

Detection of Malicious SSH Sessions: A Machine Learning Approach



By

Hamid Mujtaba Khalil

Fall 2018-MS(IS) - 00000274017

Supervisor

Dr. Hasan Tahir

Department of Computing

A thesis submitted in partial fulfillment of the requirements for the degree
of Masters of Science in Information Security (MS IS)

In

School of Electrical Engineering and Computer Science,
National University of Sciences and Technology (NUST),

Islamabad, Pakistan.

(December 2021)

Approval

It is certified that the contents and form of the thesis entitled "Detection of Malicious SSH Sessions: A Machine Learning Approach" submitted by HAMID KHALIL have been found satisfactory for the requirement of the degree

Advisor : Dr. Hasan Tahir

Signature:  _____

Date: 01-Dec-2021

Committee Member 1:Dr. Sana Qadir

Signature:  _____

02-Dec-2021

Committee Member 2:Dr. Razi Arshad

Signature:  _____

Date: 01-Dec-2021

Signature: _____

Date: _____


Dedication

I dedicate this work to my **parents** for all their sacrifices in life, their trust in my decisions and giving me the freedom to create my own life.

Certificate of Originality

I hereby declare that this submission titled "Detection of Malicious SSH Sessions: A Machine Learning Approach" is my own work. To the best of my knowledge it contains no materials previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any degree or diploma at NUST SEECS or at any other educational institute, except where due acknowledgement has been made in the thesis. Any contribution made to the research by others, with whom I have worked at NUST SEECS or elsewhere, is explicitly acknowledged in the thesis. I also declare that the intellectual content of this thesis is the product of my own work, except for the assistance from others in the project's design and conception or in style, presentation and linguistics, which has been acknowledged. I also verified the originality of contents through plagiarism software.

Student Name: HAMID KHALIL

Student Signature: 

THESIS ACCEPTANCE CERTIFICATE

Certified that final copy of MS/MPhil thesis entitled "Detection of Malicious SSH Sessions: A Machine Learning Approach" written by HAMID KHALIL, (Registration No 00000274017), of SEECs has been vetted by the undersigned, found complete in all respects as per NUST Statutes/Regulations, is free of plagiarism, errors and mistakes and is accepted as partial fulfillment for award of MS/M Phil degree. It is further certified that necessary amendments as pointed out by GEC members of the scholar have also been incorporated in the said thesis.

Signature: _____  _____

Name of Advisor: Dr. Hasan Tahir

Date: _____ **01-Dec-2021** _____

Signature (HOD): _____

Date: _____

Signature (Dean/Principal): _____

Date: _____

Acknowledgment

First of all, I would like to express my sincere gratitude to my supervisor, Dr. Hasan Tahir, for believing in me and his continuous guidance and support from inception of this idea to its completion. This work would not have been possible without him.

I am also thankful to my Guidance and Evaluation Committee, Dr. Sana Qadir and Dr. Razi Arshad for providing valuable insights and constructive feedback during this research work.

Last but not the least, I would like to thank my wife Ammara, my sister Madeeha and my friends for continuously pushing me to get this done.

Table of Contents

1	Introduction	1
1.1	Traditional SSH Security Controls	2
1.1.1	Using strong passwords	2
1.1.2	Moving SSH listeners from default to a random port	2
1.1.3	Using public-key based authentication	3
1.1.4	Firewall to allow-list IPs from which node can be accessed	4
1.2	Attacker Models	4
1.2.1	Malicious Insider	4
1.2.2	Outside Attacker	6
1.3	Problem Statement	7
1.4	Solution Definition/Description	7
1.5	Thesis Motivation	8
1.6	Thesis Contribution	8
1.7	Thesis Organization	9
1.8	Summary	10
2	Literature Review	11
2.1	What is Secure Shell (SSH)?	11

TABLE OF CONTENTS

2.2	Attacks on SSH	12
2.2.1	Brute-Force Attacks	12
2.2.2	Keystroke Timing Attacks on SSH	14
2.2.3	Man-in-the-Middle (MITM) Attacks on SSH	14
2.3	Post-Quantum SSH Security	14
2.4	Can SSH Honeypots Help?	14
2.4.1	Usecase #1: Using SSH Honeypots for Attacker Profiling	15
2.4.2	Usecase #2: Using SSH Honeypots for Botnet Detection	15
2.4.3	Usecase #3: Using SSH Honeypots for Detetcion of Malicious SSH Sessions	16
2.5	Summary	16
3	Research Methodology	18
3.1	Steps involved in the carried out research	18
3.1.1	Data Collection & Parsing	19
3.1.2	Data Cleaning	20
3.1.3	Exploratory Data Analysis (EDA)	20
3.1.4	Feature Engineering	21
3.1.5	Model Evaluation and Tuning	21
3.2	Tools & Technologies Used	22
3.3	Summary	23
4	Data Collection	24
4.1	Benign Data Collection	24
4.2	Malicious Data Collection	25
4.3	Data Parsing	27

TABLE OF CONTENTS

4.4	Data Summary	30
5	Exploratory Data Analysis & Feature Engineering	31
5.1	System Exploratory Commands	34
5.2	Deletion Commands	35
5.3	Echo File Writes	35
5.4	Difference in IP and URL Based Downloads	36
5.5	Directory Navigation Commands	37
5.6	Access Manipulation Commands	37
5.7	Feature Selection	38
5.8	Summary	42
6	Model Training & Evaluation	43
6.1	Evaluation Metrics	43
6.1.1	Accuracy	44
6.1.2	True Positive Rate (TPR)	44
6.1.3	True Negative Rate (TNR)	44
6.1.4	False Positive Rate (FPR)	44
6.1.5	False Negative Rate (FNR)	45
6.2	Model Selection	45
6.3	Initial Results	45
6.4	K-Fold Cross Validation with Hyper-Parameters Tuning	46
6.5	Discussion	50
7	Conclusion & Future Work	51
7.1	Concluision	51

TABLE OF CONTENTS

7.2	Future Work	52
7.2.1	An Improved Approach for Benign Data Collection . . .	52
7.2.2	Generalizing for Other Linux and IOT Shells	53
7.2.3	Training Models for Specialized Services / Dedicated Nodes	53
7.3	Summary	54
	Bibliography	55

List of Tables

3.1	Tools technologies used during this research work	22
4.1	Dataset summary	30
5.1	Extracted features based on domain knowledge	39
5.2	Finalized features after extensive EDA processing	41
6.1	Initial results of different classifiers	46
6.2	RF results before and after model tuning	49

List of Figures

1.1	Attacker models	5
3.1	Research methodology flowchart	19
4.1	Deployed honeypot architecture	26
5.1	Word cloud for malicious data	32
5.2	Frequency of highly used words in malicious data	32
5.3	Word cloud for benign data	33
5.4	Frequency of highly used words in benign data	33
5.5	System exploratory commands example from a malicious session	34
5.6	Echo file write example from a malicious session	35
5.7	IP based downloads example from a malicious session	36
5.8	Access manipulation commands example from a malicious ses- sion	37
5.9	Correlation matrix for features	41
6.1	K-Fold Cross Validation for $K = 3$	48
6.2	Confusion matrix for RF after tuning	49

Abstract

Cloud computing has enabled organizations to run their workloads on multi-node clusters in different private and public cloud service providers (CSPs). Most nodes run some distribution of Linux which is accessed through Secure Shell (SSH). The infrastructure is not only accessed by the engineering team members, but also by automated scripts and bots that help manage those machines. This study formulates a machine learning based technique to classify those SSH sessions into Malicious and Benign by solely using the commands executed in the shell. Thus, this research will help identify any malicious insider in an engineering team or a compromised automation script or bot that was written to help manage that infrastructure. This study also provides a capability to help reduce the damage done by those malign entities by timely notifying the security personnel.

Chapter 1

Introduction

Cloud computing is the on-demand provisioning of services like storage, orchestration systems, infrastructure, virtualized network functions and many other services over the internet. With the adaptation of cloud computing, most organizations have moved their workloads from on premise implementation to their choice of cloud service providers (CSPs). This trend has shown growth over the past few years and is anticipated to grow in the upcoming years as well due to the ease of infrastructure management provided by the cloud. Having the ability to procure on-demand Infrastructure-as-a-Service (IaaS) on the cloud to scale workloads as per the needs across multiple machines has enabled organizations to run their workloads on multi-node clusters. More than 95% of those nodes are running some flavor or distribution of Linux [1]. These nodes are not only accessed by the members of the engineering and security teams for the management of the node itself, but also by the bots and scripts responsible for maintaining the Continuous-Integration Continues-Delivery (CICD) pipelines of the workload instance running on

CHAPTER 1. INTRODUCTION

top of it. While having your workloads publicly accessible over the internet addresses the business needs, it also raises the questions of adequacy of the security controls implemented to secure the infrastructure those workloads are executing on.

Secure Shell (SSH) [2] is the protocol which sanctions the remote shell access of Linux based machines, and it is also one of the highly used protocols for management of Linux based cloud infrastructure. Traditional security controls implemented via secure SSH involve some approaches are described below.

1.1 Traditional SSH Security Controls

1.1.1 Using strong passwords

A recent research shows that password based SSH authentication is still being used at more than 65% of the publicly available machines on the internet [3]. While having the strongest of passwords does help, but this security control is still susceptible to brute forcing, dictionary attacks and password reuse with a combination of data breaches. Also, with the continuously increasing computing power day by day, the password space is narrowing down.

1.1.2 Moving SSH listeners from default to a random port

SSH by default runs on port 22 but moving this listening port to any other random port is implemented sometimes to confuse attackers into believing

CHAPTER 1. INTRODUCTION

that SSH is not enabled on the device. This trick has worked for a time but now it is seen that during the initial reconnaissance process, the bots that are constantly attacking the publicly accessible infrastructure perform scans for all open ports within the 1-65535 range. These bots have evolved over the years and they do try SSH based attacks on all open ports to see responses and correlate the response with the SSH protocol specific responses. This evolution of the attack bots have rendered this security control insufficient.

1.1.3 Using public-key based authentication

This is one of the highly used approach where password-based authentication while establishing a SSH connection is completely disabled. Instead, only public key based authentication is allowed with only a few known and white-listed public keys placed in the authorized keys file within the `/.ssh` directory on the remote node. This allows SSH connections to be made only from the entities owning the corresponding private keys to the public keys already added in the authorized keys file on the node. While this approach provides satisfactory level of security, it is still not enough. The concerns with this approach involve: Private keys can be compromised with or without the knowledge of the owner and the window between the credentials compromise and the owner's realization could very well be the attack window; This approach also opens up a whole new issue of keeping the updated keys on the node, removal of stale public keys from the authorized keys file and keys management in general.

1.1.4 Firewall to allow-list IPs from which node can be accessed

Another security control is to allow SSH access to the node from only an allow-listed set of IPs or IPs based on geolocation. This approach employs the use of some kind of host or network based firewall to achieve this objective. This technique, when used in combination with any of the above mentioned controls, provides adequate amount of security. But, it still does not shield us from a scenario where an attacker gets ahold of the access credentials and also has access to the allow-listed IPs, which is typically a case of a malicious insider.

1.2 Attacker Models

Let us define the attacker models that are basically able to bypass every single one of the SSH security controls mentioned previously, graphically represented in Figure 1.1. Let's assume an organization has its workload W1 running on a public cloud on a multi-node cluster consisting of three nodes N1, N2 and N3. Assuming the organization has enabled all four of the security controls on all of the nodes, the following attacks will still be successful but will also go unnoticed:

1.2.1 Malicious Insider

As described in the graphical representation above, a malicious insider will have legitimate access to the N1, N2 and N3 cluster nodes with the appro-

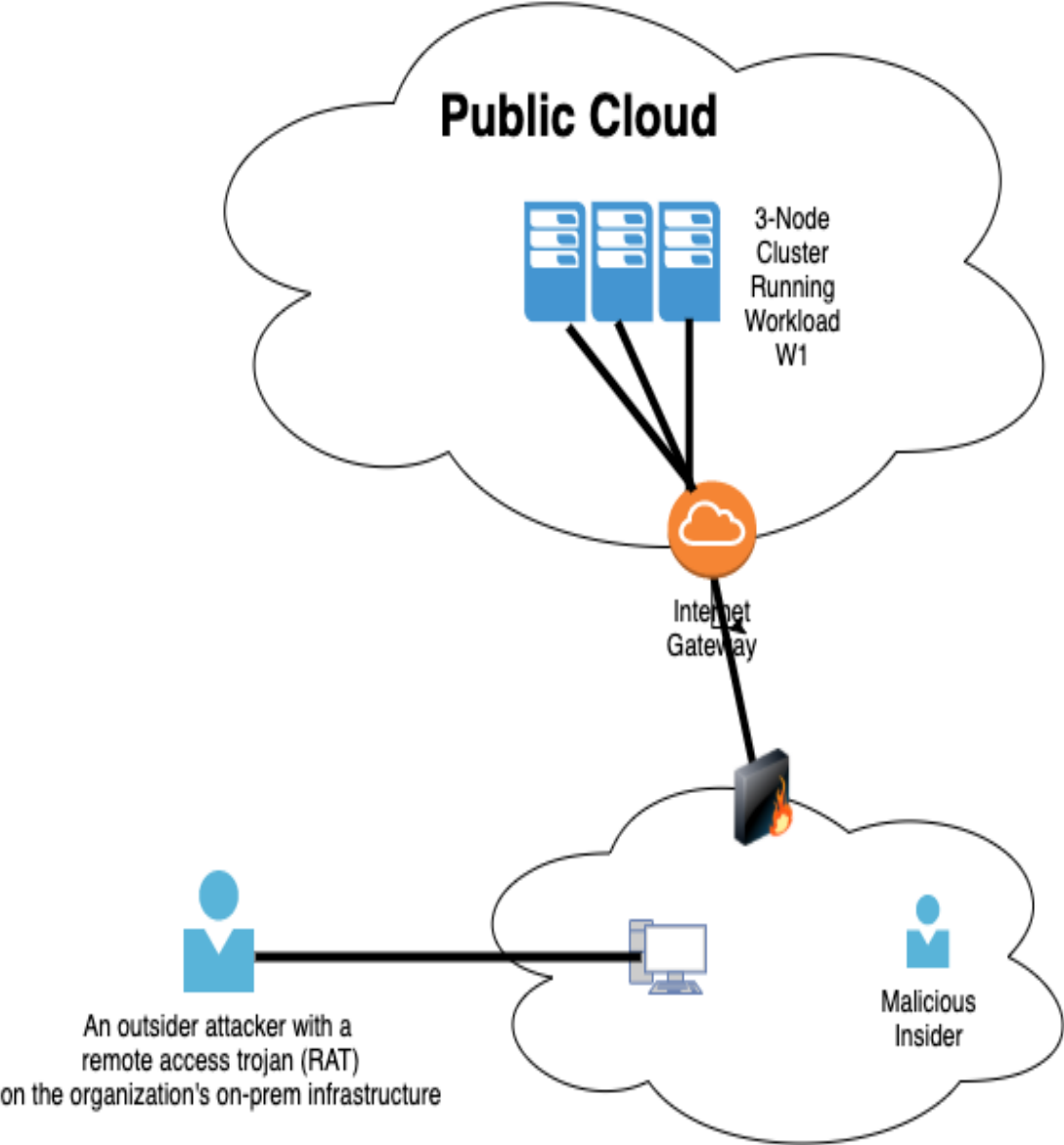


Figure 1.1: Attacker models

priate credentials to access each node separately.

1.2.2 Outside Attacker

Another attack model represented is an outsider who has a Remote Access Trojan (RAT) tool installed on one of the compromised nodes in organization's on-premise infrastructure. Now if that compromised node was an access point to the cluster nodes in cloud by the organization's team then it will have an authorized private key whose corresponding public key is already added in the authorized keys file in the cluster nodes. Also, since this compromised node resides in the organization's own network, therefore all the geolocation-based firewalls will be bypassed as all the attacks by the attacker will be proxied from this compromised node. This gives the attacker free access to cluster nodes and with the ability to perform any hostile actions on the cluster nodes in cloud.

We have established by now that these traditional security controls typically employed to secure SSH on the cloud are not ample. However, if we detect every malicious session based on the commands entered on the shell, we would have the capability to actually remediate all of the previously mentioned attack models. This can be achieved by classifying every SSH session into malicious or benign in real-time with the help of machine learning. On identification of a malicious session, a variety of appropriate actions could be performed like generating alerts for the incident response team, termination of the session and blacklisting the session access credentials to help control the damage.

1.3 Problem Statement

The use of SSH in the cloud along with the 22% of the security incidents involving an insider has questioned the security posture of the cloud infrastructure for organizations [4]. Based on this identified problem the following problem statement accurately describes the thesis motivation and the solution proposed:

“Cloud infrastructure based on Linux is widely accessed by SSH protocol. Traditional methods to secure the SSH are insufficient against a malicious insider or a compromised set of credentials. This project will explore the efficacy of detecting malicious SSH sessions with the help of machine learning by purely engineering features using the domain knowledge from the commands entered on the shell.”

1.4 Solution Definition/Description

The research proposes the following solution:

- Detection of malign SSH sessions using only the commands entered on shell.
- Engineering light-weight features by purely relying on domain-knowledge.
- Generate high accuracy non-intrusive models to detect malicious sessions in real time.

1.5 Thesis Motivation

Although many studies have been conducted that focus on botnet detection using the SSH based honeypots, but to the best of our knowledge, we have found that no previous work has been done to detect the malicious SSH sessions using the commands entered on the shell except for [5]. The lack of the study in the domain along with the abrupt adoption of cloud by the organizations presents a severe threat against the attacker model presented earlier in this study. Thus, the motivation of this research is to show a novel approach for the detection of malicious SSH sessions using machine learning by purely relying on the domain knowledge for feature engineering. The results of this work have been compared with the only previous work [5].

1.6 Thesis Contribution

This research work makes the following novel contributions:

- Establishment of a custom dataset by compiling the extracted malicious sessions from [6] and the non-malicious SSH commands history of actual users from Github Search API [7].
- To the best of our knowledge, this is the first research that explores the Cybernet honeypot dataset [6].
- Identification of malicious SSH sessions by extracting features from the commands executed on the shell by purely relying on domain knowledge.

CHAPTER 1. INTRODUCTION

- Training multiple classifiers with the extracted features to show the efficacy of domain knowledge in identification of malicious sessions.
- Provide a capability to detect malicious insiders or a compromised set of credentials.
- Unlike the work previously done, the proposed methodology can not be evaded easily by the attacker.

1.7 Thesis Organization

The following is an outline of the thesis structure. Chapter 2 provides a review of the literature on the core ideas related to this thesis. The research methodology along with the tools and technologies used during the research have been discussed in Chapter 3. Complete details of malicious and benign data collection are discussed separately in the Chapter 4 and also a summary of the final dataset is also provided in the same chapter. The exploratory data analysis (EDA), feature engineering and feature selection process is discussed in Chapter 5. In chapter 6, we start by defining our evaluation metrics followed by the model training and evaluation. We also discuss the model tuning using K-fold Cross-Validation in Chapter 6 and provide a comparison of our results with the previously published work. Finally, we conclude this research in chapter 7 and also discuss potential future work that can be carried out on basis of this research.

1.8 Summary

This chapter led the foundation of the cloud based infrastructure access and the challenges related to that access. It also highlighted how the traditional security controls are not sufficient. It also provides an overview of the thesis goals and scope, the research work's main objectives, and overall thesis organization. In the upcoming chapter, we will look at the literature review that has been conducted for this thesis.

Chapter 2

Literature Review

In this chapter, the important research work linked to this study are discussed in this chapter. A concerted effort has been made to cover the most recent studies completed in recent years. In addition, additional research has been incorporated to help the reader have a better understanding of the suggested strategy as they read through this thesis.

2.1 What is Secure Shell (SSH)?

Secure shell [2] is a network protocol that allows open a secure remote terminal and execute commands in its most basic form. However, the protocol can also be used for file transfer (SCP, SFTP); as an encrypted transport protocol for other potentially less secure protocols; and for port forwarding, which is a technique for allowing machines on opposite sides of a firewall to communicate by redirecting requests for specific ports to ports on other servers.

2.2 Attacks on SSH

Research has shown that SSH is among one of the highly attacked services on the exposed hosts on public clouds [8]. The SSH attack types involve port scanning and brute-force attacks. In SSH port scanning attacks, attacking entities perform scans for all the open ports and correlate the responses to guess the SSH port of the target host. However, brute force attacks are the most common type of attacks on SSH target hosts.

2.2.1 Brute-Force Attacks

Brute-force attacks are performed in a variety of ways. In this attack type, an attacking entity performs brute force on access credentials of the target host across the complete password space. Sometimes brute force attack is also performed using a large dictionary of already stolen username and password sets, called the dictionary attack. SSH brute force can also be sometimes performed using a limited set of username password credential sets when the attacker is trying to guess the password, called the guessing attack. The sole purpose of these attacks are to acquire a valid set of SSH credentials to the underlying host. Different types of brute-force attack mitigation techniques involving flow-record analysis, SSH log analysis and machine learning based approaches have been published by researchers. In this section, we present the recently published research in the SSH attack mitigation domain.

- One of the approach that uses Long Short Term Memory (LSTM), a deep learning model based SSH brute force attack detection is [9]. It uses the infamous CICIDS2017 dataset [10].

CHAPTER 2. LITERATURE REVIEW

- Most of the techniques easily detect the aggressive amount of of SSH guessing attacks, which forced attackers to come up with novel ideas. One approach adopted by the attackers is to run very slow guessing attacks i.e. one password guessing attempt in a span of an hour, a day or a week. An approach to detect such attacks is described in [11].
- Even unsuccessful brute-force attacks are costly for the target in terms of resources as a lot of the system resources are wasted which could have been useful for legitimate users. The scarcity of these resources is much higher in IoT devices. A study quantifies the resources acquired by an intrusion detection system (IDS) as it detects and mitigates a SSH brute force attacks in IoT devices [12].
- In order to infiltrate a system, adversaries employ a variety of user name and password combinations in SSH brute forcing assaults. Because such operations are easily detectable in log files, sophisticated attackers spread brute-force attacks over a wide range of origins. However, effectively locating such dispersed efforts proved to be a difficult task. In practise, when attackers distribute brute-force attacks over several sources, they would most likely reuse the same software across all of them to streamline their operations and save money. This means that if we can figure out what tools were used in these attempts, we can group similar tool usage together. A study focused on fingerprinting the publicly available and custom developed SSH brute force tooling is [13].

2.2.2 Keystroke Timing Attacks on SSH

SSH protocol by default, sends the typed key information immediately to the remote server. This behaviour of the protocol has been exploited in the past by analyzing keystroke timings over SSH connections to gather user or password related information [14].

2.2.3 Man-in-the-Middle (MITM) Attacks on SSH

A recent study explores how SSH key exchange process can result into a possible Man-in-the-Middle attack on Raspberry Pi 2 Model B [15].

2.3 Post-Quantum SSH Security

With the practicality of quantum computers becoming clearer day by day, researchers have started evaluating different protocols' cryptography in possible presence of a quantum adversary. Such an effort for key exchange mechanism of SSH protocol has been published [16]. Another research that evaluates the post quantum cryptographic overhead in the SSH and TLS protocols is [17].

2.4 Can SSH Honeypots Help?

SSH honeypot is an emulated environment that accepts SSH connections and acts as close to a real system as possible. SSH honeypots are decoys that imitate an actual system to lure attackers into revealing their own or the attack related information. All of the prior mentioned approaches are effective before an attack has succeeded. Think of the possibilities of all the

CHAPTER 2. LITERATURE REVIEW

sabotages an attacker can perform given an SSH session with a malicious intent has been established. This establishment of SSH session could have been resulted by either a successful brute force attack, a malicious insider or an attacker who has gained access to legitimate target credentials through a spyware or other tools and techniques. The attacker model presented in the previous section in Figure 1.1 can be referred here for the graphical illustration. SSH honeypots can help in controlling such uneventful scenarios.

The tool at hand in this scenario is the commands entered on the shell which can be utilized isolating SSH sessions with malicious intent from benign.

2.4.1 Usecase #1: Using SSH Honeypots for Attacker Profiling

A study that uses SSH honeypots for post-compromise attacker profiling using the commands and utilizing the domain knowledge of shell commands is [18]. Another interesting approach that employs Bash commands for attacker profiling is [19].

2.4.2 Usecase #2: Using SSH Honeypots for Botnet Detection

Another study focusing detection of SSH botnet infection is [20]. It uses commands executed on the shell along with other network features to detect SSH botnet infection stage. Again, there is not much domain knowledge extracted from the set of commands to build features.

2.4.3 Usecase #3: Using SSH Honeypots for Detection of Malicious SSH Sessions

While there are plenty of researches done employing SSH based honeypots in one way or another, but to the best of our knowledge, there is only one study that is directed towards detection of malicious SSH sessions using machine learning [5]. While it addresses the problem in the right fashion with a smart approach of data collection and classification, we argue that there is still room for improvement. First of all, it employs N-grams as features which is a Natural Language Processing (NLP) based technique. We can make use of domain knowledge to extract and curate features which could be simpler while producing as good results. Secondly, this research employs K-Nearest-Neighbors (KNN) for classification which is a memory hungry algorithm as it keeps all the training data in memory. This trait of the model renders into a directly proportional relationship between the training data and the prediction time as well as the training data and memory used, which makes the model a bit intrusive on the node when deployed in production. Another study that aims at detecting malicious shell commands and binaries of IoT devices is [21]. This study also uses NLP based term and character-level features.

2.5 Summary

In this section, we presented the possible usages of SSH protocol, the common attacks on SSH and their mitigation techniques. Apart from the preemptive

CHAPTER 2. LITERATURE REVIEW

techniques to detect an SSH attack, we also discussed the published research that can tell a malicious SSH session from a benign one after it has been established. We also discussed the imperfections in the already published work and the ways it can be improved.

Chapter 3

Research Methodology

In this section we present the methodology employed during this research work. We will discuss the steps in sequence that were carried out to reach the desired outcomes. Since this research can be categorized as a machine learning classification problem, the steps involved include all the processes involved in a typical machine learning pipeline including data collection parsing, data cleaning, exploratory data analysis (EDA), feature engineering and model evaluation and tuning. The complete process is graphically represented in the Figure 1.1. After the process description, this section will also provide details about the tools and technologies that will be used in this research work.

3.1 Steps involved in the carried out research

Research methodology used is graphically represented in Figure 3.1.

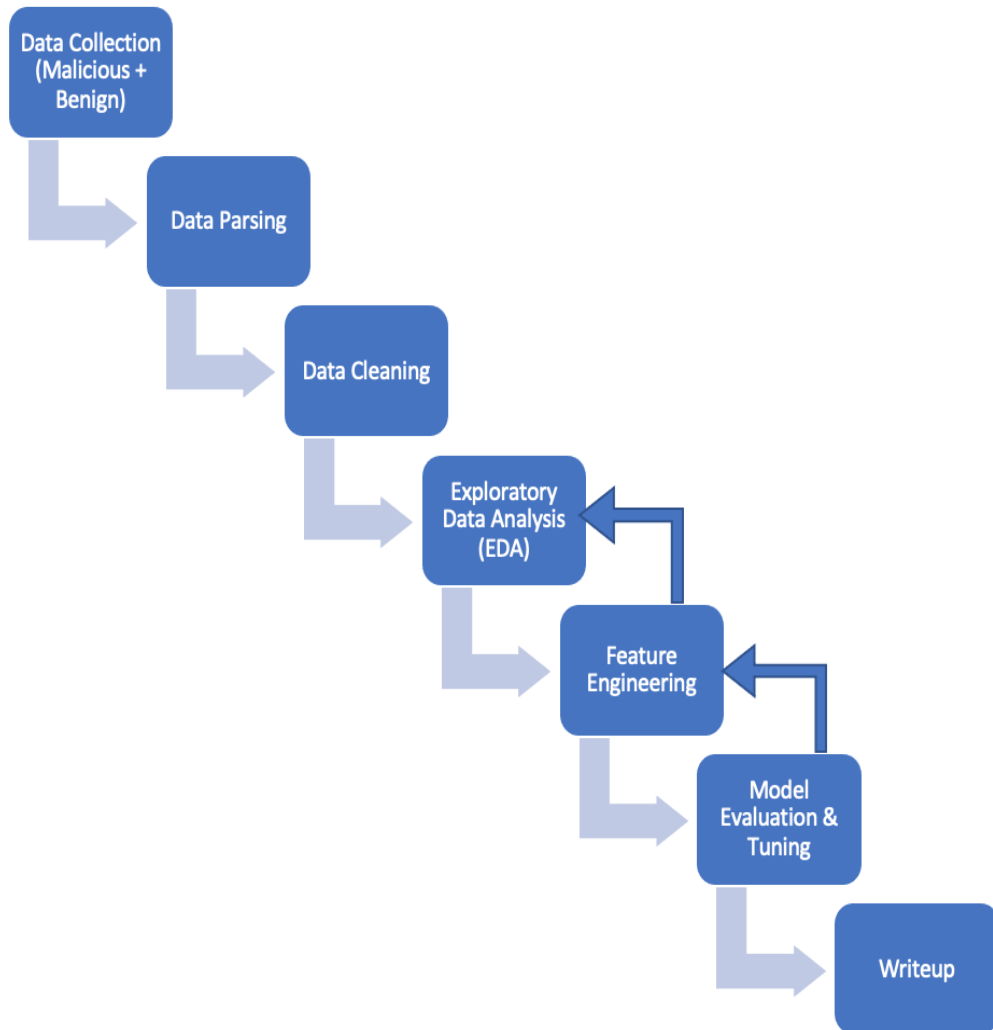


Figure 3.1: Research methodology flowchart

3.1.1 Data Collection & Parsing

This process involves collection of malicious and benign data. We will collect both data classes and compile it into a process-able format.

3.1.2 Data Cleaning

During this process data is cleaned to make sure it is free from out-liars and observations that could negatively impact the final results. A number of techniques including removal of empty and duplicate entries are performed in this process. It is to be noted that for our specific case, we will not need the data labelling process that is a step usually followed by the data cleaning process. The data collected from the benign sources will be automatically tagged as the benign data and the data collected from the malicious sources will automatically be tagged as malicious data.

3.1.3 Exploratory Data Analysis (EDA)

Datasets range from a few hundred records to millions of records with each record containing attributes up to thousands. Therefore, it is very hard to look at the numbers if the dataset is numerical or the millions of lines of text if the dataset is textual and make sense out of this. Datasets are very hard to understand in their original form. Exploratory data analysis is a process used to make inference and generate insights of the data. During this research step, data is converted into different statistical and graphical representations which better reflect the data state. This step is usually crafted according to the dataset at hand as different type of datasets require different type of exploratory techniques. For the numerical datasets, statistical functions like mean, median, standard deviation etc are applied to get an idea of the values within the data attributes. Also, researchers apply different type of visual representations like Bar graphs, Frequency distributions and word clouds to

CHAPTER 3. RESEARCH METHODOLOGY

see what's in the data. The outcome of this technique is to engineer features that amplify the model results.

3.1.4 Feature Engineering

Usually the exploratory data analysis and the feature engineering go hand in hand. Based on the insights collected in previous step, the features are engineered on which the model is trained and the results are generated. This is a cycle as represented in the the diagram as well where researchers have to keep going back and forth between the exploratory data analysis and the feature engineering steps. The ultimate goal of both of these steps is to curate features that generate models with improved results.

3.1.5 Model Evaluation and Tuning

In this step, a machine learning model is trained and evaluated against the desired outcomes or the previously published work. If the results are not satisfying against the evaluation metrics, the model tuning is performed. It is to be noted here that defining the evaluation metrics prior to the model evaluation is important so that the results can be quantified. Model tuning sometimes involves changing the model parameters from default to a few other random values. If the results are not close to the desired outcomes, then the researcher might have to go a step back to feature engineering and tune the selected features more. Model evaluation and tuning is performed repeatedly until the results surpass the previously published work against the defined evaluation metrics or the satisfying level of results are obtained.

3.2 Tools & Technologies Used

Table 3.1 lists the tools and technologies used during this research.

Tool / Technology	Rationale behind usage
Python [22]	Python programming language was used to write scripts for pulling and compiling data from Github Search API
numpy [23]	numpy is a python library that provides capability of performing complex mathematical operations
pandas [24]	All data processing was done in pandas.DataFrame [25] datatype
matplotlib [26]	This library was used for drawing graphical representations like the confusion matrix
scikit-learn [27]	Python library for machine learning implementations
wordcloud [28]	This library was added to generate word clouds from raw text
Github Search API [7]	Github Search API was used to pull publicly available user's bash history files
Cowrie SSH Honeypot [29]	Cowie honeypot was deployed for collection of malicious data
Docker [30]	Docker was used for containerization of honeypot's deployments
jupyter [31]	Jupyter is a project that enables interactive computing and it was used while performing the machine learning section
Google Colab [32]	Google Colab was used to run jupyter notebooks for all machine learning related tasks
Google Cloud Platform [33]	Honeypot was run on GCP for collection of malicious data
pycharm [34]	This IDE from JetBrains was used for writing the python scripts

Table 3.1: Tools technologies used during this research work

3.3 Summary

In this section, we presented a graphical as well as textual representation of the complete research methodology that was followed while conducting this research. We also listed all the tools and technologies that were used during this work along with the rationale behind using each of those tool / technology in Table 3.1.

Chapter 4

Data Collection

This chapter provides the in-depth insight into the the complete data collection and cleaning technique used in this research.

4.1 Benign Data Collection

Bash [35], a Linux shell, stores its commands history in the `.bash_history` file in user's home directory by default. Programmers, sometimes publish their `.bash_history` files on their Github [36] repositories with or without their knowledge. This makes their commands history publicly available and can be pulled by anyone. Since these commands were written with non-malicious intent by actual engineers on their shell, therefore, we can treat these history files as the benign dataset [5]. For the actual collection of those files, we make use of the Github Search API [7]. While we found a total of 10,654 results on Github with our initial query, it is to be noted that Github neither through its web interface, nor the API returns more than 1000 results per query. To

CHAPTER 4. DATA COLLECTION

get past this blocker, we exploited the size parameter of Github Search API to design a set of queries based on determined size ranges which limited the number of records to less than 1000 per query. Using this approach, we were able to collect a total of 9,623 files from Github. After removing empty files and files containing duplicate content, we were left with 5,577 unique records.

4.2 Malicious Data Collection

First, we established our own cloud-based honeypot network for the collection of malicious data. For this purpose, we ran a Docker containerized version of Cowrie [29] honeypot on three Ubuntu OS [37] based virtual machines (VMs) in Google Cloud from 1st of September 2021 to 15th of October. Along with these nodes, we also created a managed MySQL database [38] which was acting the storage backend for the honeypot VMs. All the activity on the honeypot was automatically being pushed to that managed database. Graphical representation of the deployed honeypot architecture is presented in Figure 4.1.

We burned through the 300\$ free credit provided by Google on 15th of October 2021. Due to the lack of resources, all three honeypots were run in low-interaction mode. After analyzing the data collected during that month and a half period, we realized that the low-interaction mode of the honeypot was not generating quality interactions by the attackers. We found that the low interaction honeypots were detectable by the attacking bots rendering into very low interactions before termination of the connection. Quantifying the detectability of those low-interaction honeypots by the attacking bots

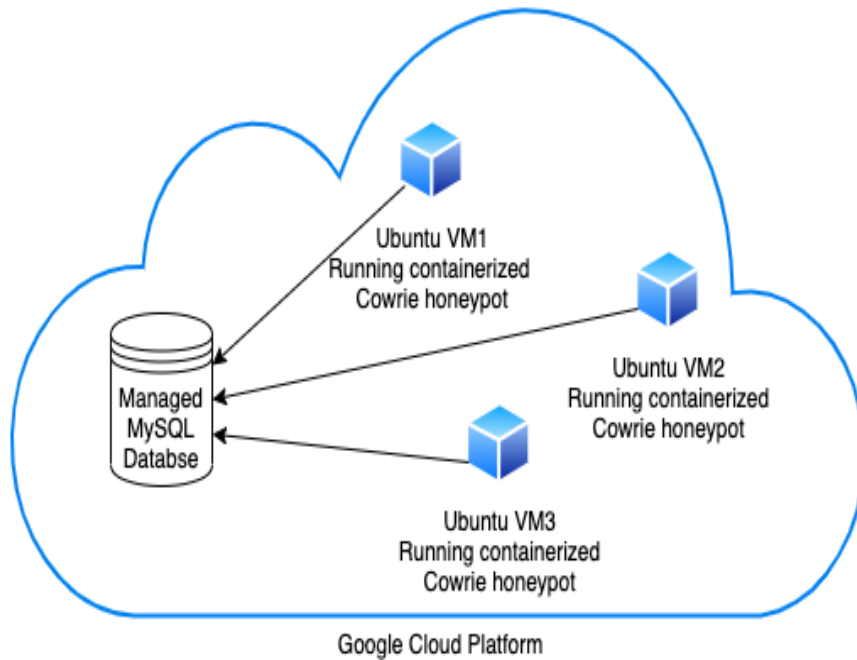


Figure 4.1: Deployed honeypot architecture

is perhaps a topic of research for another day. Upon further looking, we were able to find Cyberlab Honeypot dataset [6] that was collected on a similar model as ours but with a very high number of machines run in low to high interaction modes over the span of 10 months. This dataset was also collected using Cowrie SSH honeypot [17]. The data was collected in 10 months by running Cowrie on more than 50 Linux nodes in United States and European region in different companies and institutions. Initially, the honeypot was run in low-interaction mode running Cowrie version 1.6.0. On 8th of November 2019, the honeypot was re-run in high-interaction mode by backing it up with actual Linux machines made possible by Cowrie version 2.0.2.

4.3 Data Parsing

This data, after expansion comprised of 215.63 GB and each entry in the data contained session related information, timestamp, shell commands written, source and destination identifiers as well as geolocation data based on the source IP address. An example record in JSON format [39] is given below:

```
{
  "session_id": "db517f7638ee",
  "dst_host_identifier": "2b143c5ed9d9b0bc00109e3db0258c672eda7ca7b8da0bf379a1a73c",
  "eventid": "cowrie.command.input",
  "timestamp": "2019-12-01T00:09:45.540815Z",
  "src_ip_identifier": "4617b44fd42fdb4871bce9421561ed2b6c6d15fd8f570af627443c439",
  "dst_ip_identifier": null,
  "message": "CMD: cat /proc/cpuinfo | grep name | wc -l",
  "protocol": null,
  "src_port": null,
  "sensor": "cowrie-deployment-v02-pcrn5",
  "geolocation_data": {
    "postal_code": "48034",
    "continent_code": "NA",
    "country_code3": "US",
    "region_name": "Michigan",
    "ip": "4617b44fd42fdb4871bce9421561ed2b6c6d15fd8f570af627443c43927f727",
    "latitude": 42.4753,
    "country_name": "United States",
```


CHAPTER 4. DATA COLLECTION

```
"longitude":_83.2845,  
"location":_{  
  "lat":_42.4753,  
  "lon":_83.2845  
},  
"timezone":_"America/Detroit",  
"country_code2":_"US",  
"dma_code":_505,  
"region_code":_"MI",  
"city_name":_"Southfield"  
},  
"arch":_null,  
"duration":_null,  
"ssh_client_version":_null,  
"username":_null,  
"password":_null,  
"hasshAlgorithms":_null,  
"macCS":_null,  
"langCS":_null,  
"compCS":_null,  
"encCS":_null,  
"hassh":_null,  
"kexAlgs":_null,  
"keyAlgs":_null,  
"fingerprint":_null,
```

CHAPTER 4. DATA COLLECTION

```
"key":_null,
"type":_null,
"outfile":_null,
"destfile":_null,
"duplicate":_null,
"shasum":_null,
"url":_null,
"ttylog":_null,
"size":_null,
"filename":_null,
"data":_null
}
```

The most relevant attributes for our usecase include `session_id`, `message` and `timestamp`. It is to be noted here that this one data point does not represent the complete session information. We extracted all data points with the correlated `session_id` to get all the the information for a session. Also, one data point only included a single command that was entered on the shell. So a complete session will comprise of multiple data points each containing the same `session_id` but different `message` and `timestamp`. After collecting all the data points for a single session, we sorted them based on the `timestamp` to get the exact order of the commands entered on the shell in a single session.

Then we wrote Python scripts to parse and extract all the SSH sessions from the dataset which contained at least one executed command and

CHAPTER 4. DATA COLLECTION

dropped all sessions with no executed commands. We were able to collect a total of 2,98,667 sessions. Since the same attacks were performed against all the nodes multiple times from multiple sources, it resulted into a lot of sessions containing duplicate sequences of commands. After dropping all the sessions with duplicate sequence of commands, we were left with 2,835 unique sessions.

4.4 Data Summary

Table 4.1 represents the complete dataset details:

Total collected benign data points	9,623
Unique benign data points	5,577
Total collected malicious data points	2,98,667
Unique malicious data points	2,835
Total dataset entries	8,412

Table 4.1: Dataset summary

Chapter 5

Exploratory Data Analysis & Feature Engineering

Previous studies targeting the detection of malicious sessions like [5] and [21], primarily focused on using NLP based features like N-grams or character grams. We decided to curate some simpler features by purely relying on the domain knowledge while achieving similar if not better results. We only had the single dimensional textual data in our dataset which limited our options to explore it using the variety of traditional visualizations. First, we generated the word cloud for the malicious data without removing any command arguments or flags just to get a gist of what is present in the data, displayed in Figure 5.1.

Apart from this, we also generated bar graph for highly used words in the malicious data, word cloud for benign dataset and bar graph for highly used words in benign dataset represented in Figure 5.2, Figure 5.3 and Figure 5.4 respectively.

CHAPTER 5. EXPLORATORY DATA ANALYSIS & FEATURE ENGINEERING

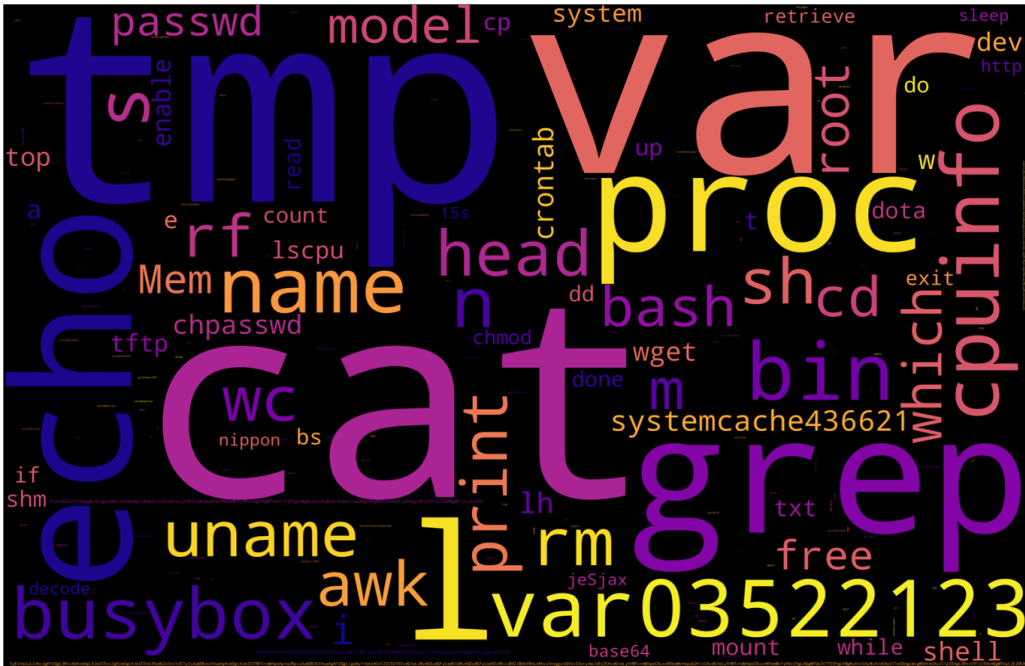


Figure 5.1: Word cloud for malicious data

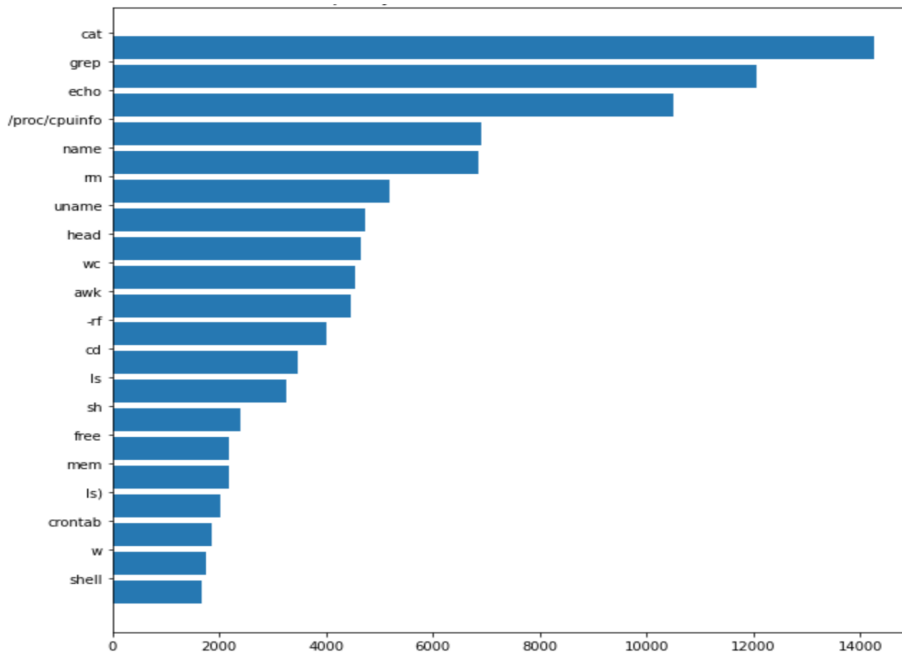


Figure 5.2: Frequency of highly used words in malicious data

CHAPTER 5. EXPLORATORY DATA ANALYSIS & FEATURE ENGINEERING

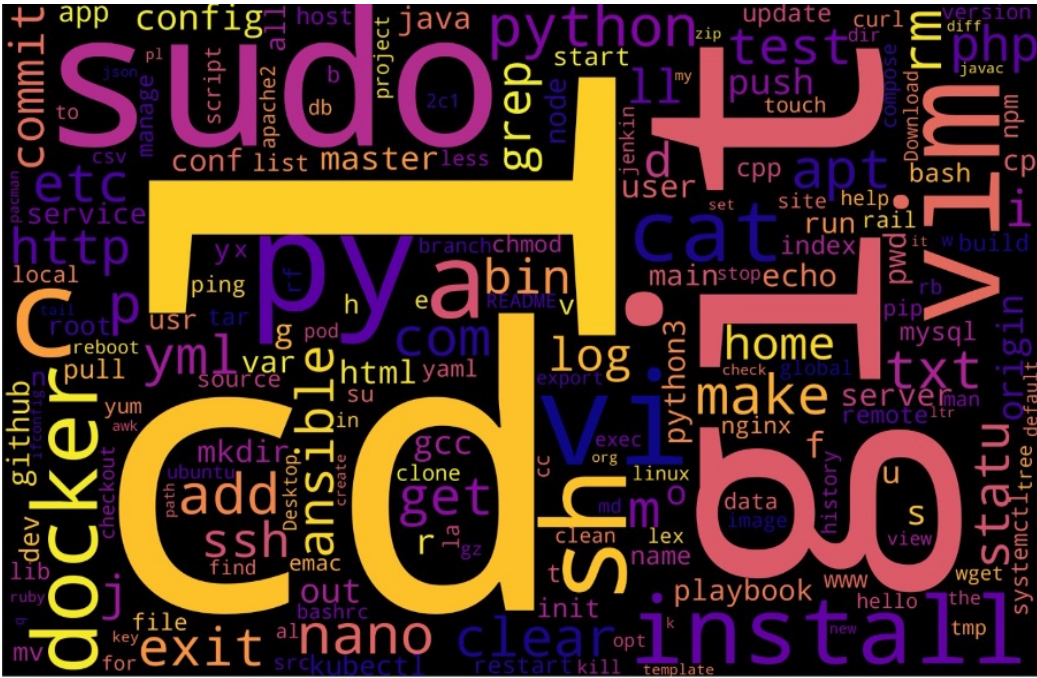


Figure 5.3: Word cloud for benign data

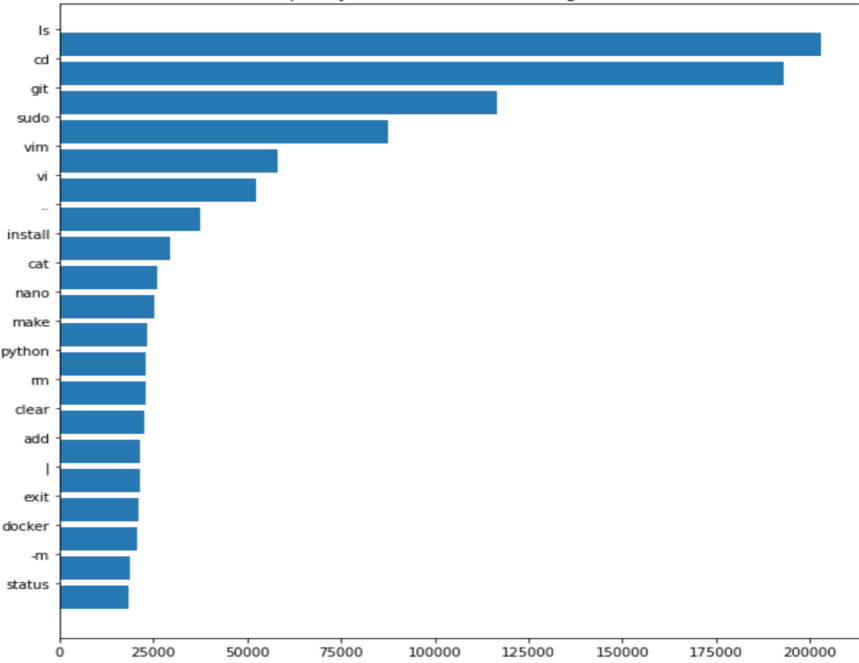


Figure 5.4: Frequency of highly used words in benign data

By looking at the word cloud, we were able to make the following observations:

5.1 System Exploratory Commands

We instantly noticed a trend of words which were either device exploratory commands or could potentially be used as an argument to a command to collect information about the underlying system. These words included `cpuinfo`, `uname`, `which`, `free`, `system`, `mem`, `shell`, `w`, `proc`, `lscpu`, `top`, etc. Our initial observations were backed by the number of occurrences of the device exploratory words in the Figure 5.2. These observations helped us curate our first feature, `system_exploratory_cmds`, a count of all the system exploratory commands, including `system`, `device`, `OS`, `shell`, `disk`, `memory` and `process` exploratory commands. An example session from malicious data containing system exploratory commands is shown in Figure 5.5.

```
CMD: enable
CMD: system
CMD: shell
CMD: sh
CMD: cat /proc/mounts; /bin/busybox YBPIT
CMD: cd /dev/shm; cat .s || cp /bin/echo .s; /bin/busybox YBPIT
CMD: tftp; wget; /bin/busybox YBPIT
CMD: dd bs=52 count=1 if=.s || cat .s || while read i; do echo $i; done < .s
CMD: /bin/busybox YBPIT
CMD: rm .s; exit
```

Figure 5.5: System exploratory commands example from a malicious session

5.2 Deletion Commands

Deletion is also a very common form of sabotage by the attacking bots on their targets. We noticed a very high number of `rm`, and `-rf` which is a deletion command and the recursive deletion flag passed to it respectively. Using this information, we curated our next feature, `deletion_cmds`, a count of all the deletion commands in a single SSH session.

5.3 Echo File Writes

The third highest word occurrence in the malicious data was the `echo`. It is a Linux command used to display variables and other information onto the shell or to a file. We know this by experience that benign users hardly ever use `echo` to write files. Instead, it is mostly used to display the values of an environment variable onto the shell. An example session from the malicious data containing file writes using `echo` command is displayed in 5.6.

```
CMD: cd /var/tmp; echo
"IyEvYmluL2Jhc2gKY2QgL3RtcApybSAAtcmYgLIgxNS_Percentage_Style_MTUtdW5peApjZCAuWD
E1LXVuaXgKcGtpbGwGLTkgY3JvbiA+HC5vdXQkd2dldCAtcSBodHRwOi8vNTQuMzcuNzAuMjQ5L2Rv
dGEyLnRhci5neiB8fCBjdXJlC1PIC1mIGhOdHA6Ly81NC4zNy43MC4yNDkvZG90YTludGFyLmd6Cn
NsZWVwIldzIjYmIHRhciB4ZiBkb3RhMi50YXluZ3oKI3JtIC1yZiBkb3RhMi50YXluZ3oKY2QgLnJzeW
5jCmNobW9kIDc3NyAqCmNkIC90bXAvLlgxNS11bml4Ly5yc3luYy9hICYmIC4vY3JvbiB8fCAuL2FuY
WNyb24KZXhpdCAw">.threatstackcloudsecops; base64 --decode .threatstackcloudsecops | bash
CMD: cat /proc/cpuinfo | grep name | wc -l
CMD: echo "root:bVD7dxMsNfY"|chpasswd | bash
CMD: echo "321" > /var/tmp/.var03522123
CMD: rm -rf /var/tmp/.var03522123
CMD: cat /var/tmp/.var03522123 | head -n 1
CMD: cat /proc/cpuinfo | grep name | head -n 1 | awk '{print $4,$5,$6,$7,$8,$9;}'
CMD: free -m | grep Mem | awk '{print $2, $3, $4, $5, $6, $7}'
CMD: ls -lh $(which ls)
CMD: which ls
CMD: crontab -l
CMD: w
CMD: uname -m
CMD: cat /proc/cpuinfo | grep model | grep name | wc -l
CMD: top
CMD: uname
CMD: uname -a
CMD: lscpu | grep Model
```

Figure 5.6: Echo file write example from a malicious session

5.4 Difference in IP and URL Based Downloads

The most common commands on the Linux to download stuff from the internet are `wget` and `curl` and we saw a very wide usage of these commands in the malicious as well as the benign. But the distinctive difference we observed from multiple records was that the source of the downloads in malicious data was an IP address instead of a domain name almost all of the time. This makes sense as attackers would want to hide their identity thus not using a domain name to host their malicious scripts as doing so could reveal their identity. On the other hand, for benign data the downloads were from domain names almost all the time. Using this information, we calculated two new numerical features called `ip_downloads` and `url_downloads`. These were simply the number of IP and URL downloads in a single SSH session respectively. An example of IP based downloads is given in Figure 5.7.

```
CMD: cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget
http://37.49.230.137/bins.sh; chmod 777 bins.sh; sh bins.sh;
CMD: enable
```

Figure 5.7: IP based downloads example from a malicious session

5.5 Directory Navigation Commands

The change directory command `cd`, has the second highest number of occurrences in benign data (Figure 5.3 Figure 5.4). Using this information, we calculated a new feature called `dir_nav_cmds`, a total count of directory navigational commands in a single SSH session.

5.6 Access Manipulation Commands

Another technique attackers use is to automate their bots to change the access credentials of the target in case of a successful attack. This is done to keep access for further remote sessions while blocking the access of the legitimate user permanently. Later, detailed exploration of the compromised hosts is done by the attackers. This is usually done using the `passwd` and `chpasswd` commands in the malicious SSH sessions. Since these access manipulation commands were found in a very high number in the malicious data points, we calculated another feature called `access_manipulation_cmds`, a count of all these access manipulation attempts in a single SSH session. An example of IP based downloads is given in Figure 5.8.

```
CMD: cat /proc/cpuinfo | grep name | wc -l
CMD: echo "root:tnfqSlohIFBV"|chpasswd|bash
CMD: echo "321">/var/tmp/.var03522123
CMD: rm -rf /var/tmp/.var03522123
CMD: cat /var/tmp/.var03522123 | head -n 1
CMD: cat /proc/cpuinfo | grep name | head -n 1 | awk
'{print $4,$5,$6,$7,$8,$9;}'
CMD: free -m | grep Mem | awk '{print $2 , $3, $4, $5, $6,
```

Figure 5.8: Access manipulation commands example from a malicious session

CHAPTER 5. EXPLORATORY DATA ANALYSIS & FEATURE ENGINEERING

Apart for these observations, some manual observations were done by looking at the data. The first thing that was noticeable was that the benign data used one command per line, while in the malicious data, multiple commands were put into single line separated by either a comma, a logical AND (&&) or the logical OR || sign. This difference was so distinctive in both of the data classes that we calculated two new features out of these namely `total_lines` and `total_cmds`. Table 5.1 summarizes the initial set of features we calculated for the dataset.

5.7 Feature Selection

Feature selection was performed with intuition as well as the generating the correlations between the features and removing one of the strongly correlated features. While the baseline features as listed in Table 5.1 produced very promising results in our initial tests, we believe transforming some of these raw counts into percentage-based features could produce much better results. For example, percentage of deletion and system exploratory commands in total commands could better reflect their weight in the model as compared to raw counts. To elaborate on this, assume a benign SSH session of 100 commands with 2 `rm` commands with respect to a malicious SSH session of only 10 commands with 2 `rm` commands. Both of these sessions have the same number of deletion commands but if we look at the percentage, it varies from 2% in the benign session to 20% in the malicious session. Similarly, having 5 system exploratory commands in a benign session of 50 commands versus having 5 of those commands in a session of only 10 commands hugely vary in

CHAPTER 5. EXPLORATORY DATA ANALYSIS & FEATURE ENGINEERING

Feature	Description	Example(s)
<i>system_exploratory_cmds</i>	A total count of System, OS, Device, Memory / Disk, Process and Shell exploratory commands	<i>sh, shell, /bin/sh, enable, uname, dmidecode, lshw, lscpu, lspci, df, which, top, atop, htop, free, fdisk, disk, ps</i>
<i>deletion_cmds</i>	A count of remove commands	<i>rm, rm -rf, rm- rf ./</i>
<i>echo_file_writes</i>	A count of file writes using echo command	<i>echo "rm -rf ." >hack.sh</i>
<i>ip_downloads</i>	A count of downloads from an IP address using wget or curl command	<i>wget https://55.5.5.5/dir/hack.sh</i>
<i>url_downloads</i>	A count of downloads from a URL using wget or curl command	<i>wget http://dummyurl.com/file.sh</i>
<i>dir_nav_cmds</i>	A count of directory navigational commands	<i>cd cd /this cd ~/.ssh/</i>
<i>access_manipulation_cmds</i>	A count of access manipulation commands	<i>passwd chpasswd</i>
<i>total_lines</i>	Total number of lines the attack commands are spanning	-
<i>total_cmds</i>	A total count of commands in all lines	-

Table 5.1: Extracted features based on domain knowledge

CHAPTER 5. EXPLORATORY DATA ANALYSIS & FEATURE ENGINEERING

percentage despite having the same number of commands. Building on this theory, we transformed `system_exploratory_cmds` and `deletion_cmds` to `system_exploratory_cmds_percentage` and `deletion_cmds_percentage` by dividing these values by `total_cmds`.

$$\text{system_exploratory_cmds_percentage} = \frac{\text{system_exploratory_cmds}}{\text{total_cmds}}$$

$$\text{deletion_cmds_percentage} = \frac{\text{deletion_cmds}}{\text{total_cmds}}$$

Our initial tests also proved that `System_exploratory_cmds_percentage` and `deletion_cmds_percentage` outperformed the non-percentage distributions of these variables. After this we generated the correlation matrix for the remaining features given in Figure 5.9.

We can clearly see that there is a very strong correlation between the `total_cmds` and the `total_cmd_lines` features. To remove this correlation, we converted them into a single ratio based feature called `lines_to_cmds_ratio`.

$$\text{lines_to_cmds_ratio} = \frac{\text{total_lines}}{\text{total_cmds}}$$

Table 5.2 lists all of the selected features.

CHAPTER 5. EXPLORATORY DATA ANALYSIS & FEATURE ENGINEERING

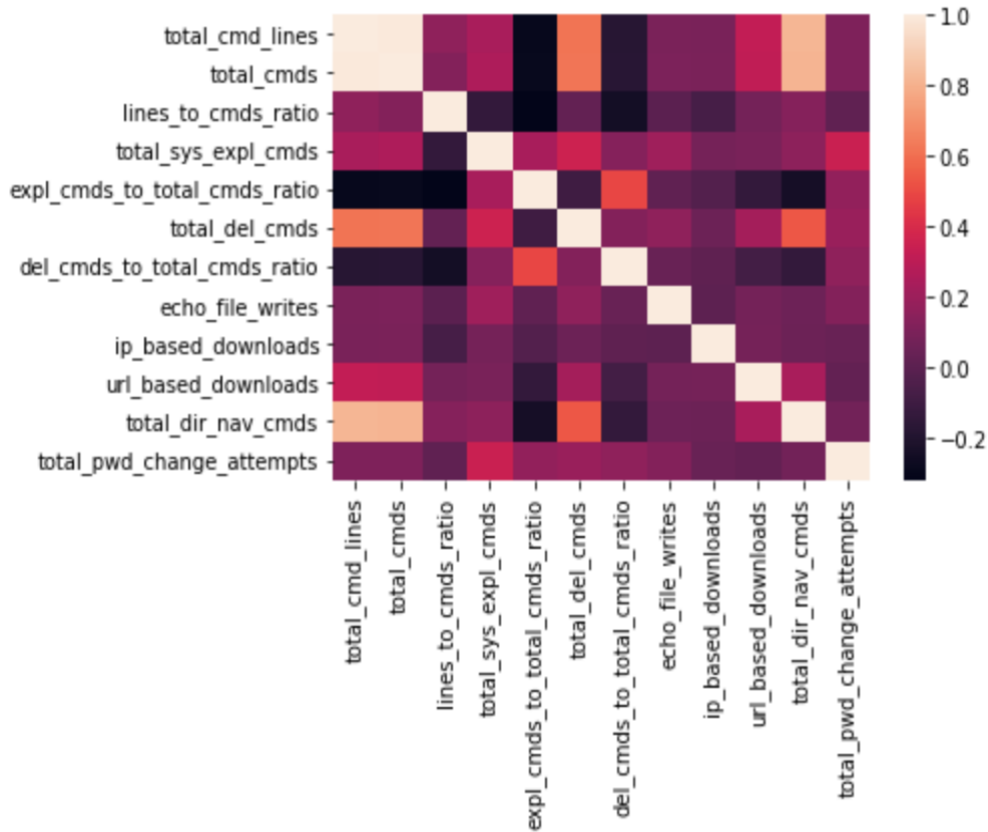


Figure 5.9: Correlation matrix for features

<i>Feature</i>	<i>Description</i>
<i>sys_exploratory_cmds_percentage</i>	<i>system_exploratory_cmds divided by total_cmds</i>
<i>deletion_cmds_percentage</i>	<i>deletion_cmds divided by total_cmds</i>
<i>echo_file_writes</i>	<i>A count of file writes using echo command</i>
<i>ip_downloads</i>	<i>A count of downloads from an IP</i>
<i>url_downloads</i>	<i>A count of downloads from a URL</i>
<i>dir_nav_cmds</i>	<i>A count of directory navigational commands</i>
<i>access_manipulation_cmds</i>	<i>A count of access manipulation commands</i>
<i>lines_to_cmds_ratio</i>	<i>total_lines divided by total_cmds</i>

Table 5.2: Finalized features after extensive EDA processing

5.8 Summary

In this section, first we conducted the exploratory data analysis of the data. Word cloud and bar graphs were generated for malicious and benign data to get a clear picture of data contents. Based on the observations, initial features were designed. Then, a theory was developed regarding percentage based feature over-performing normal number based features. Based on this theory, feature transformation was done to convert some of the normal features into percentage based features. Then, feature correlation was calculated and the correlated features were dropped. In the upcoming section, we will start the evaluation on our finalized set of features.

Chapter 6

Model Training & Evaluation

In this chapter, we first define our evaluation metrics. After that we discuss of model training methodology and the results obtained by our models against the defined evaluation metrics. Finally, we provide the comparative analysis of our results with the published work in the domain.

6.1 Evaluation Metrics

Having precise evaluation metrics helps gauge the results better. Therefore, before jumping to the results, lets define the evaluation metrics for our results. Given TP, TN, FP and FN as True Positive, True Negative, False Positive and False Negative respectively, we define our evaluation metrics as:

6.1.1 Accuracy

The number of accurately anticipated data points out of all the data points is known as accuracy.

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

6.1.2 True Positive Rate (TPR)

True positive rate, also known as sensitivity is the chance that a true positive will test positive.

$$TPR = \frac{TP}{TP + FN}$$

6.1.3 True Negative Rate (TNR)

True negative rate, also known as specificity is the chance that a true negative will test negative.

$$TNR = \frac{TN}{TN + FP}$$

6.1.4 False Positive Rate (FPR)

The false-positive rate is the relative frequency of not positive when positive is predicted.

$$FPR = 1 - TNR$$

6.1.5 False Negative Rate (FNR)

The false-negative rate is the relative frequency of not negative when negative is predicted.

$$FNR = 1 - TNR$$

6.2 Model Selection

We experimented on the following frequently used binary classification models:

- Support Vector Machines
- Decision Trees
- Random Forests
- Naïve Bayes

6.3 Initial Results

We did an 80/20 split of our data and trained the Support Vector Machines (SVM), Random Forest (RF), Decision Tree (DT) and Naïve Bayes (NB) classifiers with the 80% of the training data. The testing was done using the remaining 20% of the data. Among these classifiers, Random Forest showed the best results with an accuracy of 99.70%, TPR and TNR of 99.72% and 99.67% respectively.

With our initial tests in python's machine learning library scikit-learn, while using all the default parameters for each classifier, Random Forest

(RF) produced the most promising results. Initial results are documented in Table 6.1.

<i>Classifier</i>	<i>Accuracy</i>	<i>TPR</i>	<i>TNR</i>	<i>FPR</i>	<i>FNR</i>
<i>Random Forest</i>	0.9970	0.9972	0.9967	0.0027	0.0032
<i>Decision Tree</i>	0.9952	0.9962	0.9934	0.0037	0.0065
<i>Naive Bayes</i>	0.9322	0.8984	0.9918	0.1015	0.0081
<i>SVM</i>	0.9019	0.9757	0.7721	0.0242	0.2278

Table 6.1: Initial results of different classifiers

6.4 K-Fold Cross Validation with Hyper-Parameters Tuning

Even though the results produced by RF are satisfactory, we see two problems here. First, testing on a single distribution of data does not represent how a model will actually perform when deployed in a production environment. Doing so could result into model overfitting where the models will report great results when tested on the initial distribution but will perform poorly when tested on unseen data. Second problem is that using the model's default hyperparameters could potentially rob us from much better results. Hyperparameters are the parameters specific to each classifier which control the learning rate while training and are set before the model is trained. The default values might not be the most optimal therefore there are couple of techniques which could provide insight towards improvement. Hyperparameters can be tuned using the brute force method, random search method and grid search method. In brute force method, you try each and every possible

CHAPTER 6. MODEL TRAINING & EVALUATION

value for every hyperparameter on a data distribution and select the set of parameter values that produce the best results. In random search method, a large set of grid hyperparameter values is provided and model is trained on random set of values from the grid. Grid search selects a grid of hyperparameter values and compares them all for the best performing set of grid.

It is to be noted here that hyperparameter tuning alone does not guarantee better results for unseen data as the tuning is specific to a single distribution of data points. Therefore, we consolidate hyperparameter tuning using GridSearch along with K-Fold Cross-Validation technique to get the best performing hyperparameters on multiple data point distributions. K-Fold Cross-Validation is a technique where the training data is split into K folds with 1 fold as the validation set and the k-1 folds for training. This process is repeated K times until every fold has played a role of the validation set. An example of 3-Fold Cross Validation is depicted in Figure 6.1.

Since the results are averaged in K-Fold Cross Validation, we dropped the low performing classifiers based on the fact that their initial low prediction results will bring their overall averages low. We performed hyperparameters tuning on RF, our best performing classifier with the 10-Fold Cross Validation in hope of optimizing the `max_depth` and the `n_estimators` parameters. The 10-Fold Cross Validation is performed on every possible combination of the input hyperparameters. The input was given in the form of a python dictionary and the `GridSearchCV` method of `scikit-learn` was used with `cv` value of 10.

CHAPTER 6. MODEL TRAINING & EVALUATION

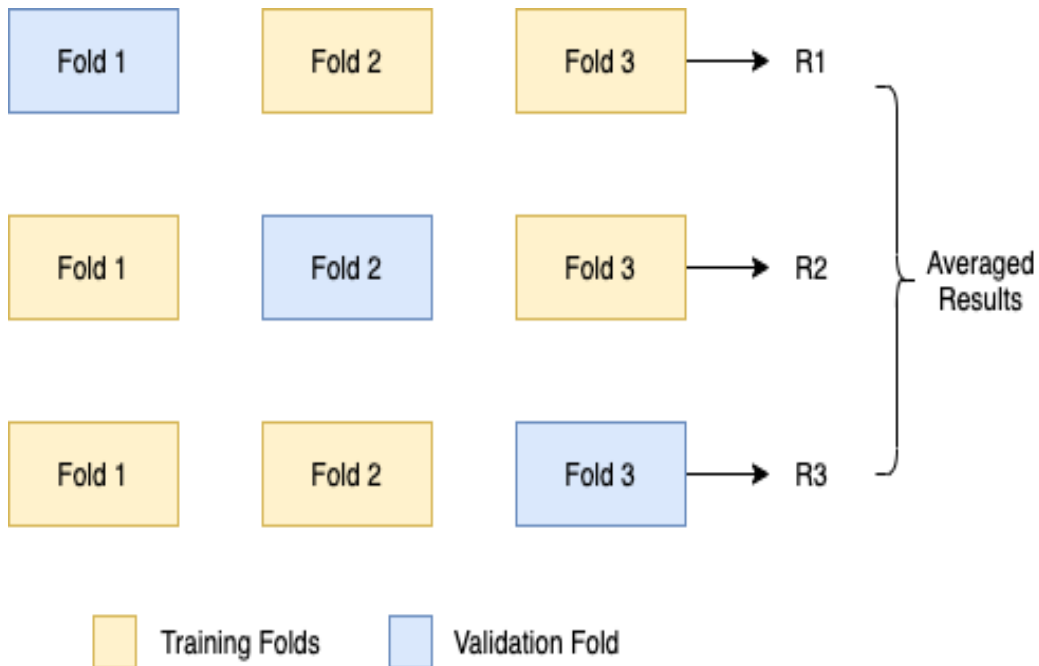


Figure 6.1: K-Fold Cross Validation for $K = 3$

```
param_grid={  
'max_depth': [None, 50, 100, 150],  
'n_estimators': [100, 200, 300, 400],  
}
```

A model was fit a total of 160 times ($4 * 4 * 10$ folds = 160). Based on the results, the best performing values for `max_depth` and `n_estimators` were `None` and `400` respectively. Training an RF classifier with these hyperparameters and testing on the test set further increased the accuracy to 0.9976 along with other attributes. Also, test dataset can easily be called as purely unseen data so we can say with confidence the final trained model will produce in similar results when deployed in production. Results of RF classifier before the model tuning as well as the results after tuning are documented in Table

CHAPTER 6. MODEL TRAINING & EVALUATION

	<i>Before Tuning</i>	<i>After Tuning</i>	<i>Difference</i>
<i>Accuracy</i>	0.9970	0.9976	+ 0.0006
<i>TPR</i>	0.9972	0.9981	+ 0.0009
<i>TNR</i>	0.9967	0.9967	Same
<i>FPR</i>	0.0027	0.0018	- 0.0009
<i>FNR</i>	0.0032	0.0032	Same

Table 6.2: RF results before and after model tuning

6.2.

Results of RF with the testing data after the hyperparameters tuning are displayed in Figure 6.2.

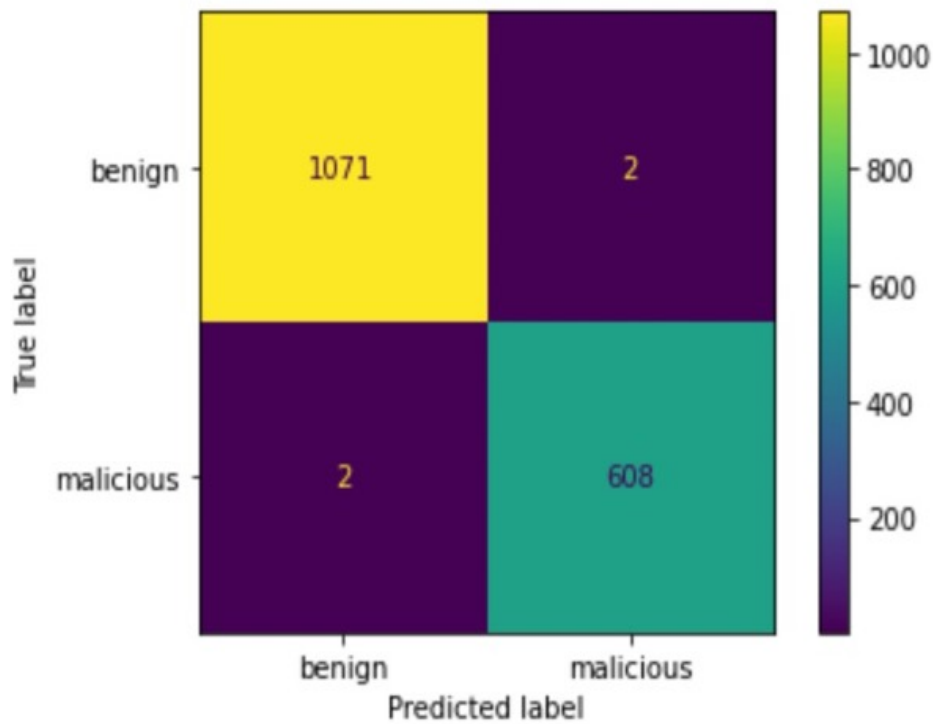


Figure 6.2: Confusion matrix for RF after tuning

6.5 Discussion

Due to the lack of study in this exact niche field, we were only able to compare our results with [5]. We extracted the features from the commands entered on the shell purely based on the domain knowledge and it produced much better accuracy and true positive rate. The previous study depends on the continuity of malicious commands in a sliding window in order to detect the malicious sessions in a robust way, our approach is independent of such assumptions. Also, we use random forest classifier which does not need to store the training data in memory unlike KNN and thus it results into a very minimalistic system which sits on the nodes without much interference. The results were further improved by applying hyperparameters tuning along with the 10-fold cross validation to add more generalization aspect to the model.

Chapter 7

Conclusion & Future Work

7.1 Concluision

Securing the cloud infrastructure is the key to securing workloads running in cloud and it acts as the baseline security control that needs to be implemented before an organization starts their cloud migration journey. While SSH is the highly used protocol to access Linux based cloud infrastructure, the traditional controls do not provide adequate amount of security against a malicious insider or a compromised set of credentials in hands of an attacker. This research has verified that employing machine learning to classify each SSH session into Malicious or Benign using only the commands entered in the shell produces promising results against the presented attack model. It has also been verified that the light-weight features extracted purely based on the domain knowledge perform better than other natural language processing (NLP) based features like word-grams and character-grams as proposed in previous works. We extracted the malicicious SSH sessions from a published

CHAPTER 7. CONCLUSION & FUTURE WORK

dataset and wrote scripts for collection of benign dataset. Both of these individual datasets were then compiled to form a single dataset. A comprehensive data exploration was performed and feature extraction was done based on the findings. Those initial features were segmented through a series of transformations for improvement and also the correlated features were dropped. Last, but not the least, we conducted our tests and further improved the results by performing K-Fold Cross Validation and Hyper-Parameters tuning. Finally, we compared our results with the previously published work, There is very little work done on the field previously and I believe it is important to address this issue in multiple ways to help contribute towards the safe adoption of cloud in the coming era.

7.2 Future Work

We believe that this research could easily be expanded into the following potential directions:

7.2.1 An Improved Approach for Benign Data Collection

While our approach already produces near optimal results, we believe these results can still be improved. Our benign dataset was collected from the publicly accessible bash history files available on Github repositories which is a closer representation of commands used in a staging environment. Usually, the commands entered in the staging or testing environments tend to include

CHAPTER 7. CONCLUSION & FUTURE WORK

the large variety of commands which are also common in actual malicious datasets. To elaborate on this, there is a very good probability of commands like `rm`, `passwd`, `uname` etc. being run on a staging environment than on a production environment. The actual production systems might never see a manual `rm` or a `passwd` command due to more restricted access and lack of change to the underlying system once it is hosting production workloads. Typically, a limited and same set of commands are run on production nodes repeatedly. Therefore, collecting a benign commands dataset which is purely from the SSH sessions logged on production nodes will produce even better results with lesser false positives and false negatives.

7.2.2 Generalizing for Other Linux and IOT Shells

This work was done on data collected using the Bash shell of Linux. There are variety of shells available with each having a unique set of executable commands. We believe this work can be further generalized for all other available shells of Linux and Linux based shells for Internet-of-Things (IoTs).

7.2.3 Training Models for Specialized Services / Dedicated Nodes

All production services can be categorized into a small set of categories. These categories include containerized services, database services, proxy services, etc. Also, the commands that are run on a node are dictated by the type of service running on the node. Therefore, training specialized models for each type of services and using those dedicated models on nodes which

CHAPTER 7. CONCLUSION & FUTURE WORK

are running same category of services could further improve the results.

7.3 Summary

We conclude our research in this section and provide a few potential research directions that can be explored to further enhance this work.

Bibliography

- [1] “Realising the Value of Cloud Computing with Linux.”
<https://www.rackspace.com/en-gb/blog/realising-the-value-of-cloud-computing-with-linux> (accessed Oct. 25, 2021).
- [2] R. W. Scheifler and J. Gettys, “The Secure Shell (SSH) Connection Protocol,” p. 1000, 1992, Accessed: Nov. 17, 2021. [Online]. Available: <https://tools.ietf.org/html/rfc4251>
- [3] R. Andrews, D. A. Hahn and A. G. Bardas, ”Measuring the Prevalence of the Password Authentication Vulnerability in SSH,” ICC 2020 - 2020 IEEE International Conference on Communications (ICC), 2020, pp. 1-7, doi: 10.1109/ICC40277.2020.9148912.
- [4] “Insider Threat Statistics You Should Know: Updated 2021 - Tessian.” <https://www.tessian.com/blog/insider-threat-statistics/> (accessed Nov. 05, 2021).
- [5] P. Dumont, D. Gugelmann, R. Meier, and V. Lenders, “Detection of Malicious Remote Shell Sessions.”

BIBLIOGRAPHY

- [6] U. Sedlar, M. Kren, L. Štefanič Južnič, and M. Volk, “CyberLab honeynet dataset,” Feb. 2020, doi: 10.5281/ZENODO.3687527.
- [7] “GitHub Search API.” <https://docs.github.com/en/rest/reference/search> (accessed Oct. 20, 2021).
- [8] D. Bove and T. Muller, “Investigating Characteristics of Attacks on Public Cloud Systems,” in Proceedings - 6th IEEE International Conference on Cyber Security and Cloud Computing, CSCloud 2019 and 5th IEEE International Conference on Edge Computing and Scalable Cloud, EdgeCom 2019, Jun. 2019, pp. 89–94. doi: 10.1109/CSCloud/EdgeCom.2019.00-13.
- [9] 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,” 2020 5th International Conference on Computer and Communication Systems (ICCCS), 2020, pp. 491-497, doi: 10.1109/ICCCS49078.2020.9118459.
- [10] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani, “Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization,” 4th International Conference on Information Systems Security and Privacy (ICISSP), Portugal, January 2018.
- [11] G. K. Sadasivam, C. Hota, and A. Bhojan, “Detection of stealthy single-source SSH password guessing attacks,” Evolving Systems, 2021, doi: 10.1007/s12530-020-09360-3.
- [12] M. M. Raikar and S. M. Meena, ”SSH brute force attack mitigation in Internet of Things (IoT) network : An edge device security

BIBLIOGRAPHY

- measure,” 2021 2nd International Conference on Secure Cyber Computing and Communications (ICSCCC), 2021, pp. 72-77, doi: 10.1109/ICSCCC51823.2021.9478131.
- [13] Vincent Ghiette, Harm Griffioen, and Christian Doerr. 2019. Fingerprinting Tooling used for SSH Compromisation Attempts. In International Symposium on Research in Attacks, Intrusions and Defenses (RAID).
- [14] Song, D.X., Wagner, D., Tian, X.: Timing analysis of keystrokes and timing attacks on SSH. In: USENIX Security Symposium, vol. (2001)
- [15] H. H. Alsaadi, M. Aldwairi, M. Al Taei, M. AlBuainain and M. AlKubaisi, ”Penetration and Security of OpenSSH Remote Secure Shell Service on Raspberry Pi 2,” 2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS), 2018, pp. 1-5, doi: 10.1109/NTMS.2018.8328710.
- [16] E. Crockett, C. Paquin, D. Stebila, Prototyping post-quantum and hybrid key exchange and authentication in TLS and SSH, in NIST 2nd Post-Quantum Cryptography Standardization Conference 2019 (2019)
- [17] Dimitrios Sikeridis, Panos Kampanakis, and Michael Devetsikiotis. 2020. Assessing the Overhead of Post-Quantum Cryptography in TLS 1.3 and SSH. In Proceedings of the 16th International Conference on Emerging Networking EXperiments and Technologies (Barcelona, Spain) (CoNEXT ’20). Association for Computing Machinery, New York, NY, USA, 149–156. <https://doi.org/10.1145/3386367.3431305>

BIBLIOGRAPHY

- [18] D. Ramsbrock, R. Berthier, and M. Cukier, “Profiling Attacker Behavior Following SSH Compromises,” 2007.
- [19] J. Hance and J. Straub, “Use of Bash History Novelty Detection for Identification of Similar Source Attack Generation,” in 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), 2020, pp. 759–766. doi: 10.1109/TrustCom50675.2020.00104.
- [20] J. T. Martínez Garre, M. Gil Pérez, and A. Ruiz-Martínez, “A novel Machine Learning-based approach for the detection of SSH botnet infection,” *Future Generation Computer Systems*, vol. 115, pp. 387–396, Feb. 2021, doi: 10.1016/j.future.2020.09.004.
- [21] H. Alasmary et al., “SHELLCORE: Automating Malicious IoT Software Detection Using Shell Commands Representation,” *IEEE Internet of Things Journal*, 2021, doi: 10.1109/JIOT.2021.3086398.
- [22] “Welcome to Python.org.” <https://www.python.org/> (accessed Nov. 17, 2021).
- [23] “NumPy.” <https://numpy.org/> (accessed Nov. 17, 2021).
- [24] “pandas - Python Data Analysis Library.” <https://pandas.pydata.org/> (accessed Nov. 17, 2021).
- [25] “pandas.DataFrame — pandas 1.3.4 documentation.” <https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html> (accessed Nov. 17, 2021).

BIBLIOGRAPHY

- [26] “Matplotlib: Python plotting — Matplotlib 3.4.3 documentation.”
<https://matplotlib.org/> (accessed Nov. 17, 2021).
- [27] “scikit-learn: machine learning in Python — scikit-learn 1.0.1 documentation.” <https://scikit-learn.org/stable/> (accessed Nov. 17, 2021).
- [28] “wordcloud · PyPI.” <https://pypi.org/project/wordcloud/> (accessed Nov. 17, 2021).
- [29] “Cowrie SSH/Telnet Honeybot: <https://cowrie.readthedocs.io>.”
<https://github.com/cowrie/cowrie> (accessed Oct. 20, 2021).
- [30] “Empowering App Development for Developers — Docker.”
<https://www.docker.com/> (accessed Nov. 17, 2021).
- [31] “Project Jupyter — Home.” <https://jupyter.org/> (accessed Nov. 17, 2021).
- [32] “Welcome to Colaboratory - Colaboratory.”
https://colab.research.google.com/?utm_source=scs-index (accessed Nov. 17, 2021).
- [33] “Cloud Computing Services — Google Cloud.” <https://cloud.google.com/>
(accessed Nov. 17, 2021).
- [34] “PyCharm: the Python IDE for Professional Developers by JetBrains.”
<https://www.jetbrains.com/pycharm/> (accessed Nov. 17, 2021).
- [35] “Bash - GNU Project - Free Software Foundation.”
<https://www.gnu.org/software/bash/> (accessed Oct. 25, 2021).

BIBLIOGRAPHY

- [36] “GitHub.” <https://github.com/> (accessed Oct. 20, 2021).
- [37] “Enterprise Open Source and Linux — Ubuntu.” <https://ubuntu.com/>
(accessed Nov. 17, 2021).
- [38] “MySQL.” <https://www.mysql.com/> (accessed Nov. 17, 2021).
- [39] “JSON Format - rfc7159.” <https://datatracker.ietf.org/doc/html/rfc7159>
(accessed Nov. 17, 2021).