1 =1514 τ_2 =739	
	DR	FP
Attack	100	0.00

other hand, none of the intervals of server data(Part-2) were identified as attack cases for the given threshold values.

4.4.3 Sensitivity Analysis

The detection performance of our approach is dependant on the threshold values τ_1 and τ_2. In order to understand the variation in the detection performance with these values, we did another set of experiments. In this experiment, we estimated the detection performance variation with these two parameters used, namely τ_1 and τ_2. Here, we collected two sets of readings by setting one parameter constant and varying another parameter in step size of one. In each case we calculated the detection rate. Figures 4.5 and 4.6 show the detection rate variation with these three parameters.

From Figure 4.5 we can see that as threshold τ_1 increases, the bruteforce attack detection rate also increases. This is because, as threshold τ_1 increases, it implies that threshold on the weighted average of parameters for a flow to qualify for failed login also increases as shown in Equation 4.5. So the number of flows having the weighted average of parameters less than this threshold value also increase[1]. In other words, the number of flows classified as failure instances in an interval increases. So an increase in the number of failed connections crosses the threshold τ_2 to be detected as an attack case.

Similarly from Figure 4.6, we can see that as threshold τ_2 increases, the bruteforce attack detection rate decreases. This is because, with an increased threshold value τ_2, the number of failed logins required for declaring an interval as attack increases. This also means that at least those many numbers of failed logins are required for classifying it as an attack. From the given figure, we can conclude that with increase in failure threshold τ_1, detection rate increases. Whereas, with increase in attack threshold τ_2, the detection rate decreases.

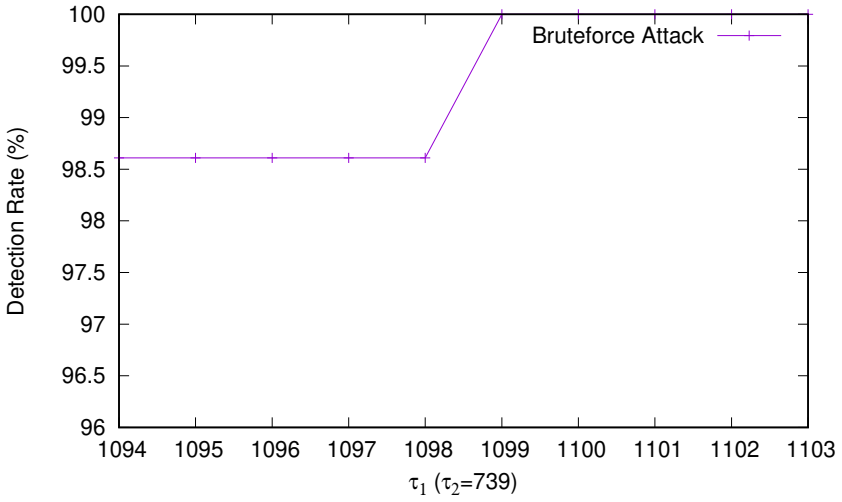

Figure 4.5: Bruteforce Attack Detection Rate for Different Values of Threshold τ_1

[1]A flow is detected as corresponding to a failed login if its average flow parameters are less than τ_1

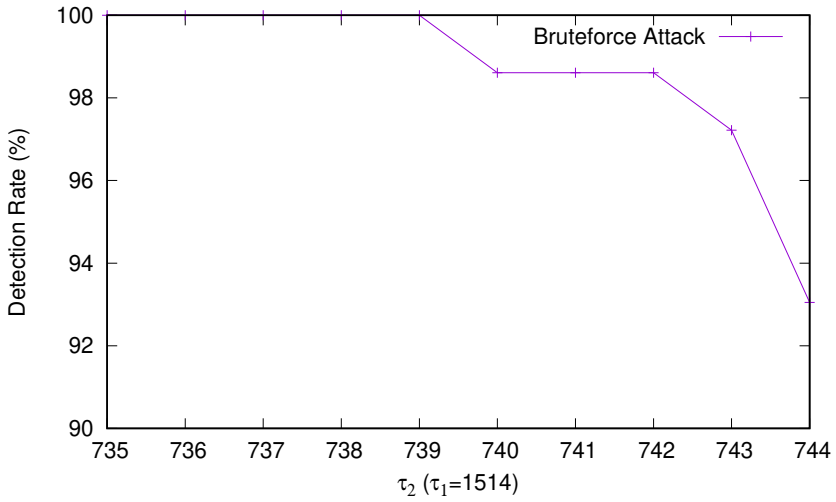

Figure 4.6: Bruteforce Attack Detection Rate for Different Values of Threshold τ_2

4.5 Conclusion

In this chapter, we proposed a method to detect bruteforce attacks by modeling failed login attempts as Poisson probability distribution. We use content similarity between flows to classify flows as generated by SSH application and these are subsequently used for evaluation. We use flow characteristics of failed logins and count the number of such events in a window period to label the corresponding window time as either normal or having bruteforce attempts.

Chapter 5

SSH Bruteforce Attack Detection and Classification with Petri-Net Modeling

5.1 Introduction

Bruteforce detection method described in the Chapter 4 is capable of detecting attacks against SSH servers. It uses network flows and their characteristics for detecting the attacks. The probability distribution model developed in Chapter 4 is limited to identifying whether there was an attack against server. As discussed in previous chapters, an attacker can try evading detection by keeping low profile about the activities and also launching the attacks from several sources. Individually these sources may or many not be aggressively generating login attempts. In general, these attacks are hard to detect and even harder to filter. We find one attempt by Javed and Paxon [17] in the literature who proposed a method to detect stealthy SSH attacks using change point detection algorithm. Detecting a stealthy and low profile attacks is a challenging problem in cyber security research. Such low profile attacks are also common in port scanning [18], web applications [46].

We argue that the limited SSH bruteforcing detection methods found in the litera-

ture fall short of meeting the practical requirements. In this chapter, we build on the idea described in Chapter 4 to detect coordinated SSH bruteforce attacks. We identify that the coordinated attacks can be of three types and propose methods for differentiating various kinds of attack scenarios. Further, we identify and list the sources of such attacks to enable easy and quick filtering by automating it. In particular our contributions in this chapter are

- Similar to the previous chapter, we use flow level information to decide whether an SSH flow corresponds to successful or unsuccessful login attempt.
- We develop a Colored Petri-Net based formal model to detect and subsequently classify bruteforce attacks into three categories as originating from single source, single domain and/or distributed sources.
- We evaluate the proposed model with datasets collected from production level SSH server and also from simulation setup and show that it is not only effective in identifying attacks but also type and contributors.
- We show that proposed model can be used to detect stealthy and low profile attacks as well.

We organize the remaining part of this chapter as follows. In Section 5.2 we describe the SSH related attack detection and prevention methods. We present the proposed Petri-Net based bruteforce attack detection and its category identification in Section 5.3. We describe the experiments done to evaluate the proposed detection method in Section 5.4. In Section 5.5 we describe how the proposed model can be used to detect stealthy attacks. Finally we conclude this chapter in Section 5.6.

5.2 Prior Work

SSH bruteforce attack detection and mitigation methods found in the literature are mainly using the authentication logs at the host level. There are many tools

like DenyHosts [14], BlockHosts [16], PAM-ABL [15], SSHGuard [7] developed for detection and mitigation. As discussed in Chapter 2, these tools do not capture the possible correlations that exists between the attack sources. The closest work to the method of ours (presented in this chapter) is of Javed and Paxon [17]. They analyzed the system logs collected over 8 years at the Lawrence Berkeley National Laboratory super-computing infrastructure. They described a way to separate misspelled passwords with that of genuine attacks using the history of connections. For e.g. if there is a successful connection from the source in question to the SSH server in the past, they do not consider them as attack sources. They also developed a model to detect the low profile attacks spanning several days and months. Our method differs from their study as we use network flow level data to detect in real time and also classify the attack type.

5.3 Proposed Bruteforce Attack Detection and Classification Method

In this section, we describe our proposed bruteforce attack detection method. In the next three subsections we present the problem formulation, bruteforcing detection and classification using Petri-Net model and how different thresholds used by the Petri-Net model are set.

5.3.1 Problem Formulation

Bruteforcing attacks systematically generate passwords for gaining access to server. Bruteforcing can be aggressive and can be done either by a single system or using a collection of systems. Mounting attack using a collection of systems makes the attack stealthier as single source is not generating all login attempts. In this context, there can be three different patterns of attacks as

(i) Single Source based Bruteforcing: Here the attacker uses a single machine and try all credentials one after the other as shown in Figure 5.1.

(ii) Single Domain based Bruteforcing: Attacker has more than one system at her disposal but all of them are geographically located in the same region (e.g. within a country). These systems possess common IP prefixes as shown in Figure 5.2.
(iii) Distributed Bruteforcing: Here the attacker has more than one system at her disposal which are located in different domains. These systems do not share common IP prefixes as shown in Figure 5.3.
We argue that in addition to identifying SSH bruteforcing, it is also important to identify which category of attack is being attempted. This can be helpful in setting appropriate filtering in place to protect SSH servers. For e.g. in case of a single source or single domain based bruteforcing a firewall rule can be added to filter IP address or a domain with known prefix. In this context we aim at both detecting the attack in the first place and subsequently classifying the attack into one or more of the three categories[1].

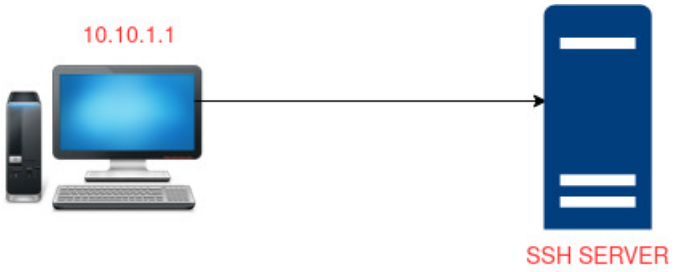

Figure 5.1: Single Source based Bruteforcing

5.3.2 Bruteforcing Detection with Timed Colored Petri-Net Model

Our detection method has three phases of operations as (i) SSH flow identification (ii) Identifying failed login attempts and (iii) Bruteforce detection in an interval. These three phases are elaborated below.

I. SSH Flow Identification: The first phase of detecting attacks is to iden-

[1]There can be parallel attacks of different categories

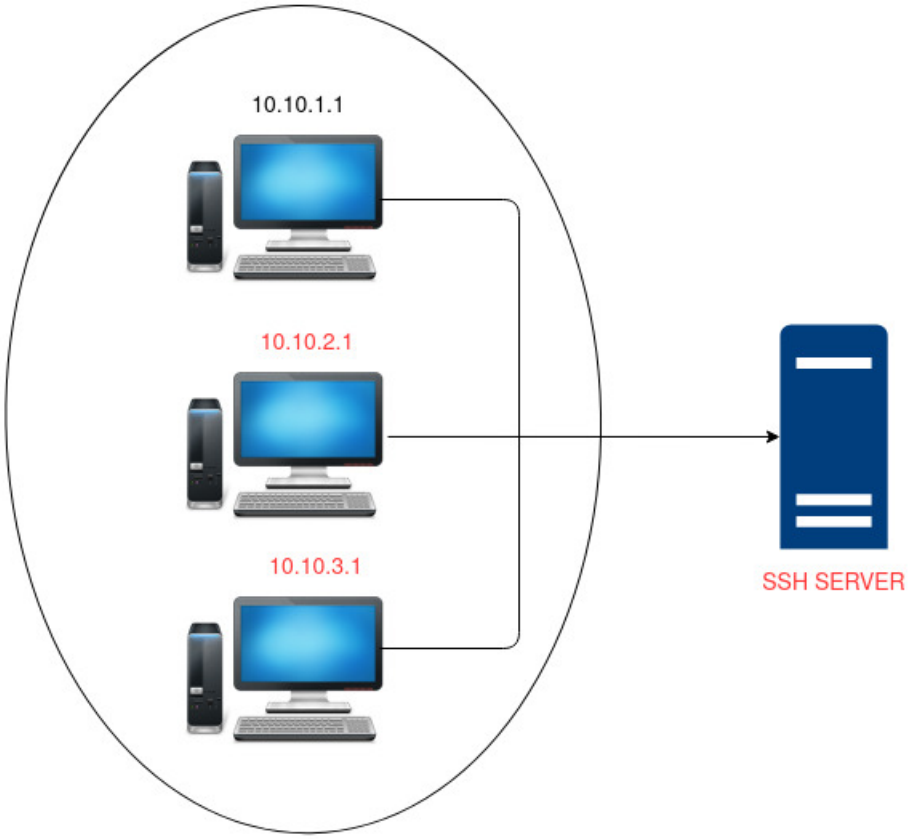

Figure 5.2: Single Domain based Bruteforcing

tify flows related to SSH communication. This can be done either by port number involved in the communication or by using content of flows as described in Chapter 4. First method is simpler of the two and can be deployed where port number of server application is known. Second method requires payload analysis but is reliable compared to the first method. Using content of flows is required when the information about which machines have exposed SSH interfaces is not known. This method uses few initial bytes from flows which typically involves communication related to SSH handshaking. For the current work, for the sake of simplicity, we use the port based detection. This is possible as the details of SSH server is known and moreover we assume the detection method is deployed at the perimeter of the network (ahead of the server).

II. Identifying Failed Login Attempts: To identify flows which are related to successful SSH login cases and failed attempts, we use flow characteristics (as

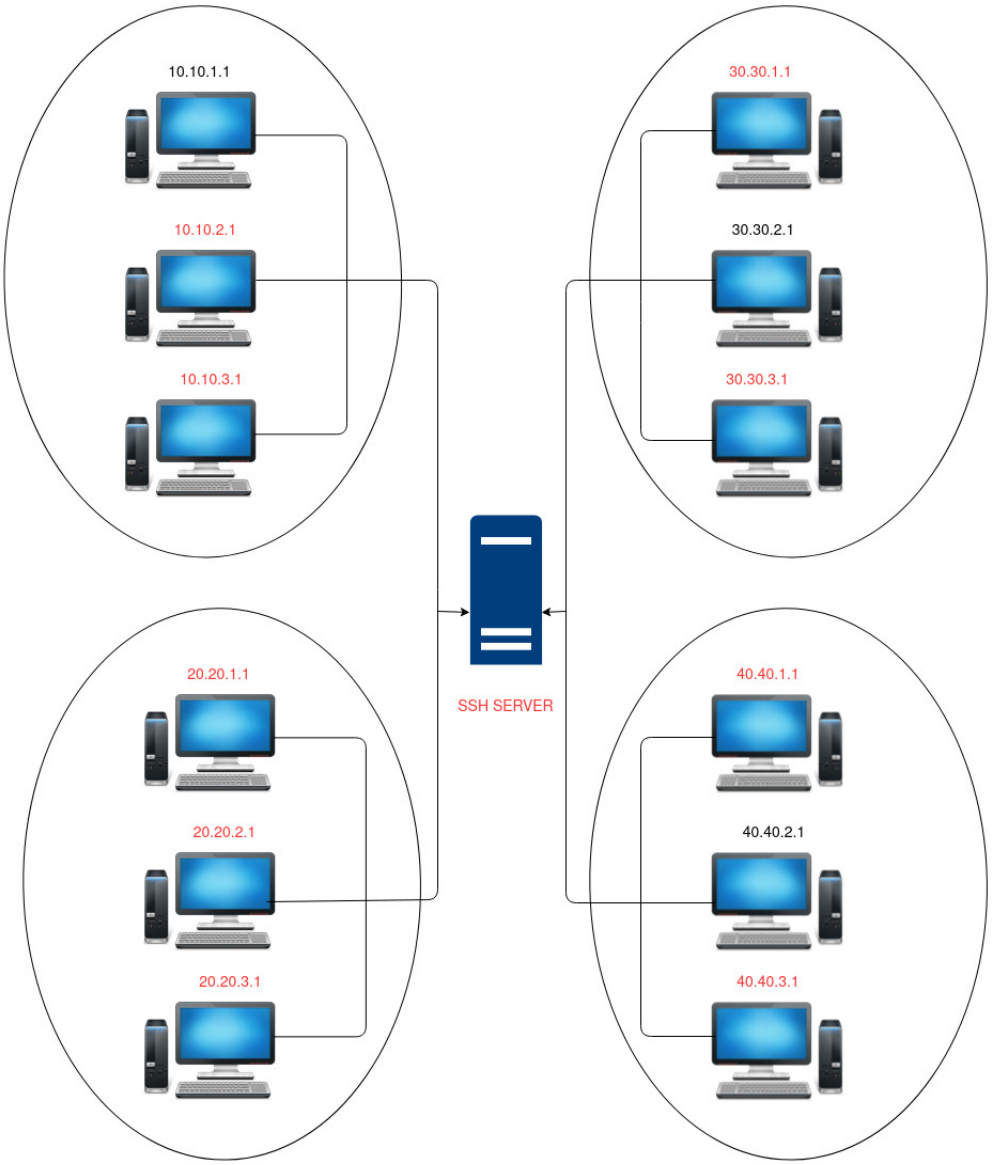

Figure 5.3: Distributed Bruteforcing

used in previous chapters) namely "Number of Packets per Flow" (f_1), "Number of Bytes per Flow" (f_2) and "Flow Duration" (f_3). The selection of these parameters is motivated by the fact that failed login attempts just open a TCP connection and do the initial handshaking by exchanging the list of cryptograhic algorithms supported and make an attempt with one password which does not work. On the other hand, the successful connections also start with handshaking but as users spend time in performing some operations after login, these connections are long lived, having many packets and exchanging several bytes. Similar observations were made in [44] as well. In Section 5.4, we show the comparison of flows for these three parameters for successful and failed login instances using different datasets.

III. Bruteforcing Detection: In order to detect the attacks and subsequent classification, we propose a Timed Colored Petri-Net model.
A Timed Colored Petri-Net is a nine-tuple [47] $CPN = (P, T, A, \Sigma, V, C, G, E, I)$ where:

- P denotes a finite set of elements called places.
- T denotes a finite set of transitions such that $P \cap T = \phi$.
- $A \subseteq P \times T \cup T \times P$ is a finite directed arc set.
- Σ is a colour set representing finite set of types. Every element of this set can be timed or untimed.
- V is a typed variable set such that $Type[v] \in \Sigma \ \forall v \in V$.
- $C : P \rightarrow \Sigma$ is a colour function. The function C assigns a colour set to every place $p \in P$. A place p is said to be timed if $C(p)$ is timed, otherwise p is untimed.
- $G : T \rightarrow EXPR_V$ is a function which assigns a guard expression to transition $t \in T$ such that: $Type[G(t)] = Bool$.

- $E : A \rightarrow EXPR_V$ is a function called as arc expression function. It assigns an arc expression to every arc $a \in A$ such that

  - Type[E(a)] = C(p)$_{MS}$ if p is untimed;
  - Type[E(a)] = C(p)$_{TMS}$ if p is timed.

  Here, p is the place connected to the arc a.

- $I : P \rightarrow EXPR_\phi$ is an initialization function which assigns an expression to a place p such that

  - Type[I(p)] = C(p)$_{MS}$ if p is untimed;
  - Type[I(p)] = C(p)$_{TMS}$ if p is timed.

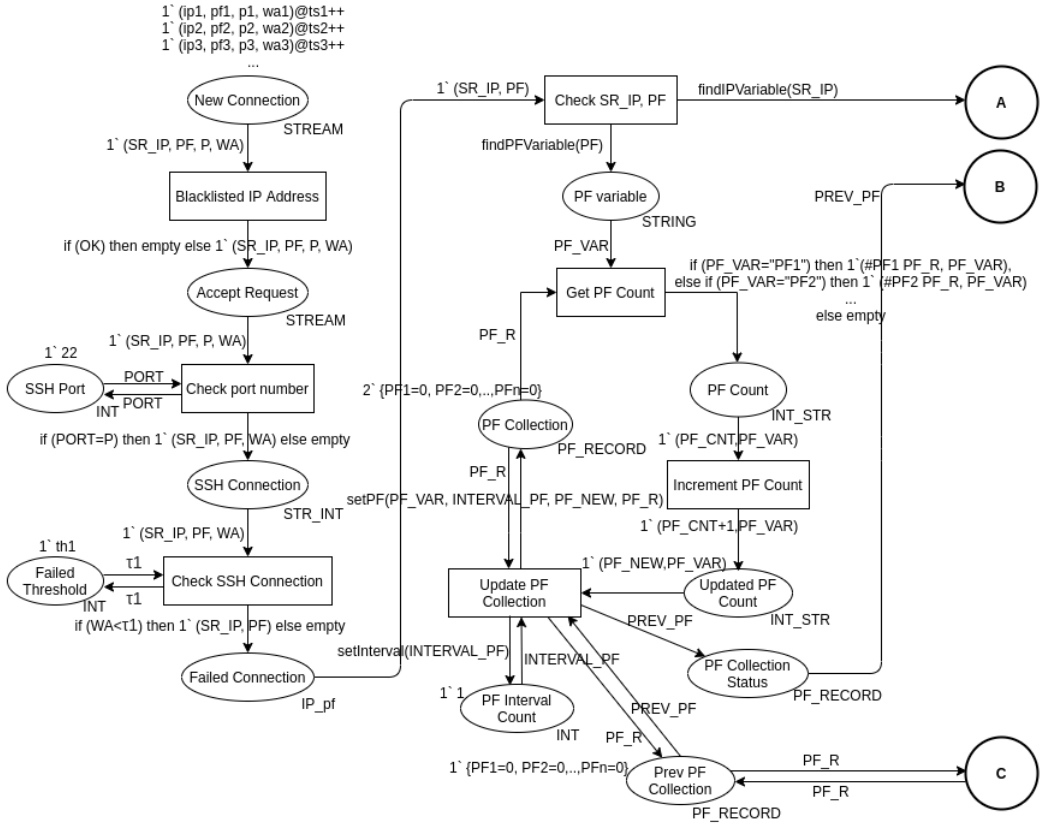

Figure 5.4: Proposed CPN Model Part-1

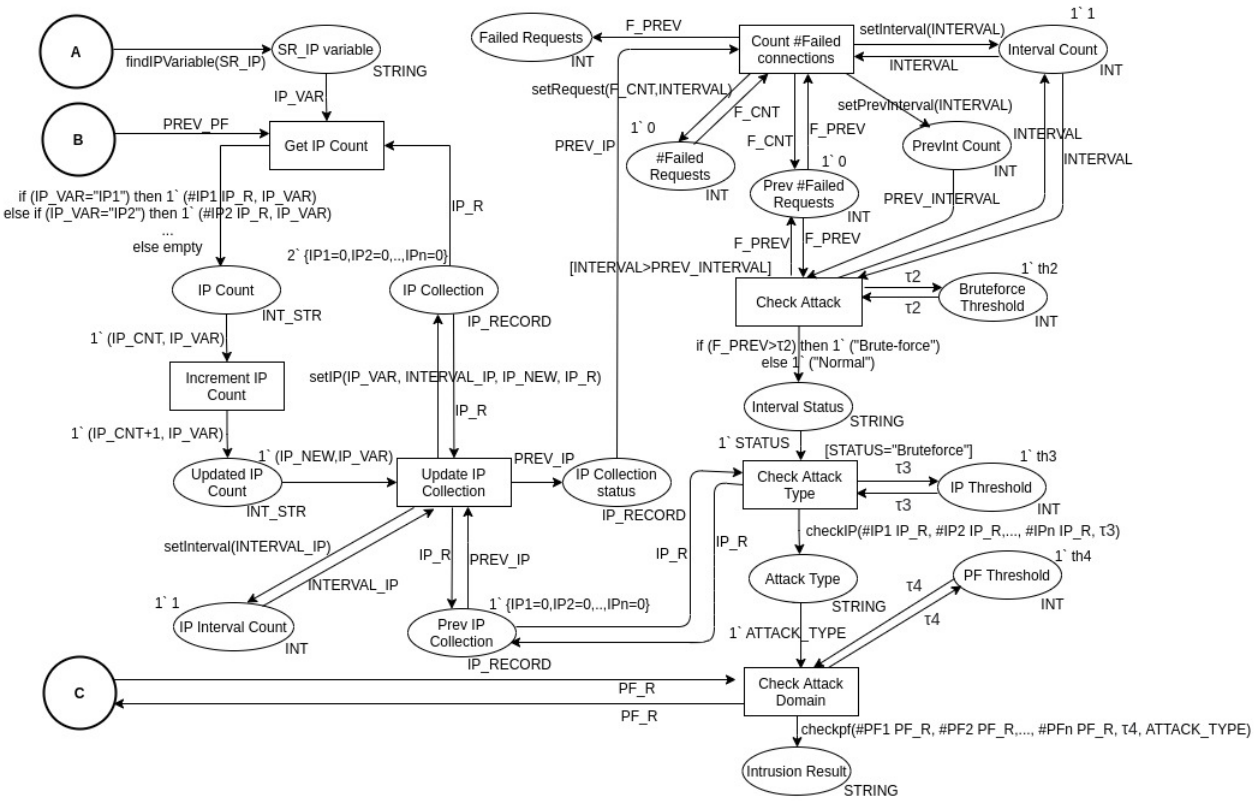

Figure 5.5: Proposed CPN Model Part-2

Our proposed Timed Coloured Petri-Net model is shown in Figures 5.4 and 5.5(circles with label A, B and C are connectors). It has 31 places, 14 transitions, and 69 arcs (all elements of this model are listed in Appendix A). For the sake of easier understanding, we elaborate the operations of this model in four parts as below.

(i) SSH Connection Verification: The place New Connection has timed tokens of user-defined color set STREAM, which is the product of color sets STRING, STRING, INT, and INT. These colors represent the incoming connection's IP address, IP prefix, port number, and weighted average of the flow parameters respectively. The flow parameters are the average duration of the connection, the average number of bytes transmitted in a flow, and the average number of packets in a flow as shown in Equation 5.1. Initially, the model determines whether the incoming connection request's IP address is blacklisted based on previous connection requests, and accepts or denies the connection accordingly. The transition 'Blacklisted IP Address' with outgoing arc has variable 'OK' of Boolean type which takes value true or false, and depending on this value, the transition gets fired. If the connection is accepted, the port number is used to identify an SSH connection. The connection is then identified as failed if the weighted average of flow parameters is less than the threshold value provided in the place named 'Failed Threshold'.

(ii) Keeping Track of Failed Login Connections IP Prefixes: The set of places, transitions and arcs in the right part of Figure 5.4 keep track of various prefixes of failed login attempts. For every failed connection, the model keeps track of the count of the IP address and IP prefix for the IP address involved in the connection. This is done by incrementing the count of a particular IP address and IP prefix whenever a connection is determined as failed. After a failed connection, the transition 'Check SSH Connection' transfers token with the value IP and prefix to the place 'Failed connection'. The transition 'Check SR_IP, PF' with outgoing arcs findPFVariable(PF) and findIPVariable(SR_IP) finds the variable corresponding to the input prefix and IP, respectively. A token in the place 'PF Variable' begins the prefix updating activity. The place 'PF Variable' contains token of type STRING corresponding to an entry in the PF_RECORD. PF_RECORD is a RECORD datatype which is a dictionary-like

data structure with the key-value pair as the PF variable (corresponding to a prefix) and its count, respectively. The transition 'Get PF Count' counts the number of such prefixes observed till now, and transition 'Increment PF Count' increments this count value by one. The transition 'Update PF Collection' with the outgoing arc 'setPF(PF_VAR,INTERVAL_PF,PF_NEW,PF_R)' updates the PF_RECORD by setting value of this PF_VAR with the new value PF_NEW, if the interval value INTERVAL_PF is not changed; otherwise, it resets the record value. The place 'PF Interval Count' contains a token of type INT denoting the current interval whose value gets updated whenever an interval changes. The transition 'Update PF Collection' takes the current interval value 'INTERVAL_PF' as input arc expression, and whenever it gets fired, the output arc setInterval(INTERVAL_PF) checks with the global timer and update the interval value.

Whenever the transition 'Update PF Collection' is fired, a token is transferred to place 'PF Collection Status', denoting the status of PF_RECORD Updation. It also transfers a token PF_R to place 'Prev PF Collection,' with the previous value of PF_RECORD, i.e., the value before its updation. This value of PF_RECORD is used further for attack source classification if an attack is detected. Like this, for each connection request, PF_RECORD gets updated for a given prefix value.

(iii) Keeping Track of Failed Login Connections IP Addresses: Similar to prefix, the model also keeps track of failed IP addresses. Here, IP_RECORD gets updated corresponding to the IP of each failed connection request. The places, transitions and arcs in the left part of Figure 5.5 collectively keep track of IP addresses involved in the failed login connections.

(iv) Detecting Bruteforce Attacks and Classification: After updating the count of the IP prefix and IP addresses involved in failed connection, the model now determines the number of failed connections in an interval and labeling it. The set of places, transitions and arcs in the right part of Figure 5.5 collectively do this detection and classification. The place '#Failed Requests' contains a token with a value denoting the number of failed connection requests in an interval. The transition 'Count #Failed Connections' takes 'F_CNT' in the form of input arc expression and sets the new value

with the help of the output arc expression function 'setRequest(F_CNT,INTERVAL)'. The function increments the F_CNT value by one, if the interval is not changed; otherwise, it resets it to 1(to include the current request). Finally, the old count value F_CNT is transferred to the place 'Prev #Failed Requests'. This value is used to check the attack status for the previous interval. As the interval changes, the model now checks if the previous interval was an attack or normal. The transition 'Check Attack' has a guard function that ensures that it fires only when there is a change in the interval (interval is labeled only after time period is over). The attack status is checked by comparing the number of failed connections in the previous interval 'F_CNT' with the threshold value 'τ_2' specified in the place named 'Bruteforce Threshold.' If the interval status is a Brute-force, the model further checks the type of attack. Transition 'Check Attack Type' takes count of all the failed IP addresses in the previous interval 'IP_R' and the threshold (on number of IP addresses) 'τ_3' as input and checks if it is a Single Source Attack. Similarly, the 'Check Attack Domain' transition takes count of all the failed IP prefixes in the previous interval 'PF_R' and the PF Threshold 'τ_4' as input and checks if it is a Single Domain Attack or Purely Distributed attack case. At last, the token in the state 'Intrusion Result' denotes the type of attack for the particular interval.

5.3.3 Setting Threshold Values for the Model

We can notice that the proposed CPN model uses few threshold values in its operation. It uses thresholds for differentiating failed and successful SSH connection flows (τ_1), declaring an interval as bruteforce attack (τ_2), classifying a bruteforce attack interval as single source attack (τ_3) and classifying an interval as single domain attack (τ_4).

(i) Setting threshold τ_1: In order to detect failed logins, the model uses the weighted average of parameters of the flow as used in Chapter 4, i.e., the average number of packets (f_1), the average number of bytes transmitted in a flow (f_2), and the average

duration of the connection (f_3), in a flow as shown in Equation 5.1.

$$WeightedAverage = 0.33 \times f_1 \quad +0.33 \times f_2 + 0.34 \times f_3 \tag{5.1}$$

This weighted average is calculated for all flows used for deriving threshold. Using these values of weighted averages the mean (μ_1) and standard deviation (σ_1) are calculated. We set the threshold on these values, as shown in the equation 5.2. Here $\delta 1$ is a constant multiplier which takes a value less than 1. This is because, the mean and standard deviation of flows with successful SSH connections are used.

$$\tau_1 = \lceil (\mu_1 + \delta_1 \times \sigma_1) \rceil \tag{5.2}$$

A flow having the weighted average of parameters less than this threshold value τ_1 is detected as corresponding to a failed SSH connection. This is due to the observation that failed connections transmit less number of packets and bytes, and also have smaller duration.

(ii) Setting threshold τ_2: Once the flows corresponding to failed SSH connections are detected, subsequently the model aggregates them over an interval and label it either as interval experiencing bruteforcing or otherwise. This is decided using the threshold τ_2. In order to set τ_2, we modelled the average number of failed login connections in a time interval as Poisson probability distribution as described in Chapter 4. Equation 5.3 shows the probability distribution function where μ_2 denotes the average number of failed login counts in an interval, n is a value of failed logins and e is a positive constant.

$$f(n, \mu_2) = \frac{{\mu_2}^n \times e^{-\mu_2}}{n!} \tag{5.3}$$

To set an appropriate value to τ_2 (to detect bruteforcing), we use the one-sided Chebyshev inequality as shown in Equation 5.4. Here X is a random variable with mean μ_2

and variance σ^2, and c is a positive constant.

$$P(X \geqslant \mu_2 + c) \leqslant \frac{{\sigma_2}^2}{{\sigma_2}^2 + c^2} \tag{5.4}$$

The threshold value τ_2 is set such that it corresponds to a low probability event using chebyshew inequality (in the tail of a Poisson probability distribution) as used in the Chapter 4 and shown in Equation 5.5.

$$\tau_2 = X \ s.t \ P(X = \lceil \beta \times \sigma_2 \rceil) \tag{5.5}$$

(iii) Setting threshold τ_3: Once an interval is detected as attack by the model it subsequently verifies whether it has single source attack using τ_3. Single source attacks are detected by the model if the number of failed login flows from a source exceeds the threshold τ_3 whose value is set as in Equation 5.6.

$$\tau_3 = \lceil (\mu_3 + \delta_3 \times \sigma_3) \rceil \tag{5.6}$$

Here μ_3 denotes the mean, δ_3 is a positive constant and σ_3 is the standard deviation which is calculated using number of connection requests from a particular IP address.
(iv) Setting threshold τ_4: Single domain attacks are detected by the model if the number of incoming failed requests from an IP prefix are more than τ_4. The threshold τ_4 is set as shown in Equation 5.7.

$$\tau_4 = \lceil (\mu_4 + \delta_4 \times \sigma_4) \rceil \tag{5.7}$$

Here μ_4 denotes the mean, δ_4 is a positive constant, and σ_4 is the standard deviation calculated from number of connection requests originating from an IP prefix.

Those intervals which are marked as bruteforcing and having more than one prefixes (either individually exceeding τ_3 or otherwise) but exceeding τ_4 are declared as distributed bruteforcing attacks.

5.4 Experiments and Evaluation

In this section we describe the experiments done to evaluate the proposed Petri-Net based detection method. In the next four subsections we describe the datasets used in the experiments, failed and successful SSH connection comparison, implementation of CPN model and sensitivity analysis of detection performance with various parameters used in the experiments.

5.4.1 Dataset Description:

We used two datasets for our experiments. First dataset is taken from a SSH server deployed in our university campus network and second one is collected from testbed setup[2] in our Lab.

I. SSH Traffic Data from Production Level Server: This dataset is the same dataset used in the experiments of the approach described in the previous chapter. As described in Chapter 4, this dataset is collected from an SSH server located in the server farm of our university network. Many users use this server for running high performance commuting applications. We collected SSH network traffic from this server for a period of 45 hours using tcpdump tool [40]. We used this dataset both for setting threshold for bruteforce attack detection and also to evaluate the detection performance of different simulated attack types.

II. SSH Traffic Data Generated from Testbed: We setup an SSH server on a Ubuntu Machine with 10 valid users. We generated both normal SSH login connections and bruteforce attacks against this server by generating requests using Python scripts. All SSH network traffic related to this attack is collected using tcpdump tool [40] on the server.

Normal SSH Flows: The goal here is to generate a dataset based on successful SSH

[2]We interchangeably use the name data generated from testbed and simulation setup.

connections. Typically a user after login will do some activity in the target machine. The activities could be navigating to a directory, some file operations and running some applications, etc. In order to generate such SSH connections, we used two scripts. First script establishes connection to the server and executes upto 15 Linux commands after login. These commands reflect a user activity upon login. It executes successive commands after a random delay. The delay between successive commands is set by generating a random number which is in between 10 seconds to 90 seconds. The total number of commands to be executed is decided with another random number generated at the beginning. The operations of this script are shown in Algorithm 1. The random number of commands executed and random delay between successive commands mimic the communication pattern of a normal connection when a user login. We used a second script which invoked multiple instances of script-1 from five different virtual machines using valid usernames and passwords to generate connections. Hence many connections (flows) overlap in this case.

Algorithm 1 Normal SSH Connection

Input: η - Total Number of Commands
Input: t_1 and t_2: Minimum and Maximum Time Delay
Input: COMMANDS [1, η]

```
1: while not interrupted do
2:     K1 ← RandomNumber(1,η)
3:     NewSSHConnection()
4:     for i = 1 to K1 do
5:         Execute(COMMANDS[i])
6:         K2 ← RandomNumber(t1, t2)
7:         Sleep(K2)
8:     end for
9: end while
```

Bruteforce Attack Flows: To generate bruteforce attack flows, we wrote another Python script which made successive login attempts using randomly generated text as password for valid usernames. As required we generated three variants of brutforce attack types from multiple virtual machines as detailed below.

(i) *Single Source Attack*: Here single instance of Python script deployed in a virtual machine attempted SSH connections. As all these connection attempts use the IP

address of virtual machine, all flows have same IP address. We executed this attack for 36 hours continuously and collected traffic corresponding to this attack.
(ii) *Single Prefix Attack*: We deployed the attack generation Python script in five virtual machines for generating this attack variant. All the five scripts parallely generated SSH login attempts to the server. The network traffic corresponding to this attack was collected. All the five virtual machines were assigned IP addresses with a common prefix of 24 bits. We mounted this attack in four batches each of duration of nine hours totalling 36 hours. We changed the IP address range assigned to virtual machines in each batch thus giving four different IP prefix values.
(iii) *Distributed Attack*: We used 10 virtual machines for this experiment with each virtual machine assigned an IP address without having overlapping prefixes. An instance of attack Python script was running in each virtual machine which generated failed SSH login connections. The data collected from this setup serves as a distributed attack for our evaluation.

5.4.2 Characteristics of Failed SSH Connections

As mentioned previously in Chapter 3 and Chapter 4 and in Section 5.3 failed and successful SSH connection flows are identified using flow parameters. The Petri-Net model also uses the same parameters to seperate the two. In particular it uses the weighted average of three parameters namely `Number of Packets per Flow`, `Number of Bytes per Flow` and `Flow Duration`. In Chpater 3 we showed that these parameters are helpful in identifying the flows corresponding to failed logins. As we used two types of datasets namely the flows collected from institute SSH server and also Simulated dataset, we repeat this exercise here with these two types of datsets. Here we show that these parameters are useful in differentiating normal and failed login attempts by comparing these parameters from flows of successful login instances and failed login cases. For this comparison we used 50 SSH connection flows each from i) institute SSH server ii) normal SSH flows from simulation setup and iii) failed SSH connections in bruteforce attack and plotted the graphs of these three parameters.

Figure 5.6, 5.7 and 5.8 show the graphs of these three parameters for the three

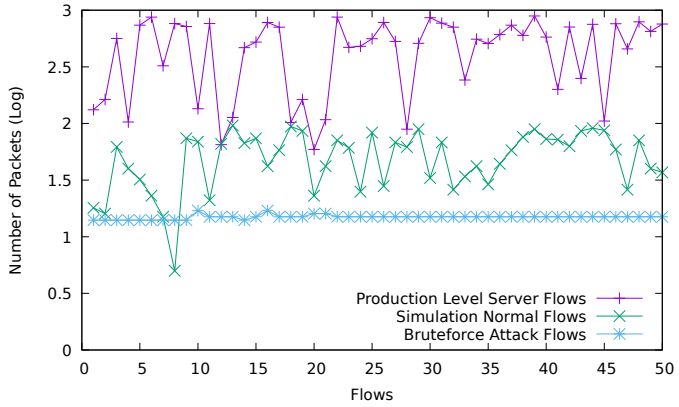

Figure 5.6: Number of Packets Per Flow for Bruteforce Attack and Successful SSH Connections

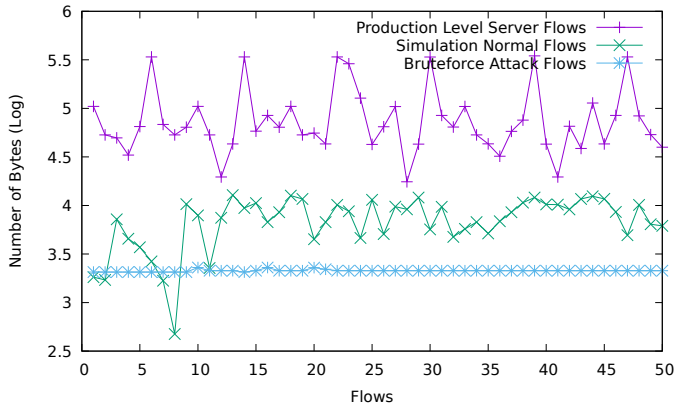

Figure 5.7: Number of Bytes Per Flow for Bruteforce Attack and Successful SSH Connections

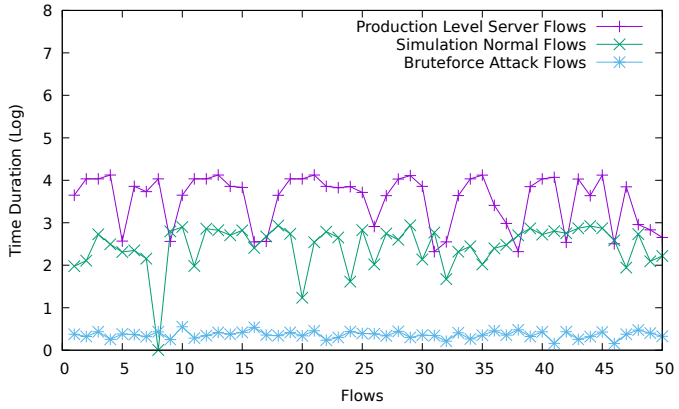

Figure 5.8: Time Duration Per Flow for Bruteforce Attack and Successful SSH Connections

types of flows. The X-axis in all three graphs is the flow number and Y axis has their respective parameter values. From Figure 5.6 we can notice that number of packets in successful SSH connections and failed login cases are quite different with successful login flows having many packets compared to flows corresponding to failed SSH connections. Figure 5.7 shows the transmitted byte count (within a flow) comparison for the three types of flows. We can again see that they have a clearly distinct characteristics with successful login cases exchanging large number of bytes compared to failed cases. Similarly the duration of flow comparison is shown in Figure 5.8. We can observe that failed login connections are short lived while successful SSH connections have longer duration. It is worth noticing that in all the three graphs, the Y axis values are shown in log scale.

5.4.3 Implementation and Evaluation:

First step in the implementation was to create a model which is formally correct and verifiable. Thus we first created a model using CPN tools [48] (Figure 5.4 and Figure 5.5 are redrawn versions of model generated using CPN tools) and once its correctness is verified using simulation, we converted into a workable program.

We implemented the proposed Petri-Net model as a Python script. Our Python code can read any network trace file in `pcap` or `tcpdump` file format and process it to reconstruct network flows. From those processed flows, it extracts the required parameters to mark every flow as either successful or failed instance. It also uses source IP address (other than SSH server IP address) attribute to check if this is already in the blacklist. If it is not in the blacklist, the flow is processed further to do other assessments as depicted in Figures 5.4 and 5.5. As flow details are aggregated over an interval for detecting attacks, the CPN model keeps track of different flows and their attributes for bruteforce attack detection and their type. The model uses threshold values for detecting failed attempts, bruteforce attack detection and subsequent classification as described in Section 5.3.3.

For our experiments, we divided the datasets from known no bruteforce attack intervals (both collected from institute server and generated with simulation) into two

parts. First part was used for setting the threshold for differentiating successful and failed SSH flows using the flow characteristics. Second portion of this data and all three types of attack datasets are used for evaluating the detection performance. Table 5.1 shows the characteristics of flows collected from known no bruteforce attack intervals in terms of number of flows of successful and failed login attempts, average number of packets, bytes in the flow and also the average duration of connection. Although the Table 5.1 shows the statistics of no attack intervals, we can see that there are quite a good number of failed login connections for institute server traffic. These failed login connections are from two sources. First source of such cases are from internal users and the second category of requests are from external sources. Failed connections from internal users are mainly due to those users who might have forgotten their login credentials and when they try some wrong passwords they result in failed connection attempts. We manually screened the external connection source IP addresses and many of them were not having any correlation except one IP series having prefix of 222.186.*.*. There were 34 such connection requests from this prefix with 8 distinct IP addresses in the Institute-Part1 dataset (5 connections in Institute-Part2) and these connections had flow parameters slightly contrary to the ones shown in Figure 5.8. These flows had less number of packets, less number of bytes being transmitted but unusually large connection duration. These connections are the ones which influence the average connection duration of failed login cases as depicted in the 10^{th} column of first and second rows in Table 5.1. Similarly the parameters of bruteforce attack dataset for different categories are shown in Table 5.2. We can notice that these parameter values confirm the observations of Figures 5.6, 5.7 and 5.8 when compared to the parameter values of Table 5.1.

Table 5.1: Dataset Statistics of Normal Intervals

Type	Interval	Total Flow Count	Successful Connection Flows				Failed Connection Flows			
			Flow Count	Avg Pkt Count	Avg Duration(s)	Avg Bytes	Flow Count	Avg Pkt Count	Avg Duration(s)	Avg Bytes
Institute-Part1	48	962	68	576.544	6219.899	106412.132	894	11.543	133.222	1691.709
Institute-Part2	42	546	42	483.904	2265.776	84531.666	504	10.521	75.481	1389.087
Simulation-Part1	96	4727	4727	59.27	401.719	8624.518	NA	NA	NA	NA
Simulation-Part2	48	2416	2416	57.381	392.44	8462.422	NA	NA	NA	NA

We processed the data collected from institute server and also from our testbed setup using the Python script which implements the proposed model. We evaluated the

Table 5.2: Dataset Statistics of SSH Bruteforce Attacks

Type of Flows	Intervals	Flow Count	Avg Failed Login Count	Avg Duration(s)	Avg Pkt Count	Avg Bytes
Simulation-Single Source	72	55337	768.569	2.135	14.813	2114.36
Simulation-Single Domain	72	275653	3828.5	2.128	14.784	2111.987
Simulation-Distributed	72	556326	7726.680	2.140	13.535	2048.730

detection performance of the proposed model for interval periods of 30 minutes and for different values of τ_2, τ_3 and τ_4. There were two sets of experiments, first one using the institute server data (Part-1 and Part-2) and all the attack dataset. Second one using the simulated (Part-1 and Part-2) and all attack datasets. We empirically found the performance variance for these values and chose the best combination yielding higher detection rate. Table 5.3 shows the performance results for the best combination of values (shown in the table) for the first experiment where Institute-Part1 data was used for deriving thresholds and the Part2 along with all attacks are used for testing. We can notice that single source attack intervals are detected as single source attacks and none of these intervals are declared as other two types as in the fourth row in the table. When the dataset corresponding to the 72 intervals of single domain bruteforcing attack is presented to the model with same threshold values, 98.61 percentage of the intervals are also detected as single source attack in addition to being detected as single domain attacks. This is due to the fact that although five sources sharing a common prefix generated the brutefrocing, the number of failed logins from these sources individually also crossed the threshold τ_3. Similar interpretation goes for the distributed attack flows as well. We believe that these detection results do not conflict. The interpretation is that it will enhance the understanding of the sources contributing to the attack and their frequency and coordination. Our Python code also automatically identifies the IP prefixes for single domain attack cases. The four IP prefixes identified after processing the 72 interval of single domain attack data are shown in Table 5.4. We can notice that it generated four 24 bit prefixes with each prefix is the aggregation of 5 IP addresses corresponding to 5 virtual machines. Similar results were obtained with the second set of experiments as well and we omit them showing here for brevity.

Table 5.3: Bruteforce Attack Detection

Parameters τ_1 =1514 τ_2=739, τ_3=739 and τ_4=709							
	Single Source		Single Domain		Distributed Cases		
	DR	Source Count	DR	Source Count	DR	Source Count	FP
Single Source	100	1	0	NA	0	NA	0.00
Single Domain	98.61	20	100	20	NA	NA	0.00
Distributed	98.61	10	0	NA	100	10	0.00

Table 5.4: Prefix Generated from the Program

Type of Data	**Prefix**	**Source Count**
Single Domain	10.242.3.*	5
	10.242.7.*	5
	10.242.11.*	5
	10.242.15.*	5

5.4.4 Sensitivity Analysis

We can notice that our CPN model uses threshold values for detecting attacks and classifying them. The detection performance is dependant on these values. In order to understand the performance variance with different thresholds, we performed an experiment. In this experiment we assess the detection performance variation with the three parameters used namely τ_2, τ_3 and τ_4. We collected three sets of readings and in each reading, we set two parameters constant and vary one parameter in step size values of one and calculate the detection rate. Figures 5.9, 5.10 and 5.11 show the detection rate variation with these three parameters. From Figure 5.9, we can see that as threshold τ_2 increases, bruteforce attack detection rate and also single source attack detection rate decrease. This is because, with an increased threshold value τ_2, the number of failed logins required for declaring an interval as attack increases. This also means that at least those many number of failed logins are required for classifying it as single source attack and other types. In our dataset the single domain and distributed attacks have roughly five and ten times more number of connections related to failed login attempts compared to single source attack. The large number of failed connections easily pass the increased threshold of τ_2 too. That is why the detection rate for these two attack cases did not decrease. However if the the single domain and distributed attacks happen to have the same number of failed logins as

single source attack, their detection rate will also decrease. From Figure 5.10 we can notice that, as the threshold τ_3 increases, the detection rate for a single source attack decreases but the overall bruteforce attack detection rate is not impacted. In addition, the detection rate for other cases also remain unaffected since threshold τ_3 is responsible only for single source attack detection and the other two threshold values, τ_2 (the threshold for attack detection) and τ_4 (the threshold for distributed attack cases) are constant. Similarly from Figure 5.11, we can see that the increase in threshold value τ_4 affects the detection rate for distributed attack cases. From these figures and the discussion we can conclude that in general as thresholds increase, the detection rate decreases.

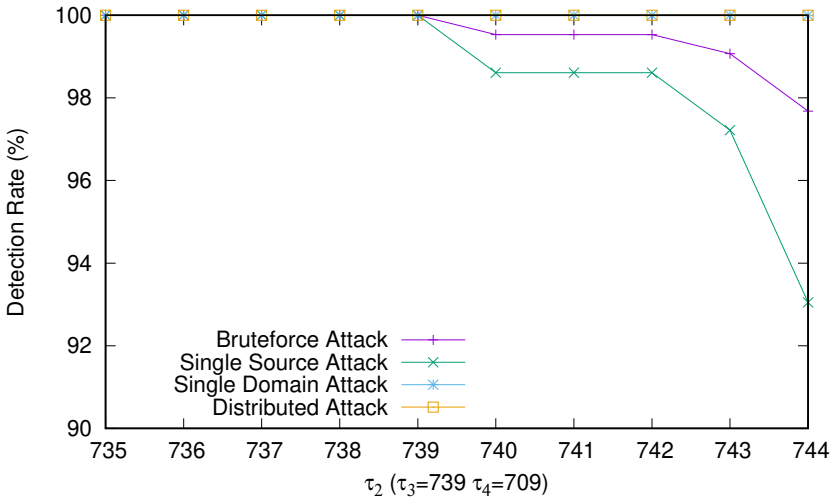

Figure 5.9: Bruteforce Attack Detection Rate for Different Values of Threshold τ_2

5.5 Detecting Stealth Bruteforce Attacks

Our proposed CPN model can detect bruteforce attacks not only originating from single source; but also from coordinated sources. In order to evade detection, an attacker can thin out the activity such that her activities do not cross any thresholds. This is a longstanding and challenging problem in cyber security research. In this section we discuss how such low profile attacks can be detected. There are two ways this can be handled.

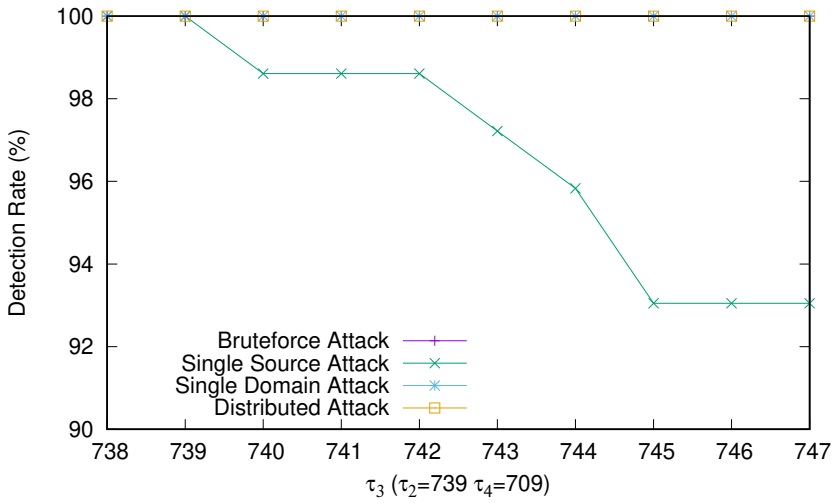

Figure 5.10: Bruteforce Attack Detection Rate for Different Values of Threshold τ_3

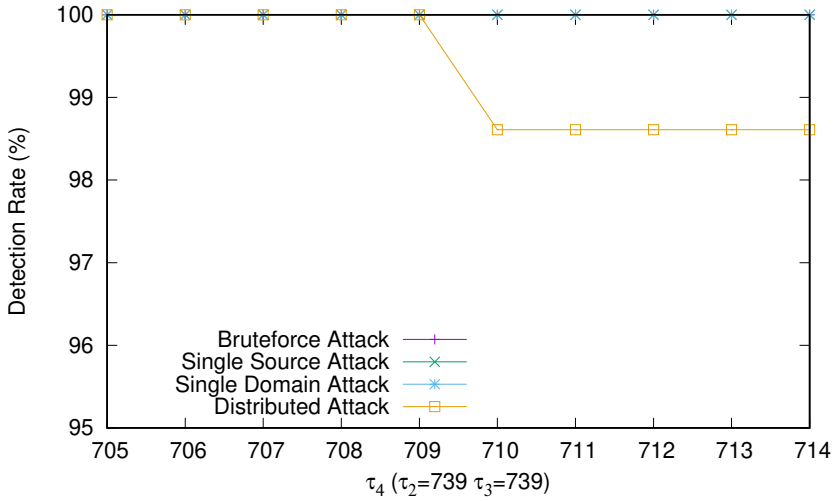

Figure 5.11: Bruteforce Attack Detection Rate for Different Values of Threshold τ_4

(i) Increasing the Time Interval: By increasing the time interval used for detecting the attacks, the activities which are falling across the boundaries of multiple smaller time intervals can be noticed.

(ii) Reducing the Thresholds: Second method is to reduce the threshold values so that even low profile activities are noticed. However there is a trade-off between the attack detection rate and false alarm rate.

Table 5.5: Dataset Statistics of SSH Stealthy Bruteforce Attack

Type	Total Flow Count	Normal Connection Flows				Attack Connection Flows			
		Flow Count	Avg Pkt Count	Avg Duration	Avg Bytes	Flow Count	Avg Pkt Count	Avg Duration	Avg Bytes
Simulation Single Source Stealthy	872	860	56.732	402.287	8429.651	12	14.66	2.065	2104
Simulation Single Domain Stealthy	876	864	57.773	400.535	8511.546	12	14.83	2.284	2115

Table 5.6: Stealthy Bruteforce Attack Detection

Attack Type	Duration	Intervals	Detected	Detected Failed Logins		Threshold τ_1	Threshold τ_2
				Bruteforce Attack	Normal		
Single Source	12 Hours	2	2	12	1	1424	5
		3	3	12	3	1469	3
		4	4	12	3	1471	2
Multiple Sources	12 Hours	2	2	12	2	1424	5
		3	3	12	3	1469	3
		4	4	12	5	1471	2

In order to detect such stealthy attacks using proposed CPN model, we adopt both the methods i.e., increasing the time horizon and decreasing the thresholds. To evaluate the detection performance of the proposed method, we generate stealthy attacks and collect another dataset. We generated stealthy attacks both from single source and distributed sources at low frequency of approximately one attempt for every one hour along with normal SSH connections. We used the Python script 1 for generating successful SSH connections (as in Algorithm-1) and parallely the attack generator script made connection attempts using randomly generated passwords. This combination of SSH connections serves as stealthy attack case for our experiments. Both single source and multi-source attacks were launched for 12 hours duration each. Table 5.5 shows the statistics of this dataset in terms of number of connections and their average flow characteristics.

We evaluated the detection performance of the proposed method for different intervals of time and the results are presented in Table 5.6. The twelve hour data was

divided into intervals of six, four and three hours each and evaluated the detection performance. For this experiment we used the Simulation Part-1 dataset for deriving the thresholds and used Part-2 of this dataset along with the stealthy dataset for evaluating detection performance. Table 5.6 shows the duration of data collection for each type of attack (which is 12 hours), number of intervals, intervals detected as attack along with the total number of failed login connections for the entire twelve hours (from both normal and bruteforce attack cases). The seventh column shows the threshold τ_1 used for detecting failed login connections (this threshold changes as the interval size changes because all the parameters are estimated over the same duration using Simulated-Part1 dataset). The last column shows the threshold used for detecting bruteforcing attack τ_2. It is worth noting that τ_2 is set using Equation 5.5 which in turn uses the mean number of failed cases in an interval. However in Simulation-Part1 dataset there are no failed login cases. Thus we used a low value of 0.1 as the mean number of failed login cases and adjusted the value of τ_2 according to chebyshew inequality constraint of Equation 5.4. For e.g. the first row in the table indicates that twelve hours of data is divided into two intervals of six hours and both of them are correctly identified as attack intervals. In the two intervals all twelve flows belonging to bruteforcing attack cases are detected as failed logins while one flow out of 860 flows of successful SSH connection is declared as failed login for the threshold value 1424. Similar interpretation goes for the remaining rows as well. We can notice that proposed CPN model can adopt to different behaviors and attack patterns and can detect attacks with minimum number of errors.

5.6 Conclusion

Secure Socket Shell exposes an interface for credential based remote login. SSH applications are susceptible to bruteforcing. In this chapter, we proposed a Petri-Net model to detect SSH bruteforcing attacks and subsequently classify them into one of three types as originating from single source, single domain or distributed attacks. We evaluated the proposed model with network traffic collected from production level

server and also generated using a testbed setup. The proposed model is not only effective in detecting such attacks but can also classify them.

Chapter 6

Conclusion and Future Work

This chapter summarizes the SSH authentication log analysis and SSH attack detection techniques presented in the thesis and provides few directions for future work in this area. The objective of our work is to develop a method to detect bruteforce attacks using network packet/flow level information. As a secondary objective, we aim to classify these attacks based on the possible similarity among sources of attacks.
The motivation for the thesis stems from the fact that protecting SSH servers against bruteforce attacks is required for safe computing. Consequences of a compromised systems can be severe. We first presented a case study on SSH authentication logs and then proposed methods for SSH attack detection. We covered previous works and literature related to SSH case studies and auditing. We also discussed work related to SSH bruteforce attack detection based on various techniques(threshold-based, statistical, machine learning). Subsequently the research gaps were identified that many existing tools are host-based and not scalable for large networks. An alternative method would be to use network traffic or flow-level information. Although these methods are scalable, they are limited to identifying whether bruteforcing against the server is attempted or not. Our proposed method not only detects such attacks but also identifies any correlation between them. With the proposed method, we were also able to identify stealthy and low-profile attacks as well. We experimented with datsets collected from real SSH servers and also generated in a testbed.

6.1 Thesis Contribution

This section summarises the SSH bruteforce attack detection methods and attack categorization to understand the possible correlation between the contributing sources. We also studied SSH authentication logs to determine attack trends before presenting our attack detection methods. These contributions are summarized in the following subsections.

6.1.1 Secure Socket Shell Authentication Log Analysis

As a first contribution, we presented a case study using SSH logs collected from a server deployed in our university network. Through this study, we were able to get insights into the attack patterns. Our analysis reveals that attackers try various methods to break into the system. The observations made were as follows:

1. We identified the most common usernames used by attackers to obtain access and found that a few of them are quite common. The top ten usernames were used in 12.8% of attempted break-in attempts.

2. A handful of sources make repeated attempts to break into the system spanning several weeks. The analysis showed that a large fraction of source IP addresses and prefixes were repeating in many weeks data generating failed logins.

3. Further, the origin of attacks was geographically well spread. We mapped sources of such attacks to their country-wise geographical location to find that malicious login attempts come from vary diverse sources and locations, with few countries having a significant share.

After studying the logs and finding trends, we also proposed a flow-based method to identify successful and failed login attempts. This method used few parameters taken from the network flows. We used machine learning algorithms to classify the flows corresponding to failed and successful SSH connections. These algorithms showed very high accuracy of classification in our experiments.

6.1.2 SSH Bruteforce Attack Detection with Probability Distribution Model

Our second contribution proposes a method to detect SSH bruteforce attacks using network flow level data. This detection method has three phases as
i) Identifying flows belonging to SSH connections
ii) Classifying each flow either as corresponding to successful or failed SSH connection and
iii) Detecting bruteforce attacks.

For the first phase, our proposed method uses a combination of port number and content of flows. It uses content similarity between a known reference SSH flow and the flow in question to decide whether the flow belongs to SSH connection. This is done with Deep Packet Inspection (DPI) by comparing the keywords within the flow. SSH clients and servers exchange a set of supported protocol suits as part of their handshaking to negotiate and agree on a particular protocol suit. This exchange of protocols is typically done in plain text format, which allowed us to do DPI on this portion of data.

SSH flows corresponding to failed login attempts were identified with a set of statistical flow parameters. The premise for using flow parameters comes from the observation that, flows corresponding to failed connections have a few number of packets in a flow, less number of bytes transferred between the host and server, and low duration of the connection.

Subsequently, we proposed a method for detecting SSH bruteforce attacks. Login failures arise in two situations. First, if a legitimate user forgets her credentials, second case when a bruteforce attack has been attempted. To distinguish between these two cases, we model the failed SSH connections in a known non attack interval as a Poisson probability distribution. Using this distribution, our model identifies the rare probability events corresponding to unusual number of failed logins in a chosen interval as bruteforce atacks. We validated our approach by experimenting with network traffic collected from a production level SSH server and a testbed setup.

6.1.3 SSH Bruteforce Attack Detection and Classification with Pertri-Net Modeling

The probability distribution model discussed in the previous subsection is limited to identifying whether there was an attack against the server. An attacker can try avoiding detection by keeping a low profile about the activities and launching the attacks from several sources so that no single source generates password guesses vigorously. Therefore individually, these sources may or may not be attempting to login aggressively. If the attack sources are distributed, this will result in a change in the IP address and port parameters. This poses difficulty for detecting these types of attacks. In our next contribution, we build on the idea described previously. In this work we present a method to detect coordinated SSH bruteforce attacks. We extend our previous model of detecting attacks into both detection and classification method. Proposed method classify the bruteforce attacks as one or more of three types based on the source of the attack. It classifies the attacks as single source, single domain and/or distributed attack. For detecting and classifying the bruteforce attacks we proposed a Petri-Net-based state transition model. We implemented this model in python language. This implementation is able to process the network flow data as input and assign labels to chosen intervals of test data. As flow details get aggregated over an interval for detecting attacks, the proposed model keeps track of different flows and their attributes for the detection of bruteforce attack and their type. As in the previous contribution, we use SSH network traffic collected from the production level SSH server and from the testbed setup to evaluate the proposed method. We also extended this work to identify low profile and stealthy attacks. The proposed model adopts to detect such attacks with two modifications. First by setting low thresholds for detection and second by increasing the time horizon. We generated stealthy attacks in a testbed and evaluated the detection performance of the proposed model.

Overall the two proposed models were successful in identifying the bruteforce attacks against SSH servers.

6.2 Future Work

Our work on SSH authentication log analysis and bruteforce attack detection can be extended in many ways. Following are some of the possible extensions.

1. `Proactive Logging of Events:` We recollect that, operating systems log events pertaining to login attempts. However these logs only indicate whether there was a successful login from a source and how many times the source attempted before succeeding to login. Once an adversary succeeds to login, she has complete control over the machine and can perform many activities. One can extend our work to initiate proactive log generation after a suspicious successful login. This can monitor user activity more closely and may help answering questions related to what the attacker did after gaining access to the system.

2. `Developing User Interface:` Our proposed work based on Petri-Net modeling and also Probability distribution based model can be extended further to create an interactive user interface. The network administrator can use this as a tool to get detection results and attack statistics such as time and attack source information as reports.

3. `Integration with IDS:` The proposed Petri-Net model for attack detection can be integrated with Intrusion Detection and Prevention systems like SNORT. With this, the overall performance can be evaluated for real-time network traffic. Rules can be added in IPS to block the malicious IP address identified from the detection method.

4. `Attack Detection on Other Applications:` There are several other applications such as Web servers, File Transfer Protocol, which are also the target of such bruteforce attacks. The proposed Petri-Net model and Probability distribution models for SSH attack detection and classification can be adopted to detect attacks on these applications as well.

5. `Identifying Different Attack Phases:` The proposed methods for SSH bruteforce attack detection use the network flow information. Some of the prior

works made observation that the number of packets in different phases of brute-force attacks are different. The given method can be extended to consider various phases which are observed during a bruteforce attack. The work [20] described a SSH attack detection method mentioning these attack phases, with every phase having different flow characteristics based on the packet per-flow and the number of flow records in a given time interval. These observations can be incorporated into our model as part of future work.

Bibliography

[1] "https://www.putty.org/(accessed on 20-10-2020)."

[2] R. Hofstede, M. Jonker, A. Sperotto, and A. Pras, "Flow-Based Web Application Brute-Force Attack and Compromise Detection," *Journal of Network and System Management*, vol. 25, no. 4, pp. 735–758, 2017.

[3] N. DeMarinis, S. Tellex, V. P. Kemerlis, G. Konidaris, and R. Fonseca, "Scanning the Internet for ROS: A view of Security in Robotics Research," in *ICRA'19: Preceedings of the International Conference on Robotics and Automation*. IEEE, 2019, pp. 8514–8521.

[4] "https://blog.sucuri.net/2013/07/ssh-brute-force-the-10-year-old-attack-that-still-persists.html(accessed on 15-06-2021)."

[5] Venafi, Inc., "SSH Security Vulnerability Report (Venafi, Inc.)," Tech. Rep., 2014.

[6] "https://www.fail2ban.org/wiki/index.php/main_page(accessed on 30-10-2020)."

[7] "https://www.sshguard.net/ (accessed on 31-10-2020)."

[8] Y. Wu, P. Cao, A. Withers, Z. T. Kalbarczyk, and R. K. Iyer, "Poster: Mining Threat Intelligence from Billion-scale SSH Brute-Force Attacks," in *NDSS'20: Proceedings of the Network and Distributed System Security*, 2020, pp. 1–3.

[9] R. Hofstede, L. Hendriks, A. Sperotto, and A. Pras, "SSH Compromise Detection using NetFlow/IPFIX," *ACM SIGCOMM computer communication review*, vol. 44, no. 5, pp. 20–26, 2014.

[10] 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 *BIBE'14: Proceedings of the International Conference on Bioinformatics and Bioengineering*. IEEE, 2014, pp. 379–385.

[11] D. Shmagin, "Utilizing Machine Learning Classifiers to Identify SSH Brute Force Attacks," 2019, william & Mary University.

[12] P. Khandait, N. Hubballi, and B. Mazumdar, "Efficient keyword matching for deep packet inspection based network traffic classification," in *COMSNETS'20: International Conference on COMmunication Systems NETworkS*, 2020, pp. 567–570.

[13] "http://danger.rulez.sk/projects/bruteforceblocker/(accessed on 30-10-2020)."

[14] DenyHosts, "http://denyhosts.sourceforge.net/(accessed on 30-01-2020)," 2005.

[15] PAM-ABL, "http://pam-abl.sourceforge.net/(accessed on 22-03-2021)."

[16] BlockHosts, "https://www.aczoom.com/archive-2016/blockhosts/(accessed on 30-01-2021)," 2005.

[17] M. Javed and V. Paxson, "Detecting Stealthy, Distributed SSH Brute-forcing," in *CCS '13: Proceedings of the 2013 ACM SIGSAC Conference on Computer Communications Security*. Association for Computing Machinery, 2013, pp. 85–96.

[18] C. Gates, "Coordinated Scan Detection," in *NDSS'09: In the Proceedings of 16th Annual Network and Distributed System Security Symposium*. IEEE, 2009, pp. 1–13.

[19] A. Sperotto, R. Sadre, P.-T. de Boer, and A. Pras, "Hidden Markov Model modeling of SSH brute-force attacks," in *International Workshop on Distributed Systems: Operations and Management*. Springer, 2009, pp. 164–176.

[20] L. Hellemons, L. Hendriks, R. Hofstede, A. Sperotto, R. Sadre, and A. Pras, "SSHCure: A Flow-based SSH Intrusion Detection System," in *IFIP International Conference on Autonomous Infrastructure, Management and Security*. Springer, 2012, pp. 86–97.

[21] "http://sshcure.sf.net/ (accessed on 31-10-2020)."

[22] M. M. Najafabadi, T. M. Khoshgoftaar, C. Calvert, and C. Kemp, "Detection of SSH Brute Force Attacks Using Aggregated Netflow Data," in *ICMLA'15: Proceedings of the 14th International Conference on Machine Learning and Applications*. IEEE, 2015, pp. 283–288.

[23] K. Hynek, T. Beneš, T. Čejka, and H. Kubátová, "Refined Detection of SSH Brute-Force Attackers Using Machine Learning," vol. 580, pp. 49–63, January 2020.

[24] M. D. Hossain, H. Ochiai, F. Doudou, and Y. Kadobayashi, "SSH and FTP Brute-force Attacks Detection in Computer Networks: LSTM and Machine Learning Approaches," in *ICCCS'20: Proceedings of the 5th International Conference on Computer and Communication Systems*, 2020, pp. 491–497.

[25] G. K. Sadasivam, C. Hota, and B. Anand, "Classification of SSH Attacks using Machine Learning Algorithms," in *ICITCS'16: Proceedings of the 6th International Conference on IT Convergence and Security*. IEEE, 2016, pp. 1–6.

[26] L. Sacramento, I. Medeiros, J. Bota, and M. Correia, "Flowhacker: Detecting Unknown Network Attacks in Big Traffic Data using Network Flows," in *TrustCom/BigDataSE'18: Proceedings of the 17th International Conference On Trust, Security And Privacy In Computing And Communications/12th International Conference On Big Data Science And Engineering*. IEEE, 2018, pp. 567–572.

[27] Z. He, T. Zhang, and R. B. Lee, "Machine Learning based DDoS Attack Detection from Source Side in Cloud," in *CSCloud'17: Proceedings of the 4th International Conference on Cyber Security and Cloud Computing*. IEEE, 2017, pp. 114–120.

[28] A. Satoh, Y. Nakamura, and T. Ikenaga, "Identifying User Authentication Methods on Connections for SSH Dictionary Attack Detection," in *2013 IEEE 37th Annual Computer Software and Applications Conference Workshops*, July 2013, pp. 593–598.

[29] J. P. Owens, "A Study of Passwords and Methods Used in Brute-Force SSH Attacks (CLARKSON UNIVERSITY)," Tech. Rep., 2008.

[30] R. Bezut and V. Bernet-Rollande, "Experimental Study: Study of Dictionary Attacks on SSH (University of Technology of Compiegne)," Tech. Rep., 2010.

[31] C. Valli, P. Rabadia, and A. Woodward, "Patterns and Patter - An Investigation into SSH Activity Using Kippo Honeypots," in *DigitalForensics '13: Proceedings of the 11th Australian Digital Forensics Conference*, 2013, pp. 141–149.

[32] J. Faust, "Distributed Analysis of SSH Brute Force and Dictionary Based Attacks(St. Cloud State University)," Tech. Rep., 2018.

[33] P. M. Cao, Y. Wu, S. S. Banerjee, J. Azoff, A. Withers, Z. T. Kalbarczyk, and R. K. Iyer, "CAUDIT: Continuous Auditing of SSH Servers to Mitigate Bruteforce Attacks," in *NSDI'19: Proceedings of the 16th USENIX Conference on Networked Systems Design and Implementation*. USENIX Association, 2019, pp. 667–682.

[34] A. Sharma, Z. Kalbarczyk, R. Iyer, and J. Barlow, "Analysis of Credential Stealing Attacks in an Open Networked Environment," in *NSS '10: Proceedings of the 2010 Fourth International Conference on Network and System Security*. IEEE Computer Society, 2010, pp. 144–151.

[35] A. Abdou, D. Barrera, and P. C. v. Oorschot, "What Lies Beneath? Analyzing Automated SSH Bruteforce Attacks," in *PASSWORDS'15: Proceedings of the official Passwords Conference*. Springer, 2015, pp. 72–91.

[36] P. Cao, E. Badger, Z. Kalbarczyk, R. Iyer, and A. Slagell, "Preemptive Intrusion Detection: Theoretical Framework and Real-world Measurements," in *HotSoS*

'15: Proceedings of the 2015 Symposium and Bootcamp on the Science of Security. Association for Computing Machinery, 2015, pp. 5:1–5:12.

[37] CISCO, "https://tools.cisco.com/security/center/resour ces/ssh_honeypot 27-04-2021)."

[38] P. Khandait, N. Tiwari, and N. Hubballi, "Who is Trying to Compromise Your SSH Server? An Analysis of Authentication Logs and Detection of Bruteforce Attacks," in *ICDCN '21: Adjunct Proceedings of the 2021 International Conference on Distributed Computing and Networking.* Association for Computing Machinery, 2021, pp. 127–132.

[39] "https://www.pfsense.org/(accessed on 30-10-2020)."

[40] tcpdump, "https://www.tcpdump.org/ (accessed on 30-01-2021)," 2000.

[41] M. Jonker, R. Hofstede, A. Sperotto, and A. Pras, "Unveiling Flat Traffic on the Internet: An SSH Attack Case Study," in *IM'15: IFIP/IEEE International Symposium on Integrated Network Management.* IEEE, 2015, pp. 270–278.

[42] "https://www.cs.waikato.ac.nz/ml/weka/(accessed on 20-10-2020)."

[43] Y. Tian, C. Herley, and S. Schechter, "StopGuessing: Using Guessed Passwords to Thwart Online Guessing," in *EuroSP '19: Proceedings of the 2019 Conference on Europian Security and Privacy.* IEEE, 2019, pp. 1–14.

[44] R. Hofstede, A. Pras, A. Sperotto, and G. D. Rodosek, "Flow-Based Compromise Detection: Lessons Learned," *IEEE Security Privacy*, vol. 16, no. 1, pp. 82–89, 2018.

[45] P. Velan, M. Husák, and D. Tovarňák, "Rapid prototyping of flow-based detection methods using complex event processing," in *NOMS'18: Proceedings of IEEE/IFIP Network Operations and Management Symposium*, 2018, pp. 1–3.

[46] N. Tripathi and N. Hubballi, "Application Layer Denial-of-Service Attacks and Defense Mechanisms: A Survey," *ACM Computing Surveys*, vol. 54, no. 4, pp. 1–33, 2021.

[47] K. Jensen and L. M. Kristensen, *Coloured Petri nets: Modelling and Validation of Concurrent Systems*. Springer Science & Business Media, 2009.

[48] CPNTOOLS, "http://cpntools.org/ (accessed on 30-04-2021)."

Appendix A

In this appendix we list the various elements of the Colored Petri-Net model presented in Chapter 5.
The Timed Colored Petri-Net is a nine-tuple [47] $CPN = (P, T, A, \Sigma, V, C, G, E, I)$. The elements of the proposed CPN mode are

`Places`: P= $\{P_1, P_2, \cdots, P_{31}\}$

`Transitions`: T= $\{T_1, T_2, \cdots, T_{14}\}$

`Arcs`: A= $\{A_1, A_2, \cdots, A_{69}\}$

`Type Set`: Σ= $\{INT, FLOW, STRING, STREAM, BOOL, STR\_INT, IP\_Pf, INT\_STR, IP\_RECORD, PF\_RECORD\}$

`Variable Set`: V= $\{PORT, P, V, WA, \tau_1, \tau_2, \tau_3, \tau_4, SR\_IP, PF\_VAR, OK, PF, IP\_VAR, PF\_CNT, PF\_NEW, PF\_CNT, PREV\_PF, IP\_NEW, INTERVAL, INTERVAL\_PF, INTERVAL\_IP, PREV\_INTERVAL, IP\_R, PREV\_IP, PF\_R, F\_CNT, F\_PREV, STATUS, ATTACK\_TYPE\}$

`Colour Function C`: The function C assigns colors to various places as follow. $C(P_1) = STREAM$, $C(P_2) = STREAM$, $C(P_3) = STR\_INT$, $C(P_4) = IP\_pf$, $C(P_5) = INT$, $C(P_6) = INT$, $C(P_7) = STRING$, $C(P_8) = PF\_RECORD$, $C(P_9) = INT\_STR$, $C(P_{10}) = INT\_STR$, $C(P_{11}) = INT$, $C(P_{12}) = PF\_RECORD$,

$C(P_{13}) = PF\_RECORD$, $C(P_{14}) = STRING$, $C(P_{15}) = INT\_STR$, $C(P_{16}) = IP\_RECORD$, $C(P_{17}) = INT\_STR$, $C(P_{18}) = IP\_RECORD$, $C(P_{19}) = INT$, $C(P_{20}) = IP\_RECORD$, $C(P_{21}) = INT$, $C(P_{22}) = INT$, $C(P_{23}) = INT$, $C(P_{24}) = INT$, $C(P_{25}) = INT$, $C(P_{26}) = INT$, $C(P_{27}) = INT$, $C(P_{28}) = STRING$, $C(P_{29}) = STRING$, $C(P_{30}) = INT$, $C(P_{31}) = STRING$

`Guard Function G`: $G(T_{12}) = [INTERVAL > PREV\_INTERVAL]$, $G(T_{13}) = [STATUS = "Bruteforce"]$

`Arc Expression E`: The function E assigns expressions to all the arcs as follows.

$E(A_1) = 1`(SR\_IP, PF, P, WA)$

$E(A_2) = if\ OK\ then\ empty\ else\ 1`(SR\_IP, PF, P, WA)$

$E(A_3) = 1`(SR\_IP, PF, P, WA)$

$E(A_4) = PORT$

$E(A_5) = PORT$

$E(A_6) = if\ (PORT = P)\ then\ 1'(SR\_IP, PF, WA)\ else\ empty$

$E(A_7) = 1`(SR\_IP, PF, WA)$

$E(A_8) = \tau_1$

$E(A_9) = \tau_1$

$E(A_{10}) = if\ (WA < \tau_1)\ then\ 1'(SR\_IP, PF)\ else\ empty$

$E(A_{11}) = 1`(SR\_IP, PF)$, $E(A_{12}) = findPFVariable(PF)$

$E(A_{13}) = findIPVariable(IP)$

$E(A_{14}) = PF\_VAR$

$E(A_{15}) = if(PF\_VAR = "PF1")\ then\ 1`(\#PF1PF\_R, PF\_VAR)\ else\ if\ (PF\_VAR = "PF2")\ then\ 1`(\#PF2\ PF\_R, PF\_VAR)\ \cdots\ else\ empty$

$E(A_{16}) = 1`(PF\_CNT, PF\_VAR)$

$E(A_{17}) = 1`(PF\_CNT + 1, PF\_VAR)$

$E(A_{18}) = 1`(PF\_NEW, PF\_VAR)$

$E(A_{19}) = setPF(PF\_VAR,\ INTERVAL\_PF,\ PF\_NEW,\ PF\_R)$

$E(A_{20}) = PF\_R$

$E(A_{21}) = PF\_R$

$E(A_{22}) = PREV\_PF$

$E(A_{23}) = PREV\_PF$

$E(A_{24}) = PF\_R$

$E(A_{25}) = PF\_R$

$E(A_{26}) = PF\_R$

$E(A_{27}) = INTERVAL\_PF$

$E(A_{28}) = setInterval(INTERVAL\_PF)$

$E(A_{29}) = IP\_VAR,\ E(A_{30}) = PREV\_PF$

$E(A_{31}) = if\ (IP\_VAR = "IP1")\ then\ 1`(\#IP1\ IP\_R, IP\_VAR)\ else\ if\ (IP\_VAR = "IP2")\ then\ 1`(\#IP2\ \ IP\_R, IP\_VAR)\ \ \cdots\ \ else\ \ empty,\ \ E(A_{32}) = 1`(IP\_CNT, IP\_VAR)$

$E(A_{33}) = 1`(IP\_CNT + 1, IP\_VAR)$

$E(A_{34}) = 1`(IP\_NEW, IP\_VAR)$

$E(A_{35}) = setIP(IP\_VAR, INTERVAL\_IP, IP\_NEW, IP\_R),\ E(A_{36}) = IP\_R$

$E(A_{37}) = IP\_R$

$E(A_{38}) = setInterval(INTERVAL\_IP)$

$E(A_{39}) = INTERVAL\_IP$

$E(A_{40}) = IP\_R$

$E(A_{41}) = PREV\_IP$

$E(A_{42}) = IP\_R$

$E(A_{43}) = IP\_R$

$E(A_{44}) = PREV\_IP$

$E(A_{45}) = PREV\_IP$

$E(A_{46}) = F\_PREV$

$E(A_{47}) = setInterval(INTERVAL)$

$E(A_{48}) = INTERVAL$

$E(A_{49}) = INTERVAL$

$E(A_{50}) = INTERVAL$

$E(A_{51}) = setPrevInterval(INTERVAL)$

$E(A_{52}) = F\_PREV$, $E(A_{53}) = F\_CNT$

$E(A_{54}) = F\_CNT$

$E(A_{55}) = setRequest(F\_CNT, INTERVAL)$

$E(A_{56}) = PREV\_INTERVAL$

$E(A_{57}) = F\_PREV$

$E(A_{58}) = F\_PREV$

$E(A_{59}) = \tau_2$

$E(A_{60}) = \tau_2$

$E(A_{61}) = if\ (F\_PREV > \tau_2)\ then 1`("Brute - force")\ else 1`("Normal")$,

$E(A_{62}) = 1`STATUS$

$E(A_{63}) = \tau_3$

$E(A_{64}) = \tau_3$

$E(A_{65}) = checkIP(\#IP1\ IP\_R, \#IP2\ IP\_R, \cdots, \#IPn\ IP\_R, \tau_3)$

$E(A_{66}) = 1`ATTACK\_TYPE$

$E(A_{67}) = \tau_4$

$E(A_{68}) = \tau_4$

$E(A_{69}) = checkpf(\#PF1PF\_R, \#PF2PF\_R, \cdots, \#PFnPF\_R, \tau_4, ATTACK\_TYPE)$

`Initialization Function I`: The function I assigns initialisation expression to various places as follows.

$I(P_1) = 1`(ip1, pf1, p1, wa1)@ts1 + +1`(ip2, pf2, p2, wa2)@ts2 + +1`(ip3, pf3, p3, wa3)@ts3 + + \cdots$

$I(P_3) = 1'(22)@0$

$I(P_5) = 1'(th1)@0$

$I(P_8) = 2'(PF1 = 0, PF2 = 0, \cdots, PFn = 0)@0$

$I(P_{12}) = 1'(PF1 = 0, PF2 = 0, \cdots, PFn = 0)@0$

$I(P_{13}) = 1'(1)@0$

$I(P_{16}) = 2'(IP1 = 0, IP2 = 0, \cdots, IPn = 0)@0$

$I(P_{19}) = 1'(IP1 = 0, IP2 = 0, \cdots, IPn = 0)@0$

$I(P_{20}) = 1'(1)@0$

$I(P_{22}) = 1'(1)@0$

$I(P_{24}) = 1'(0)@0$

$I(P_{25}) = 1'(0)@0$

$I(P_{26}) = 1'(th2)@0$

$I(P_{28}) = 1'(th3)@0$

$I(P_{30}) = 1'(th4)@0$

Table 1 shows the list of arcs in the Coloured Petri-Net model. Each arc is associated with an input state or transition and the output state or a transition. For example, the first entry in the table shows an arc with the input state P_1 with the output transition T_1, with an arc expression function $1`(SR\_IP, PF, P, WA)$ and colour function of the state P_1 as STREAM.

Table 1: Arcs in the Petri-Net model

Input State/ Transition	**Output State/ Transition**	**Arc Expression Function**	**Colour**
P1	T1	$1`(SR\_IP, PF, P, WA)$	STREAM
T1	P2	$if\ (OK)\ then\ empty\ else\ 1`(SR\_IP, PF, P, WA)$	STREAM
P2	T2	$1`(SR\_IP, PF, P, WA)$	STREAM
T2	P3	$PORT$	INT
P3	T2	$PORT$	INT
T2	P4	$if\ (PORT = P)\ then\ 1`(SR\_IP, PF, WA)\ else\ empty$	STR_INT
P4	T3	$1`(SR\_IP, PF, WA)$	STR_INT

Continued on next page

Table 1 – continued from previous page

Input State/ Transition	Output State/ Transition	Arc Expression Function	Colour
T3	P5	$\tau 1$	INT
P5	T3	$\tau 1$	INT
T3	P6	$if(WA < \tau 1)\ then\ 1`(SR\_IP, PF)\ else\ empty$	IP_pf
P6	T4	$1`(SR\_IP, PF)$	IP_pf
T4	P7	$findPFVariable(PF)$	STRING
T4	P8	$findIPVariable(SR\_IP)$	STRING
P7	T5	PF_VAR	STRING
T5	P9	$if\ (PF\_VAR = "PF1")\ then\ 1`(\#PF1\ PF\_R, PF\_VAR)\ else\ if\ (PF\_VAR = "PF2")\ then\ 1`(\#PF2\ PF\_R, PF\_VAR) \cdots else\ empty$	INR_STR
P9	T6	$1`(PF\_CNT, PF\_VAR)$	INT_STR
T6	P10	$1`(PF\_CNT + 1, PF\_VAR)$	INT_STR
P10	T7	$1`(PF\_NEW, PF\_VAR)$	INT_STR
T7	P11	$setPF(PF\_VAR, INTERVAL\_PF, PF\_NEW, PF\_R)$	PF_RECORD
P11	T7	PF_R	PF_RECORD
P11	T5	PF_R	PF_RECORD
T7	P12	$PREV\_PF$	PF_RECORD
T7	P13	PF_R	PF_RECORD
T7	P14	$setInterval(INTERVAL\_PF)$	INT
P13	T7	$PREV\_PF$	PF_RECORD

Continued on next page

Table 1 – continued from previous page

Input State/ Transition	**Output State/ Transition**	**Arc Expression Function**	**Colour**
P14	T7	$INTERVAL\_PF$	INT
P12	T8	$PREV\_PF$	PF_RECORD
P13	T14	PF_R	PF_RECORD
T14	P13	PF_R	PF_RECORD
P8	T8	IP_VAR	STRING
T8	P15	$if\ (IP\_VAR = "IP1")\ then$ $1`(\#IP1\ IP\_R, IP\_VAR)\ elseif$ $(IP\_VAR = "IP2")\ then$ $1`(\#IP2\ IP\_R, IP\_VAR) \cdots else\ empty$	INT_STR
P15	T9	$1`(IP\_CNT, IP\_VAR)$	INT_STR
T9	P16	$1`(IP\_CNT + 1, IP\_VAR)$	INT_STR
P16	T10	$1`(IP\_NEW, IP\_VAR)$	INT_STR
T10	P17	$setIP(IP\_VAR, INTERVAL\_IP,$ $IP\_NEW, IP\_R)$	IP_RECORD
P17	T10	IP_R	IP_RECORD
P17	T8	IP_R	IP_RECORD
T10	P18	$setInterval(INTERVAL\_IP)$	INT
P18	T10	$INTERVAL\_IP$	INT
T10	P19	IP_R	IP_RECORD
P19	T10	$PREV\_IP$	IP_RECORD
P19	T13	IP_R	IP_RECORD
T13	P19	IP_R	IP_RECORD
T10	P20	$PREV\_IP$	IP_RECORD

Continued on next page

Table 1 – continued from previous page

Input State/ Transition	**Output State/ Transition**	**Arc Expression Function**	**Colour**
P20	T11	$PREV\_IP$	IP_RECORD
T11	P21	F_PREV	INT
T11	P22	$setInterval(INTERVAL)$	INT
P22	T11	$INTERVAL$	INT
T11	P23	$setPrevInterval(INTERVAL)$	INT
T11	P24	F_CNT	INT
P24	T11	F_PREV	INT
T11	P25	$setRequest(F\_CNT, INTERVAL)$	INT
P25	T11	F_CNT	INT
P22	T12	$INTERVAL$	INT
T12	P22	$INTERVAL$	INT
P23	T12	$PREV\_INTERVAL$	INT
P24	T12	F_CNT	INT
T12	P24	F_CNT	INT
P26	T12	$\tau 2$	INT
T12	P26	$\tau 2$	INT
T12	P27	$if(F\_PREV > \tau 2)\ then\ 1`(" Bruteforce")$ $else\ 1`(" Normal")$	STRING
P27	T13	$1`(STATUS)$	STRING
P28	T13	$\tau 3$	INT
T13	P28	$\tau 3$	INT
T13	P29	$checkIP(\#IP1\ \ IP\_R, \#IP2\ \ IP\_R, \cdots, \#IPn\ IP\_R, \tau 3)$	STRING

Continued on next page

Table 1 – continued from previous page

Input State/ Transition	**Output State/ Transition**	**Arc Expression Function**	**Colour**
P29	T14	1‘$(ATTACK\_TYPE)$	STRING
T14	P30	τ 4	INT
P30	T14	τ 4	INT
T14	P31	$checkPF(\#PF1\,PF\_R, \#PF2\,PF\_R, \cdots,$ $\#PFn\,PF\_R, \tau 4, ATTACK\_TYPE)$	STRING

Publications

1. Pratibha Khandait, Namrata Tiwari and Neminath Hubballi, "Who is Trying to Compromise Your SSH Server ? An Analysis of Authentication Logs and Detection of Bruteforce Attacks", The 4th International Workshop On Networking Women in Distributed Computing And Networks (NWDCN 2021) held along with ICDCN 2021, Pages 127–132, Nara japan.

2. Neminath Hubballi, Namrata Tiwari and Pratibha Khandait,"Distributed SSH Bruteforce Attack Detection with Flow Content Similarity and Login Failure Reputation", 15th ACM ASIA Conference on Computer and Communications Security (AsiaCCS 2020 Posters), Pages 916–918, Taipei, Taiwan.

3. Namrata Tiwari and Neminath Hubballi, "Secure Socket Shell Bruteforce Attack Detection with Petri-Net Modeling", IEEE Transactions on Network and Service Management (Under Review)