2022-06

Detection and prevention of username enumeration attack on SSH protocol: machine learning approach

Agghey, Abel

NM-AIST

https://doi.org/10.58694/20.500.12479/1628

DETECTION AND PREVENTION OF USERNAME ENUMERATION ATTACK ON SSH PROTOCOL: MACHINE LEARNING APPROACH

Abel Zauru Agghey

A Dissertation Submitted in Partial Fulfillment of the Requirement for the Degree of Master's in Information System and Network Security of the Nelson Mandela African Institution of Science and Technology

Arusha, Tanzania

June, 2022

ABSTRACT

Over the last two decades (2000–2020), the Internet has rapidly evolved, resulting in symmetrical and asymmetrical Internet consumption patterns and billions of users worldwide. With the immense rise of the Internet, attacks and malicious behaviors pose a huge threat to our computing environment. Brute-force attack is among the most prominent and commonly used attacks, achieved out using password-attack tools, a wordlist dictionary, and a usernames list – obtained through a so – called an enumeration attack. In this study, we investigate username enumeration attack detection on SSH protocol by using machine-learning classifiers. We apply four asymmetrical classifiers on our generated dataset collected from a closed-environment network to build machine-learning-based models for attack detection. The use of several machine-learners offers a wider investigation spectrum of the classifiers' ability in attack detection. Additionally, we investigate how beneficial it is to include or exclude network ports information as features-set in the process of learning. We evaluated and compared the performances of machine-learning models for both cases. The models used are k-nearest neighbor (KNN), naïve Bayes (NB), random forest (RF) and decision tree (DT) with and without ports information. Our results show that machine-learning approaches to detect SSH username enumeration attacks were quite successful, with KNN having an accuracy of 99.93%, NB 95.70%, RF 99.92%, and DT 99.88%. Furthermore, the results improved when using ports information. The best selected model was then deployed into intrusion detection and prevention system (IDS/IPS) to automatically detect and prevent username enumeration attack. Study also recommends the use of Deep Learning in future studies.

DECLARATION

I, Abel Zauru Agghey, do declare hereby to the Senate of the Nelson Mandela African Institution of Science and Technology that, this project report is my own original work and that it has neither been submitted nor is it being concurrently submitted for a degree award in any other institution.

Abel Zauru Agghey

11/07/2022

Name and Signature of Candidate Date

The above declaration is confirmed by:

Dr. Jema David Ndibwile

11/07/2022

Name and Signature of Supervisor 1 Date

Dr. Mussa Ally Dida

11/07/2022

Name and Signature of Supervisor 2 Date

COPYRIGHT

CERTIFICATION

The undersigned certify that they have read and hereby recommend for acceptance by the Senate of the Nelson Mandela African Institution of Science and Technology the dissertation entitled: "*Detection and Prevention of Username Enumeration Attack on SSH Protocol: Machine Learning Approach"*, in Partial Fulfilment of the Requirements for the Degree of Master's in Information and Communication Science and Engineering of the Nelson Mandela African Institution of Science and Technology.

Dr. Jema David Ndibwile

11/07/2022

Name and Signature of Supervisor 1 Date

Dr. Mussa Ally Dida

11/07/2022

Name and Signature of Supervisor 2 Date

ACKNOWLEDGEMENTS

Completing this study successfully was made possible by several parties through their dedicated efforts, support and guidance.

First and foremost, I thank God for granting me knowledge, ability, and opportunity of not only undertaking this research study but also persevering and completing it satisfactorily.

I also gratefully acknowledge my main supervisor: Dr. Jema David Ndibwile and co-supervisor Dr. Mussa Ally Dida, and my mentor Mr. Sanket Mohan Pandhare who not only introduced me to the topic but also provided their support throughout this study. Their engagement, remarks, and comments have been invaluable.

My deep appreciation goes out to all the lecturers for their help and support. I also take this opportunity to thank my fellow classmates for the classwork we did together and for their valuable inputs in this dissertation.

Lastly, I appreciate all the help and guidance I received from my parents, siblings, and friends. Thank you very much.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF APPENDICES

LIST OF ABBREVIATION AND SYSMBOLS

COVID-19	Corona Virus Disease of 2019
CVE	Common Vulnerabilities and Exposures
DNS	Domain Name System
DT	Decision Tree
FN	False Negative
FTP	File Transfer Protocol
GHz	Gigahertz
HPC	High Performance Computers
HTTP	Hyper Text Transfer Protocol
IDS	Intrusion Detection System
IoT	Internet of Things
IPS	Intrusion Prevention System
IPv4	Internet Protocol Version Four
KNN	K-Nearest Neighbor
LSTM	Long Short-Term Memory
ML	Machine Learning
OpenSSH	Open Secure Socket Shell
PCAP	Packet Capture
RAM	Random Access Memory
RF	Random Forest
ROC	Receiver Operating Characteristic
SSH	Secure Socket Shell
TCP	Transmission Control Protocol
TN	True Negative
TP	True Positive
VPN	Virtual Private Network

CHAPTER ONE

INTRODUCTION

1.1 Background of the Problem

The Internet is widely recognized for its rapid growth and tremendously usage in current years (Alshehri & Meziane, 2017). Over four billion individuals have Internet access and utilize it on a regular basis. The figure equates to 63.2% of the global population having access to the Internet. According to statistics, Internet usage surged by 1266% during the course of two decades (2000–2020) (Infante-Moro *et al.*, 2016; internetworldstats., 2021). The explosiveness and widespread nature of the Internet have made almost everyone to rely on computer networks for their day-to-day activities (Hoque *et al.*, 2014). With an immense rise in dependency on the Internet and computer networks services, cyberattacks and malicious behaviors have become unexceptional in our computing environment (Jang-Jaccard & Nepal, 2014; Najafabadi *et al.*, 2014).

Cyberattacks cost millions of dollars each year, and the number of victims globally is significantly growing. At least 14 cyberattack victims emerge every second, which equals more than one million attacks every day (Jang-Jaccard & Nepal, 2014). For more than two decades, there has been a growth in the number of instances reported, as well as the complexity of the attacks. As a result, any person connected to the internet, whether internally or externally, is constantly at risk of malicious activity and cyber-attacks (Hansman & Hunt, 2005; Najafabadi *et al.*, 2014).

The emergence of attacks and malicious behaviors pose significant danger to computer security. They attempt to deviate from the deployed network security mechanism by exploiting the vulnerabilities found in the target networks. They disrupt normal network operations, such as causing network equipment to malfunction, attempting to overload a network, denying network services to authorized users, drastically reducing network throughput, scanning maliciously, and other similar activities (Hoque *et al.*, 2014; Najafabadi *et al.*, 2014).

Computer system attacks are achievable at several levels, ranging from data link layer to application layers. Attacks can also be classified as passive or active attacks (Pawar & Anuradha, 2015; Sheikh, 2020). An active attack occurs when attackers change system resources and cause effect to their operations. A passive attack occurs when attackers gather or make use of information from the systems but do not cause effect to the system resources (Liu

& Morgan, 2018; Srivastava, 2021). Password-based attacks, like dictionary-based attacks and brute-force attacks, are among the various types of computer attacks (Nagamalai *et al.*, 2011; Pawar & Anuradha, 2015).

The brute-force attack often referred to as a high-level attack, is among the most popular insurmountable challenges in today's computer system attacks. In brute-force, attackers attempt to log in by trying different passwords on the victim's machine to reveal the login passwords (Alata *et al.*, 2006; Anandita *et al.*, 2015; Hewlett-Packard:Top Cyber Security Risks Threat report, 2010; Hossain *et al.*, 2020; Najafabadi *et al.,* 2014; Vykopal, 2011). They generate password combinations using automated tools. Several smart brute-force attack tools are available, including Hydra, the most well-known brute-force attack tool, which comes pre-installed in Kali Linux operating system (Hossain *et al.*, 2020; Najafabadi *et al.*, 2014). Other tools highlighted by Kyaw *et al.* (2016) are John the Ripper, Cain and Abel. Brute-force attacks can be used against a wide range of services or protocols, with SSH and FTP being among the primary target for the attack.

In order to achieve a dictionary-based or brute-force attack, an attacker needs to have two important items; *a valid and existing list of usernames of the targeted system* and *a wordlist dictionary* (A text file containing a collection of words for use in the attacks). Therefore, one of the key first steps when attempting to gain access or launch an attack on a victim system or application is to enumerate usernames. An attacker first gathers the essential information about a user (Dave, 2013). Once intended usernames have been enumerated, targeted password-based attacks can be launched against found usernames.

Username enumeration is a sort of a passive attack (reconnaissance) that retrieves a list of existing and valid usernames from a system requiring user authentication (Li & Qiu, 2012; Virtue Security, 2021). This means an attacker could leverage to enumerate valid users on a targeted system (Rapid7, 2017). Since an attacker can quickly generate a list of legitimate usernames from the username enumeration attack, the time and effort necessary to brute-force a login is considerably reduced (Portswigger, 2018). However, it does not allow the attacker to log in immediately. Rather, it gives half of the necessary information which the attacker could use to run a password-based attacks such as brute-force to further exploit the obtained information. Once a list of validated usernames is created from the username enumeration attack, the attacker can perform another round of brute-force attacks. However, this time against the found usernames until access to the targeted system is eventually gained.

The username enumeration attacks can be initiated in any system that requires user authentication, including SSH server. Bhagwat and Kadwalkar (2020) described that Secure Socket Shell, SSH, is a typical software-based technique that deals with network security. It lets users over a network remotely connect and send data to the systems through a publicly exposed interface (Khandait *et al.*, 2021). Whenever users send data to the network, SSH automatically encrypts it. The SSH decrypts the data when it reaches its intended recipient (Čeleda *et al.*, 2019). The secured connection between the sender and receiver results in transparent encryption, making SSH a vital protocol in remote systems management (De Fuentes *et al.*, 2018). With emergencies like the COVID-19 crisis where millions of employees work from home, using their own devices and accessing corporate assets through their home Wi-Fi, SSH protocol plays even a major role in remote system management (Gupta *et al.*, 2020). The SSH enables secure logins to remote computer systems. Network administrators and web admins use it to securely access remote servers, switches, routers, virtualization platforms, and operating systems. In addition, most administrators routinely utilize an SSH client to secure file transfers, automate data transfers using SSH scripts, set up VPNs, test applications, reboot systems, modify file permissions, and manage user access (Fiterău-Broştean *et al.*, 2017). The SFTP video streaming, generating a single authorized session for many connections, remote backups, linking distant files to a local directory, and utilizing an encryption key for several accounts instead of individual passwords are just a few tasks. However, the above use cases highlight the importance of SSH protocol and its secure and effective remote systems management. It is not a complete security solution because of the different drawbacks SSH encounters (Ylonen, 2019).

Specific versions of OpenSSH experience suffering from a timing-based attack: If a valid username with a long password is given, the time taken to respond is noticeably longer than for an invalid username with a long password (Kannisto & Harju, 2017). The attacker can enumerate the service's registered usernames by exploiting how the server responds to forged queries. The server would respond with an authentication failure if the username does not exist, but the outcome would be different if the user exists. Other areas where username enumeration occurs are a website login page, and its '*forgot password'* functionality.

The demand for traffic anomaly detection in cybersecurity is increasing because of the enormous and rapid expansion of sophisticated computer attacks, including password-based attacks (Najafabadi *et al.*, 2014). Several approaches for detecting and mitigating password-based attacks, such as brute-force, have been suggested, developed, and deployed on a various systems and services, including SSH, FTP, and HTTP. However, in the era of cybersecurity, username enumeration attacks continue to be a problem. The majority of the recommended solutions focus on detecting and preventing password-based attacks, ignoring the fact that username enumeration is the first attack to identify and resist.

Inspired by the advancement and promising results of machine-learning techniques in traffic anomaly detection and mitigation (Elmrabit *et al.*, 2020; Mahesh, 2018; Nawir *et al.*, 2019), this study aims to develop a machine-learning model for detecting and preventing of username enumeration attack on SSH protocol by applying and analyzing machine-learning classifiers.

1.2 Statement of the Problem

In the existing literature, several detection and prevention approaches for password-based attacks, including dictionary-based or brute-force attacks on different services or protocols such as SSH, FTP, HTTP, have been proposed, developed and implemented. Some of these methods incorporated machine-learning techniques, and others incorporated traffic authentication techniques. However, most proposed methods focus on detecting and preventing password-based attacks generated by various intelligent tools such as Hydra and Medusa. The previous approaches failed however to put into consideration the following:

(i) At first sight, detection and prevention of the username enumeration attack (reconnaissance). Before any password-related attacks are deployed, an attacker must already have a valid and existing list of usernames of the targeted system and wordlist dictionaries.

(ii) The valid and existing list of usernames is usually acquired by deploying the username enumeration, which is the first phase before actually deploying password-related attacks such as brute-force.

In the current practices of detecting and preventing password-related attacks, studies indicated the attacks are based on a precompiled list of usernames of the targeted system (Najafabadi *et al.*, 2014; Owens & Matthews, 2008). However, in reality, precompiling thousands of users is almost impossible in a production environment. This is because most of these proposed approaches were lab-based studies where a precompiled list of usernames is possible due to its environmental nature, quite contrary to production environments where username enumeration is to be deployed first. Therefore, the detection and prevention of the username enumeration attack is highly needed to deny an opportunity for an attacker to retrieve a valid and existing list of usernames of the targeted system.

1.3 Rationale of the Study

The demand for traffic anomaly detection in cybersecurity is increasing because of the enormous and rapid expansion of sophisticated computer attacks, including password-based attacks (Najafabadi *et al.*, 2014). Several approaches for detecting and mitigating password-based attacks, such as brute-force, have been suggested, developed, and deployed on a various systems and services, including SSH, FTP, and HTTP. However, in the era of cybersecurity, username enumeration attacks continue to be a problem. The majority of the recommended solutions focus on detecting and preventing password-based attacks, ignoring the fact that username enumeration is the first attack to identify and resist. Inspired by the advancement and promising results of machine-learning techniques in traffic anomaly detection and mitigation (Elmrabit *et al.*, 2020; Mahesh, 2018; Nawir *et al.*, 2019), this study aims to develop a machine-learning model for detecting and preventing of username enumeration attack on SSH protocol by applying and analyzing machine-learning classifiers

1.4 Research Objectives

1.4.1 General Objective

The general objective of the research is to develop a machine-learning model that detects and prevents username enumeration attacks on SSH protocol.

1.4.2 The Specific Objectives

(i) To review the existing approaches for anomaly detection and identify the requirements for the proposed model.

(ii) To develop a machine-learning based model for detecting and preventing username enumeration attack.

(iii) To evaluate the performance and validate the developed model.

1.5 Research Questions

This research intends to answer the following questions:

(i) What are the requirements for developing a model for detecting and mitigating username enumeration attack?

(ii) How can a username enumeration detection and prevention model be developed?

(iii) How well does the developed model perform?

1.6 Significance of the Study

Cybersecurity is an important field that plays a vital role in protecting the systems and networks against unauthorized access, modification, and destruction (Hossain *et al.*, 2020). Several attacks have been discussed in the cybersecurity era from their types, detections, and mitigation techniques. Among other attacks discussed in cybersecurity is brute-force attack with its detection and prevention approaches. However, brute-force attacks suffer from one major drawback; detection and prevention of username enumeration from first sight. Therefore, the proposed approach is meant to be implemented as a computer algorithm that will be the basis of future development of username enumeration detection and prevention systems that will improve the classification accuracy between anomaly and normal traffic. The approach is not meant to completely eliminate cyberattacks. However, it will be a means which will offer assistance when accurately classifying traffic and create a base for further research in the cybersecurity era. This research has provided the groundwork for future research in username enumeration attack detection and prevention, theoretically and practically. It demonstrated how the attack had been overlooked in developing brute-force attack detection methods. Furthermore, the study demonstrated how machine-learning integration with intrusion detection and prevention systems may contribute to the management and possibly eliminate computer attacks.

The study has also published its dataset and made it available to the research community through open access to further facilitate research in username enumeration attacks (Agghey, 2021).

1.7 Delineation of the Study

This study only involves the use of Common Vulnerabilities and Exposures (CVE) with the identification CVE-2018-15473 from the public exploit database (exploits-db, 2018). The CVE is entirely written in python. Furthermore, the exploitation only works in OpenSSH server from version 2.3 to version 7.7. It is important to note that this study has certain limitations. The experiments conducted in this study only works for version 2.3 up to version 7.7 of the OpenSSH server configured in the Linux Operating System. Despite the models' better performances, the dataset size obtained is insufficient for machine-learning tasks. Restricted computer resources may also impair the models' performances.

CHAPTER TWO

LITERATURE REVIEW

2.1 Brute-force Attack: The Overview

The brute-force attack is one of the numerous security threats that network service administrators must manage (Saito *et al.*, 2016). As Stiawan *et al.* (2019) defined, brute-force attack is password-related experimentation that uses a mixture of possible ASCII characters either in separation or combinations. According to Joshi *et al.* (2015), Kaynar (2016), Stiawan (2017), Stiawan *et al.* (2017) and kaspersky (2021), brute-force attack is an old attack technique but still prevalent and effectively used by attackers.

This sort of attack can be deployed into several services or protocols. However, SSH and FTP protocols have been the major victims of brute-force attacks. Studies steered by Javed and Paxson (2013) on distributed brute-force SSH attack. Najafabadi *et al.* (2014) on SSH brute-force attack, Stiawan *et al.* (2019) on FTP brute-force in IoT network and by Hossain *et al.* (2020) on FTP and SSH brute-force are good examples to how prone these protocols are to this attack. The web application is another area where brute-force attack can prevail, as discussed by Hofstede *et al.* (2017).

Attackers in a brute-force attack attempt to log in to the aforementioned protocols to reveal the user's login credentials (Anandita *et al.*, 2015; Vykopal, 2011). A normal brute-force attack tests for the correct user and password combination, usually without knowing if the system user exists. It uses trial-and-error for checking correct user login credentials (kaspersky, 2021). It can also be repeatedly done by using a precompiled list of user's accounts and passwords until successfully achieved (Lee *et al.*, 2016; Vizváry & Vykopal, 2013). Subject to the complexity of the password, brute-force can take a few minutes or a couple of years. Brute-force attack can simply be achieved with the least attacking experience and intervention with the help of password attack tools such as hydra (Hossain *et al.*, 2020).

Brute-force attacks may lead to life-threatening impacts, including stealing private sensitive information and data such as bank accounts or social security details (Najafabadi *et al.*, 2015). Shortly, in brute-force attacks, once attackers gain access to targeted systems, the damage potential is nothing short of catastrophic (techcesscyber, 2018).

2.2 Brute-force Attack: The Current Status

A brute-force attack is still a major threat to our computing environment to date. Several studies and reports have been demonstrated this. For example, in its 2020 data breach investigation report, Verizon indicated that more than eighty percent of all breaches occurred in hacking in one way or another involved the use of brute-force attacks (Nathan & Scobell, 2020). The studies conducted by Stiawan (2017), Kaynar (2016) and Joshi *et al.* (2015) highlighted that of all the total cyberattacks available; brute-force attack occupies twenty-five percent and "it is the most common form of attack to compromise servers facing the Internet" (Kavitha, 2011).

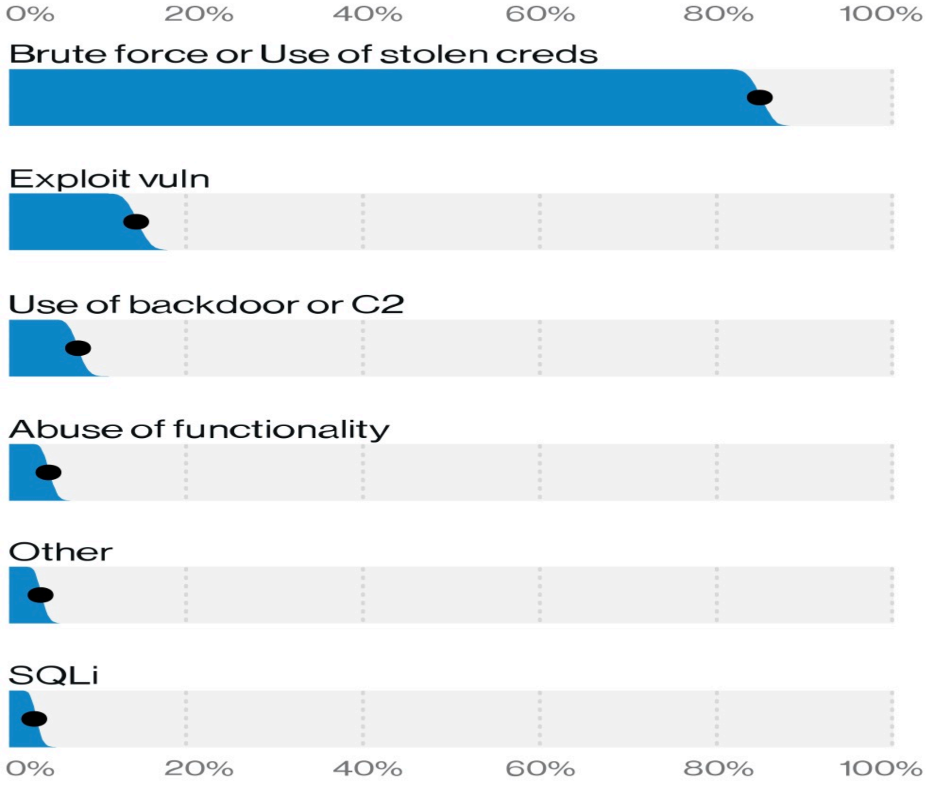


Figure 1: Top hacking varieties in breaches (https://Verizon/2020)

2.3 Complication of Brute-force Attack Detection

The username enumeration attack to get a list of existing usernames, works hand in hand with password-related attacks like brute-force attacks. Studies on brute-force attacks on different protocols have been carried out over the past couple of years. In various studies, the dominance of brute-force attacks has indeed been observed. For example, one of the studies that observed the prevalence of brute-force attacks is by Owens and Matthews (2008). They studied passwords and methods used in brute-force SSH attacks. Vykopal *et al.* (2009) studied and employed network-based dictionary attack detection methods on SSH protocol. Vykopal (2011) again presented a study on taxonomy and prevalence of brute-force attack. The three studies had promising results, however, none of them ever addressed the issue of username enumeration attack.

Satoh *et al.* (2012) analyzed SSH dictionary attack detection based on netflow. Their study suggested a novel attack detection method. Javed and Paxson (2013) examined stealthy distributed brute-force attack on SSH protocol. Vizváry and Vykopal (2013) observed flow-based detection of RDP brute-force attacks. The three approaches proved to be successful with promising results. The focus was on the detection of password-based attacks, but there was no effort on detecting username enumeration attacks.

Najafabadi *et al.* (2014) investigated brute-force attacks at the network level on SSH protocol. Joshi *et al.* (2015) analyzed a cryptographic technique for protecting text messages from brute-force and cryptanalytic attacks. Najafabadi *et al.* (2015) detected SSH brute-force attacks using aggregated flow data, and Lee *et al.* (2016) examined SSH brute-force in a multi-user environment. All of these studies provided outstanding results in a research community. However, the issue of username enumeration attack was never discussed.

Furthermore, Saito *et al.* (2016) detected and prevented brute-force attacks using disciplined IPs from IDs logs. Patil *et al.* (2017) examined and implemented a multilevel system to mitigate brute-force attacks for cloud security. Hofstede *et al.* (2017) investigated flow-based web application brute-force attacks and comprise detection method. Similarly, both studies attained outstanding results, but none focused on detecting the username enumeration attacks.

Stiawan *et al.* (2019) examined brute-force attack patterns in IoT networks on FTP and SSH protocols. Hossain *et al.* (2020) also investigated brute-force attacks detection in computer networks. However, studies mentioned above on different protocols and services focused on detecting password-related attacks and there was no effort to detect username enumeration attacks.

2.4 Supervised Learning on Brute-force Attack Detection

Machine-learning is a branch of artificial intelligence that allows machines to learn without having to be plainly programmed (Charbuty & Abdulazeez, 2021; Hamid *et al.*, 2016). Instead, machine-learning automates operations by skillfully taking each stage in a maintained way. Machine-learning contains several learning techniques categorized as supervised and unsupervised learning. This categorization is subjected to the existence or nonexistence of a labelled dataset. Supervised learning uses labelled samples to train the model, allowing it to anticipate comparable unlabeled samples. There are no training samples in unsupervised learning. Hence it relies on the arithmetical method of density approximation. Unsupervised learning is based on gathering or grouping data of the same types to uncover the underlying design of the data (Pahwa & Agarwal, 2019; Sharma & Kumar, 2017).

Machine-learning ability to recognize and give clues on real-life issues is greatly valued and thus lead to their appeal and perverseness. These accomplishments have steered the adoption of machine-learning in numerous fields (Apruzzese *et al.*, 2018; Jordan & Mitchell, 2015). Cybersecurity is among other field affected by this trend where intrusion detection systems (IDS) are advanced with machine-learning modules (Buczak & Guven, 2016). With their real-time response and adaptive learning process, machine learning algorithms are becoming particularly efficient in intrusion detection systems (Ahsan *et al.*, 2021). The advancement in machine learning techniques has presented promising and impressive results in detecting, identifying, classifying, predicting and mitigating a diverse range of cyberattacks. They exemplify unsurpassed choice over conventional rule-based algorithms (LeCun *et al.*, 2015).

Attacks and anomaly detection use supervised learning, where a known dataset is used to make classification or predictions. This training dataset contains input features and target values. The supervised learning algorithm then builds a model to make a prediction of the target values (Ndibwile *et al.*, 2015).

In literature, the most notable examples include the work done by Vykopal *et al.* (2009), whereby a decision tree classifier in supervised learning was adopted to demonstrate and describe the novel network-based approach on detection of dictionary-based attack along with the capability to realize all successful attacks. According to their study, SSH break-in attempts at a flow level were examined, revealing a dictionary attack pattern. The evaluation was accomplished in a large high-speed university network with promising results.

Another work was analyzed by Satoh *et al.* (2012) on detecting SSH dictionary-based attack detection using machine learning and subsequently suggested two novelty detection elements. In this approach, the combination of these two elements contains four functions. The first three functions recognize transition points of a sub-protocol through flow features and machine learning algorithms. The last function discovers an individual attack through differences in the inter-arrival time of an auth-packet and then differentiates between a successful and an unsuccessful attack through the existence of a connection protocol.

Moreover, the study done by Najafabadi *et al.* (2014) implemented several classifiers to develop models for detecting brute-force attack on SSH protocol at the network level. The study indicated that four different supervised learning classifiers were used to enable comparative study on the efficiency of learned models in distinguishing the brute-force traffic from the normal one. In their work, the dataset was generated from a live production network for over a 24 hours period. The study highlighted that the learned models were effective in detecting brute-force attack with a high rate of detection and low false alarms.

Hynek *et al.* (2020) proposed a study on detecting redefined brute-force attacks using a machine-learning approach. Their study used extended IP flow features obtained from backbone network traffic and machine-learning algorithms to differentiate successful and unsuccessful login. The dataset generated from a real environment using a wide assessment of captured traffic steered developing a machine learning model that successfully reduced the number of false positives with similar sensitivity levels.

Furthermore, the study done by Hossain *et al.* (2020) proposed the adoption of supervised learning and deep learning on detecting brute-force attack on two protocols, SSH and FTP, at the network level. In their work, Long Short-Term Memory (LSTM) and five different classifiers; J48, naïve Bayes (NB), decision table (DT), random forest (RF) and k-nearest neighbor were used for extra protection. The study elaborated that a well-known dataset from CICIDS2017 (Sharafaldin *et al.*, 2018) was used. The developed models learnt the traffic features and identified the ones with FTP and SSH brute-force attacks and those without.

2.5 Research Gap

All the aforementioned studies have focused and achieved excellent results on detecting and mitigating password-related attacks such as brute-force that are generated by various password attack tools. However, none of them has adequately included and addressed the issue of detection and mitigation of username enumeration attack. Considering that for any password-related attack to be launched, an attacker must have gathered all information, including the list of usernames of the targeted system obtained from the usernames enumeration attack. Therefore, the detection and prevention of the username enumeration attack is highly needed in order to deny an opportunity for an attacker to retrieve a valid and existing list of usernames of the targeted system.

CHAPTER THREE

MATERIALS AND METHODS

3.1 Introduction

This Chapter discusses the materials and methods used in this study in details. It discusses the study area, experimental setup, attack scenario, data generation and labelling, research framework, data preprocessing and models development.

3.2 Study Area

The study is a laboratory-based, conducted at Nelson Mandela African Institution of Science and Technology's laboratory in Arusha. We chose Mandela's lab because it has all the necessary equipment, including high-performance computers (HPC), important for this study. All the experiments in the laboratory were carried out in a closed-environment network. A closed-environment network is a private network with no external connectivity that is only accessible to approved devices.

3.3 Dataset Generation

The generation of a dataset for this study was achieved through the use of public exploit and normal traffic packets capture (*pcap*) retrieved from several public training repositories. The dataset generation process involved several subphases in this work, including experimental setup, attack scenario, data collection and labelling.

3.3.1 Experimental Setup

The attack simulation was carried out in a closed-environment network consisting of a victim machine, penetration testing platform, data collection point and benign users. The victim machine – SSH server was registered with thousands of users. The SSH server was a patched version of OpenSSH server 7.7 (*OpenSSH*, 2021) that listens on standard TCP port 22 for inbound and outbound traffic. We chose this version because the attack occurs between version 2.3 and 7.7 (exploits-db, 2018). The SSH server runs on Ubuntu Linux 20.04 (x64) with a 2.8 GHz Intel Core i7 CPU with a16GB RAM computer. A penetration testing platform - Kali Linux 2020.4 (x64) with kernel version 5.9.0 targets this SSH server. This penetration platform operates on a machine with 16 GB of RAM and a 3.4 GHz Intel Core i7 CPU. The data collection point installed with network monitoring tools collected all the traffic flowing through the network topology. The data collection server runs on Linux Mint 20.2 with 16 GB RAM computer and a 2.8 GHz Intel Core i7 CPU. The benign users represented normal users accessing the network. The IP addresses for the SSH server, penetration testing system and data collection server are 192.168.56.115, 192.168.56.117 and 192.168.100.116, respectively, and are in the private IPv4 range. Table 1 shows the summary of the devices used and Fig. 2 shows the network topology of the experimental setup.

Table 1: Summary of devices used in experimental setup

Device	Operating System	RAM	CPU	IP Address
Victim Machine	Ubuntu 20.04	16 GB	2.8 GHz Intel Core i7	192.168.56.115
Penetration Platform	Kali Linux 2020.4	16 GB	3.4 GHz Intel Core i7	192.168.56.117
Data Collection Point	Linux Mint	16 GB	2.6 GHz Intel Core i7	192.168.56.116
Benign Users	Various	Various	Various	192.168.56.XX

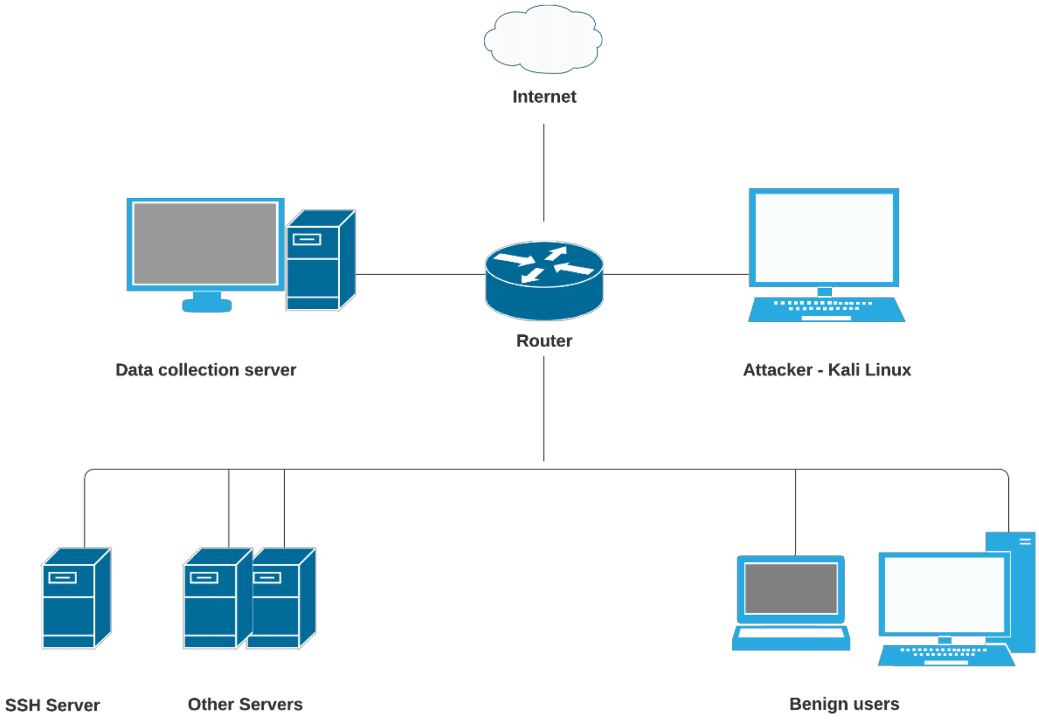
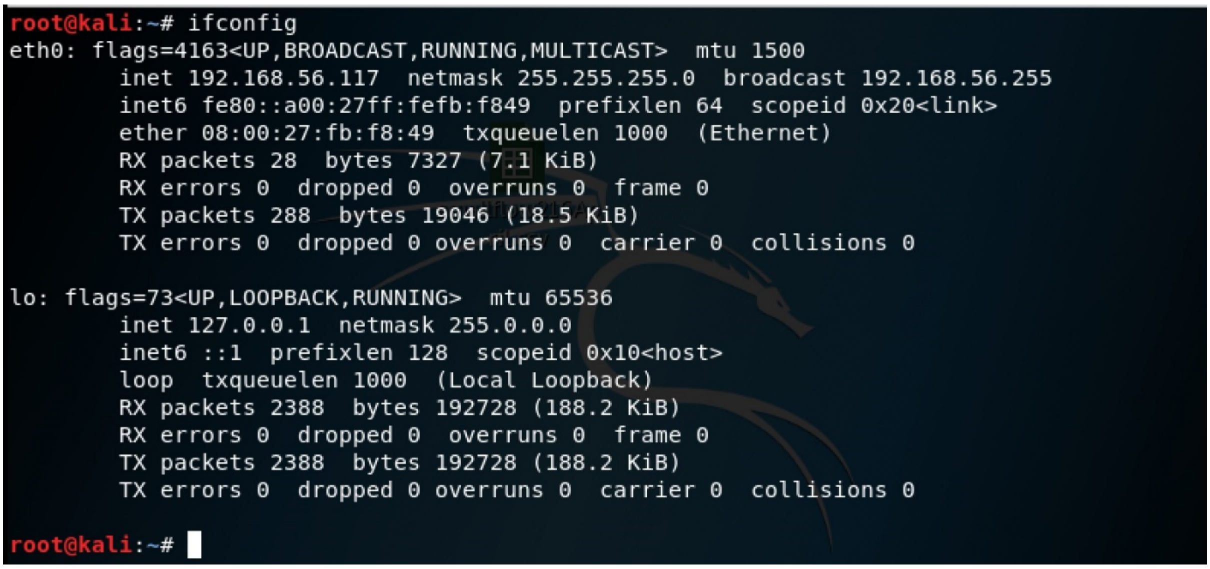


Figure 2: Network topology of the experimental setup

3.3.2 Attack Scenario

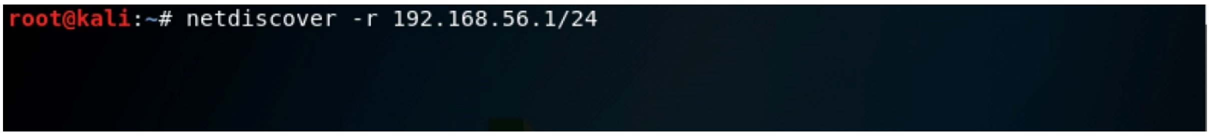
The attack simulation was launched from Kali Linux, a penetration testing platform, to SSH server, a victim machine. The common vulnerabilities and exposures (CVE) with the identification number CVE-2018-15473 retrieved from the public exploits database (exploits-db, 2018) was used to achieve this. The CVE is developed entirely in Python language. Before launching an attack from the penetration testing platform to the victim machine, information gathering and scanning steps were conducted. The information-gathering step was used to identify the IP Address of the penetration testing platform and victim machine using the *ifconfig* and *netdiscover* command, respectively. Figure 3 up to Fig. 5 show the information-gathering step. The scanning step examined all the protocols and services available to the victim machine using the *Nmap* command shown in Fig. 6 and Fig. 7. The attack was launched after the information gathering and scanning phase using the CVE mentioned above. It was accomplished by employing the attack command in Fig. 8.

Figure 9 depicts the attack's output by listing all the usernames found on the SSH server, including the root account. It displays a list of all existing usernames by indicating "*valid user"* and *"not a valid user"* for those not found in the system. To get a mix of normal and attack traffic, the tcpreplay tool (cite) was used to initiate a *pcap* file of normal traffic obtained from the public training repository (Stratosphere IPS, 2019).



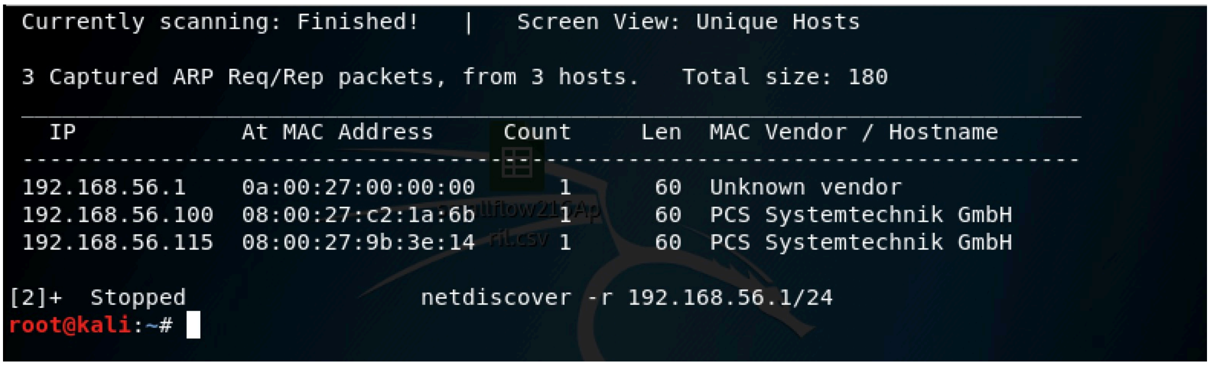
```
root@kali:~# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.56.117  netmask 255.255.255.0  broadcast 192.168.56.255
        inet6 fe80::a00:27ff:fefb:f849  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:fb:f8:49  txqueuelen 1000  (Ethernet)
        RX packets 28  bytes 7327 (7.1 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 288  bytes 19046 (18.5 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 2388  bytes 192728 (188.2 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2388  bytes 192728 (188.2 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@kali:~# 
```

Figure 3: The *ifconfig* command to identify the IP Address of penetration platform



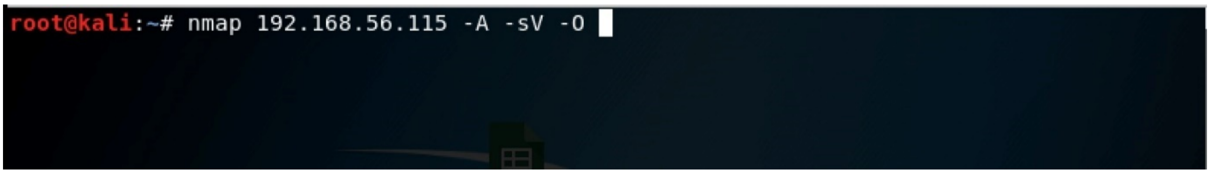
```
root@kali:~# netdiscover -r 192.168.56.1/24
```

Figure 4: The *netdiscover* command to identify the IP Address of the victim machine



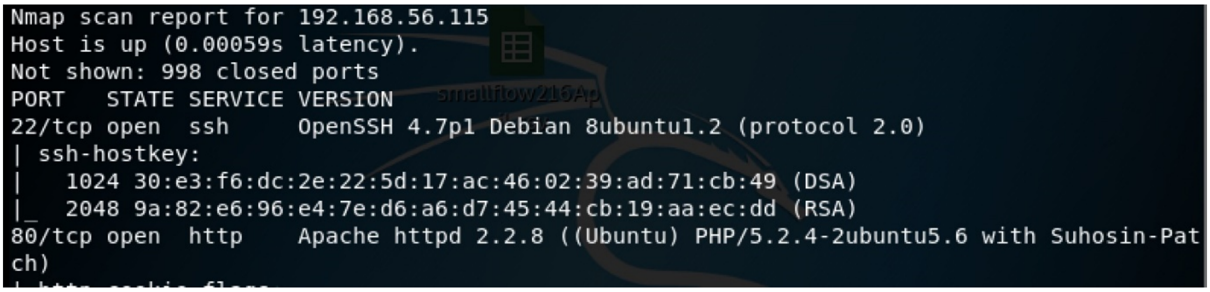
```
 Currently scanning: Finished!   |   Screen View: Unique Hosts

 3 Captured ARP Req/Rep packets, from 3 hosts.   Total size: 180
 _____________________________________________________________________________
   IP            At MAC Address     Count     Len  MAC Vendor / Hostname
 -----------------------------------------------------------------------------
 192.168.56.1    0a:00:27:00:00:00      1      60  Unknown vendor
 192.168.56.100  08:00:27:c2:1a:6b      1      60  PCS Systemtechnik GmbH
 192.168.56.115  08:00:27:9b:3e:14      1      60  PCS Systemtechnik GmbH

[2]+  Stopped                 netdiscover -r 192.168.56.1/24
root@kali:~#
```

Figure 5: Output of *netdiscover* command



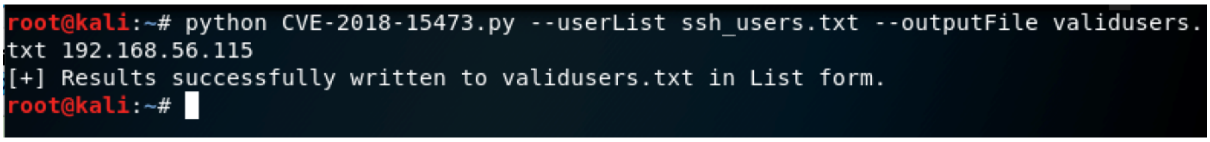
```
root@kali:~# nmap 192.168.56.115 -A -sV -O
```

Figure 6: The *nmap* command to scan open ports and services



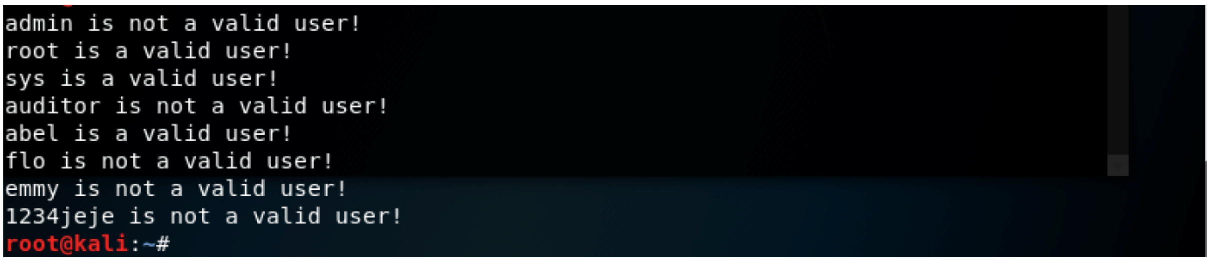
```
Nmap scan report for 192.168.56.115
Host is up (0.00059s latency).
Not shown: 998 closed ports
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 4.7p1 Debian 8ubuntu1.2 (protocol 2.0)
| ssh-hostkey:
|   1024 30:e3:f6:dc:2e:22:5d:17:ac:46:02:39:ad:71:cb:49 (DSA)
|_  2048 9a:82:e6:96:e4:7e:d6:a6:d7:45:44:cb:19:aa:ec:dd (RSA)
80/tcp open  http    Apache httpd 2.2.8 ((Ubuntu) PHP/5.2.4-2ubuntu5.6 with Suhosin-Pat
ch)
```

Figure 7: The output of *nmap* command



```
root@kali:~# python CVE-2018-15473.py --userList ssh_users.txt --outputFile validusers.
txt 192.168.56.115
[+] Results successfully written to validusers.txt in List form.
root@kali:~#
```

Figure 8: Username enumeration command



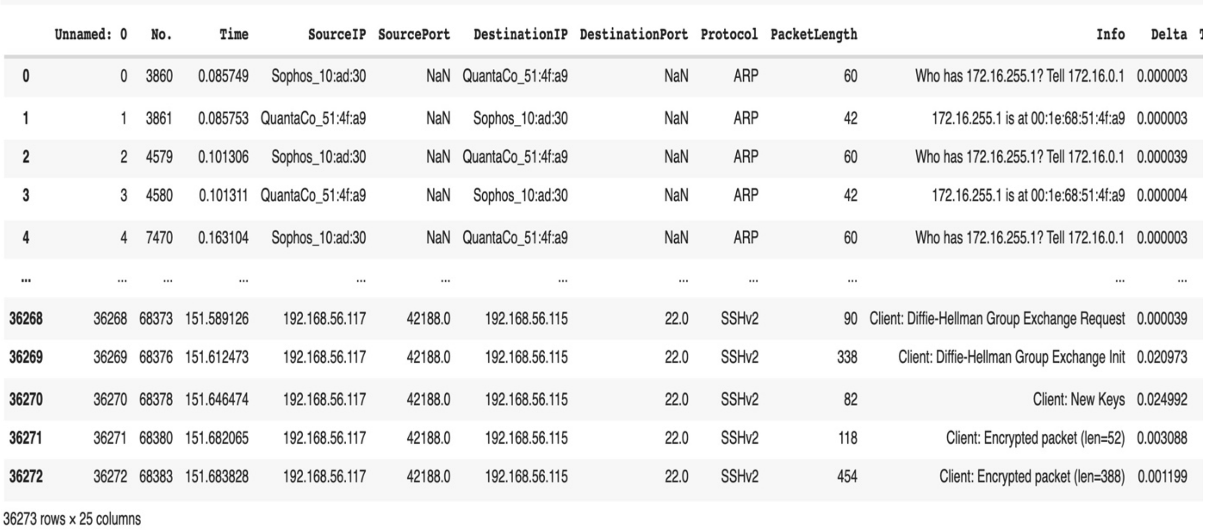
```
admin is not a valid user!
root is a valid user!
sys is a valid user!
auditor is not a valid user!
abel is a valid user!
flo is not a valid user!
emmy is not a valid user!
1234jeje is not a valid user!
root@kali:~#
```

Figure 9: Output of username enumeration

3.3.3 Data Collection and Labelling

The dataset was straightly collected from our closed-environment network using network monitoring tools *Wireshark* (Wireshark, 2021) and *tcpdump* (tcpdump, 2021) installed in the data collection point. The study chose *Wireshark* and *tcpdump* since they are open source and support more than 1100 protocols with detailed information. Additionally, huge community support and the ability to filter packets during and after capturing make them the most preferred networking monitoring tools. This study used both experimental and simulation research methods to collect raw packet data. The experiments were conducted by simulating attacks in for nine consecutive days. A total of 36273 raw packet data were collected, each containing 25 features with label exclusive. The packet data collected were then given their corresponding labels, username enumeration attack, non-username enumeration attack with the help of the domain experts. We chose the terms "username enumeration attack" and "non-username enumeration" instead of the traditional "attack" and "normal" label notations since normal traffic data could contain attacks other than username enumeration attack, which is the focus of our research. Since the goal of this study is to detect username enumeration attacks, we found that labeling dataset in that way is more suitable.

The username enumeration attack corresponds to the attack traffic while non-username enumeration traffic corresponds to the normal traffic. This traffic reflects different services, including emails, DNS, HTTP, web, a few to mention. We finally managed to get a raw dataset comprising attack traffic and normal traffic. Figure 10 shows some features and entries obtained in a raw dataset.



	Unnamed: 0	No.	Time	SourceIP	SourcePort	DestinationIP	DestinationPort	Protocol	PacketLength	Info	Delta
0	0	3860	0.085749	Sophos_10:ad:30	NaN	QuantaCo_51:4f:a9	NaN	ARP	60	Who has 172.16.255.1? Tell 172.16.0.1	0.000003
1	1	3861	0.085753	QuantaCo_51:4f:a9	NaN	Sophos_10:ad:30	NaN	ARP	42	172.16.255.1 is at 00:1e:68:51:4f:a9	0.000003
2	2	4579	0.101306	Sophos_10:ad:30	NaN	QuantaCo_51:4f:a9	NaN	ARP	60	Who has 172.16.255.1? Tell 172.16.0.1	0.000039
3	3	4580	0.101311	QuantaCo_51:4f:a9	NaN	Sophos_10:ad:30	NaN	ARP	42	172.16.255.1 is at 00:1e:68:51:4f:a9	0.000004
4	4	7470	0.163104	Sophos_10:ad:30	NaN	QuantaCo_51:4f:a9	NaN	ARP	60	Who has 172.16.255.1? Tell 172.16.0.1	0.000003
...	...	...	...	...	...	...	...	...	...	...	...
36268	36268	68373	151.589126	192.168.56.117	42188.0	192.168.56.115	22.0	SSHv2	90	Client: Diffie-Hellman Group Exchange Request	0.000039
36269	36269	68376	151.612473	192.168.56.117	42188.0	192.168.56.115	22.0	SSHv2	338	Client: Diffie-Hellman Group Exchange Init	0.020973
36270	36270	68378	151.646474	192.168.56.117	42188.0	192.168.56.115	22.0	SSHv2	82	Client: New Keys	0.024992
36271	36271	68380	151.682065	192.168.56.117	42188.0	192.168.56.115	22.0	SSHv2	118	Client: Encrypted packet (len=52)	0.003088
36272	36272	68383	151.683828	192.168.56.117	42188.0	192.168.56.115	22.0	SSHv2	454	Client: Encrypted packet (len=388)	0.001199

36273 rows × 25 columns

Figure 10: Raw dataset collected before data preprocessing

3.4 Research Framework

Figure 11 depicts the research framework, which provides the study's logical flow and clear explanation of how the research was carried out, from data generation, data preprocessing, models developments and validation to the delivery of improved models. We firstly generated the data and obtained our dataset. The raw dataset is stored in cloud storage. We used Google Drive to store our dataset in this study. The dataset is then preprocessed, involving missing data treatment, categorical encoding, data projection and data reduction. The dataset is split into training and testing subsets before training on the models. In this work, we used the training subset on four different classifiers.

After training the proposed models, performances are then evaluated using different evaluation metrics, and the models' parameters are tuned to get an optimized model. Finally, the best selected model is deployed on an intrusion detection and prevention system (IDS/IPS) to enable automatically detection and prevention of username enumeration attacks. An intrusion prevention system (IDS) is a software or hardware-based system that detects and alerts unauthorized or undesired access attempts, modifications, or resource restrictions on a computer system (Abubakar & Pranggono, 2017). The intrusion detection system, in particular, aids in the detection of external and internal attacks perpetrated by both users and hackers (Jain, 2021).

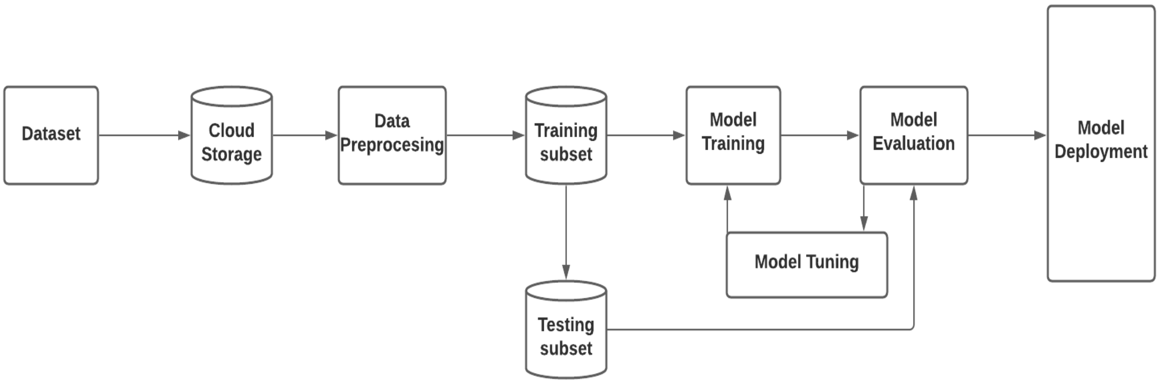


Figure 11: Research framework

3.5 Data Preprocessing

Data pre-processing is the data mining technique that transforms raw datasets into a readable and understandable format. Machine learning algorithms make use of the datasets in mathematical format, and such format is achieved through data pre-processing (Chandrasekar *et al.*, 2017; Huang *et al.*, 2015). In this work, data pre-processing involved treating missing values, encoding categorical values, data projection and data reduction. The preprocessed dataset was then fed to the model as input data. All the data pre-processing techniques were carried out using the scikit-learn library.

3.5.1 Missing Data Treatment

Missing data treatment in the dataset was conducted using the imputation technique. The imputation technique involves either deletion of missing values or exchanging/replacing them with estimations (Aljuaid & Sasi, 2016). In this work, the imputation technique was done by replacing the missing values with the estimations. We chose to replace the missing values rather than deleting them to avoid discarding a large proportion of the dataset and introducing biases. The imputation technique was also done for both categorical and numerical features. The *most frequent strategy* was used within each column for the categorical features to replace the missing values. For the case of numerical features, *a constant strategy* was implemented to replace the missing values.

3.5.2 Categorical Encoding

Categorical encoding aims to transform categorical values into numerical values. Categorical values/variables represent string values rather than a continuous number (McGinnis *et al.*, 2018). We convert categorical variables to numerical values because machine learning algorithms prefer numbers over strings which have no true order. There are many techniques used for categorical encoding. The most commonly used ones are label encoding and one hot encoding. This work used label encoding techniques to convert categorical variables to numerical values. We selected this approach over one hot encoding technique because it straightly converts categorical data to numerical data; hence, it does not increase the dimension of the dataset.

3.5.3 Data Projection

Data projection, also called feature scaling, scales the data values into a similar range. Feature scaling changes the appearance of the data and aids in the speeding up of the algorithm's calculations (Bollegala, 2017). The dataset utilized in this study includes variables with varying scales. As a result, the dataset was feature scaled to convert the feature vector into a format that is more suitable for machine learning algorithms. For feature scaling of the dataset, there are a variety of scalers available. *MinMaxScaler (), StandardScaler ()* are the most widely used ones. In this work, all features were scaled into the same predefined range using the *Min-Max scaling* method. *MinMaxScaler ()* scales the data so that all its values lie between 0 and 1.

3.5.4 Data Reduction

Data reduction intends to reduce the size of datasets using several techniques. In this study, data reduction was implemented using the feature selection technique. Features selection intend to find the best features in the dataset (Ahmad & Aziz, 2019). Hence, it aids in decreasing the number of irrelevant features that increase computational complexity, training time, dataset dimension and enhance performances (Wong *et al.*, 2021). This work used mutual information (MI) (Sharmin *et al.*, 2019) as a features selection technique. The selection of the mutual information technique was made due to its capacity to capture non-linear relations among variables. The MI also has the advantage of being able to compute both categorical and numerical variables, and deal with many classes (Rahmaninia *et al.*, 2020). Seven different features were selected from the dataset using the aforementioned features selection technique. The description of each feature is shown in Table 2.

Table 2: Description of the features selected

Feature name	Feature description
Time	Packet duration time
Packet Length	The length of the packet
Delta	Time interval between packets
Flags	Flags seen in the packet
Total Length	The total length of the packet
Source Port	The source port of the packet
Destination Port	The destination port of the packet

3.6 Machine Learning Model Development

The machine-learning model is the outcome of the machine-learning training procedures. Machine learning algorithms are used to train machine-learning models (Ahsan *et al.*, 2021). This work employs four distinctive machine learning algorithms/classifiers for the model development. We selected these classifiers because they have dissimilar features and lightweight computation. As discussed in Chapter two, they have also been used and shown performances in other intrusion detection-related studies. We also picked different classifiers in order to investigate a wider scale of investigation in username enumeration attack detection. Therefore, we examined K-Nearest- Neighbor (KNN), Naïve Bayes (NB), Random Forest (RF) and Decision Tree (DT) machine-learning classifiers.

3.6.1 Decision Tree (DT)

A decision tree is a widely known machine-learning classifier created in a tree-like structure (Cherfi *et al.*, 2018). Because of decision trees' precision across many data types and their ease of analysis, have discovered a diversity of implementation domains (Mazraeh *et al.*, 2019). The decision tree contains the internal nodes representing attributes and leaf nodes representing the class label. The root node, a notable attribute for data separation, is first selected to form a classification rule. The path is then chosen from the root node to the leaf nodes (Adel *et al.*, 2017; Priyanka & Kumar, 2020). The root node and internal nodes are referred to as non-terminal nodes and are associated at the decision stage. The leaf nodes are collectively referred to as terminal nodes, exemplifying final classification. In a decision tree, any path from the root to the leaf node is characterized by a data separating sequence until a Boolean outcome is reached (Adel *et al.*, 2015; Liang *et al.*, 2019). Thus, it is a structural illustration of the internal nodes and links in the knowledge relationship (Charbuty & Abdulazeez, 2021). Figure 12 illustrates the structure of the decision tree.

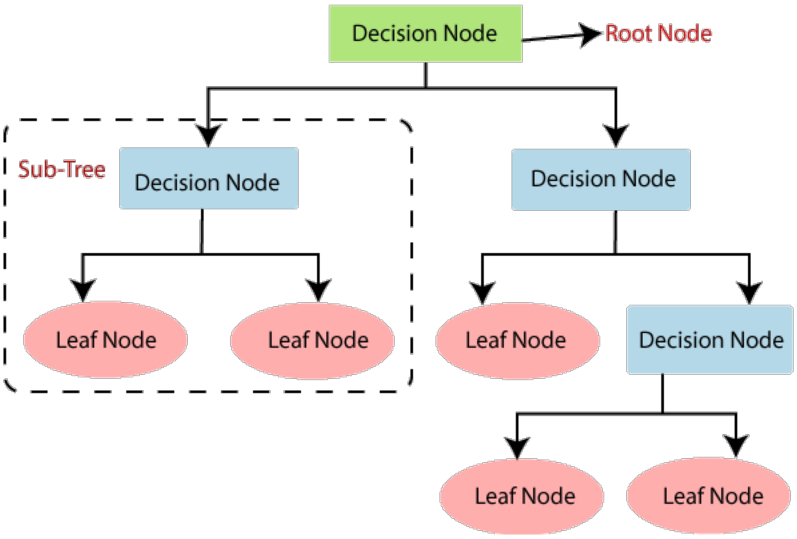


Figure 12: The structure of the Decision Tree (https://javapoint/2021)

The decision tree classifier operates by recognizing associated attribute values as input data and produces decisions as output. First, the tree is encoded as a string of symbols for computation reasons. The string is then decoded, and pointers are assigned to each training data to determine the proper classification route (Cherfi *et al.*, 2018). Finally, the classifier examines the training data to recognize the attributes with higher information gain than the rest. Information gain tells how important a given attribute is. It decides the ordering of the nodes of the decision tree classifier. As such, an attribute can effectively categorize or classify the data. The root node is the attribute with the highest information gain since it instantly classifies the training data into different classes. The attribute's information gain is calculated as follow:

$$Information\ Gain = Entropy\ - Weighted\ Entropy \qquad (1)$$

Mathematically can be presented as:

$$Gain(A)\ \ = H(Set) - (\ w_1\ \times H(a_1) + w_2\ \times H(a_2) + \ldots + w_m\ \times H(a_m) \qquad (2)$$

Where

a_1, a_2, a_m are the different values of attribute A.

w_1, w_2, w_m are the weights of the subsets split by using the value of attribute A

$H\ (Set)$ is Entropy.

Entropy measures the purity or impurity of data instances that are often employed in information theory. In addition, the entropy of instances can be used to determine their homogeneity. The entropy of the data instances is calculated as follow:

$$H(Set) = -P_1\ \times log_2 P_1\ -\ P_2\ \times\ log_2 P_2 \qquad (3)$$

Where:

P_1 is the proportion of the first decision.

P_2 is the proportion of the second decision

Similarly, each attribute is examined one by one in the increasing order of information gain and constructs the tree. Consequently, the attributes with the next lower-level information gain are used to divide the training data into sub-part until each training data record is given its class label (Patil & Kulkarni, 2019). When the class labels for training data samples are known, therefore decision trees are built by analyzing them.

3.6.2 Random Forest

Random Forest (RF) is another dominant machine-learning classifier under supervised learning algorithms (Li *et al.*, 2020) introduced by Breiman (2001). Similarly, to decision tree classifiers, random forest is also used in machine-learning classification problems. Because of its ability to deal with categorical and numerical attributes, minimal training time complexity, quick prediction, robustness to unbalanced datasets, embedded feature selection approach, and inherent metrics to order attributes by relevance, the random forest classifier is suitable for classification problems (Resende & Drummond, 2018). Random Forest is an ensemble learning that consists of decision trees. This is because random forest uses a bootstrap aggregation technique to combine a variety of data sets and a feature selection process to anticipate the outcome. It is formally defined as a classifier comprising a group of tree-structured classifiers. Equation 4 shows the mathematical representation of a random forest classifier.

Similarly, Random Trees blend single model trees with Random Forest concepts, where each node comprises k randomly select features in the tree. As a result, it improves the accuracy of Random Forest over a single tree. The operation of the Random Forest classifier is conducted in two different steps. The first step creates the asymmetrical forest of the specified dataset, and the second one predicts the classifier acquired in the initial stage (Bhavani *et al.*, 2020). Fig. 13 illustrates the structure of the random forest.

$$\{h_k\ (X, T_k)\}, k = 1, 2. \dots, L \qquad (4)$$

Where:

T_k are random samples that are dispersed in a uniform manner.

X each tree casts vote for the most popular class.

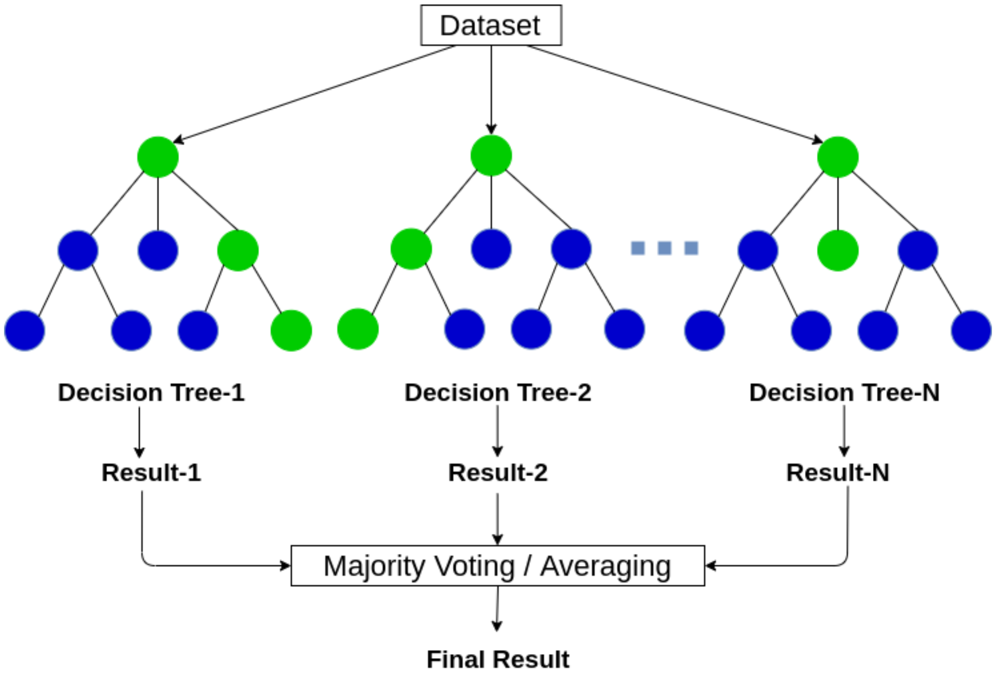


Figure 13: The structure of Random Forest (https://ai-pool/2021)

3.6.3 Naïve Bayes

Naïve Bayes (NB) is a common probabilistic machine-learning classifier used in classification or prediction problems. It is referred to as a probabilistic classifier because it functions by computing the probability of a certain class in a specified dataset. Naïve Bayes contains two probabilities: class and conditional probabilities. Class probability is the ratio of every class instance occurrence to the total instances. Conditional probability is the quotient of every feature occurrence for a certain class to the sample occurrence of that class (Alqahtani *et al.*, 2020; John & Langley, 2013). Equation 5 shows the mathematical presentation of Naïve Bayes.

$$f_i^{NB}(x) = \Pi_{j=1}^{n} P\left(X_{j=x_j} \middle| C = i\right) P(C = i) \tag{5}$$

Where:

$P\left(X_{j=x_j} \middle| C = i\right)$ is the class-condition probability distribution, which is defined as:

$$P\left(X_{j=x_j} \middle| C = i\right) = \frac{P(X = x|C = i)P(C=i)}{P(X=x)} \tag{6}$$

Where:

$X = (X_1, \ldots \ldots X_n)$ is a feature vector and C being a class.

Naïve Bayes classifier presumes that features are independent for a given class and contemplates association between the features (Han *et al.*, 2011). It assigns the class label to sample cases based on the most frequent values of the features. During the training phase, it determines the prior probability of each class using the occurrences of each feature for each class. Naive Bayes calculates the class posterior probability based on the class prior probability. It concludes that the predictor's result outcome for a particular class is independent of the values of another predictor. Naïve Bayes allocates the class label to the new data using the probabilities mentioned above (Mehmood & Rais, 2016). Figure 14 illustrates the structure of Naïve Bayes.

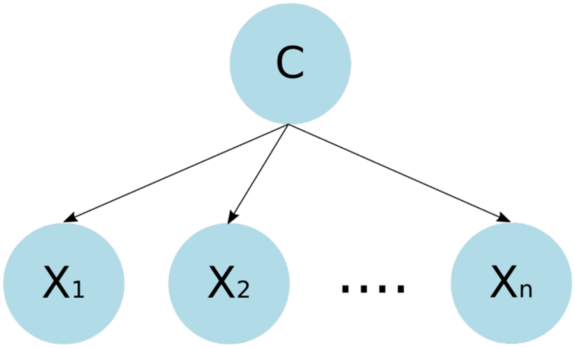


Figure 14: The structure of Naive Bayes network (https://mdpi/2021)

3.6.4 K-Nearest Neighbor

K-Nearest Neighbor (KNN) is a non-parametric classifier that has been applied to a variety of classification problems. The term non-parametric refers to the absence of any assumptions about the underlying data distribution. It means that the model structure is based on the given dataset (Malhotra *et al.*, 2017). The non-parametric nature of the classifier is quite useful in practice, as most real-world datasets do not adhere to mathematical theoretical assumptions. Other advantages that make the classifier quite useful in practices are the ability to work well with huge amounts of training data and resistance to noisy data (Gou *et al.*, 2019).

K-Nearest Neighbor considers three important elements in its classification manner: record set, distance and value of K. It functions by calculating the distance between sample points and training points. The smallest distance point calculated is the nearest neighbor (Bhatia & Vandana, 2010). The nearest neighbor is measured with respect to the value of k (in our case k=4); this defines the number of nearest neighbors required to be examined in order to define the class of sample data point (Soofi & Awan, 2017).

There are four distinct measures of calculating the smallest distance in K-Nearest Neighbor classifier. They are Euclidean distance, Manhattan distance, Minkowski distance, Hamming distance. However, this study only employed Euclidean distance measure as illustrated in Equation 7. Its ability to quickly calculate the shortest distance between two points makes it favorable among others.

$$d\,(x,y) = \sqrt{\Sigma_{i=1}^{k}(x_i - y_i)^2} \qquad (7)$$

3.7 Implementation

The implementation of model's development was conducted on a computer with 512 GB SSD storage, Intel® Core i7 2.8 GHz CPU, 16 GB 1600 MHz DDR3 RAM, AMD Radeon R9 M370X 2048 MB, Intel® Iris Pro 1536 MB that came pre-installed with Macintosh operating system. In addition, the GPU environment from Google Collaboratory with 25 GB RAM was used in the training phase. Python v3.7 (Turner *et al.*, 2018) was used as the programming language, with scikit-learn (Feurer *et al.*, 2019) as the library. The scikit-learn framework was chosen because it allows models to be deployed and interpreted across several devices.

3.8 Training Phase

In this study, two folds of experimentations were conducted. The first fold excludes source and destination ports as input features in the process of learning. The second one includes them as our input features. This is because network administrators sometimes customize the destination port to some different number other than the default port number for SSH protocol which is port 22.

All the classification models were trained using a subset of 80% data of the given dataset and the remaining subset of 20% to test the models. The train-test split ratio was based on Pareto (Dunford *et al.*, 2014) principle and was even for each classifier. From a dataset of 36 273 instances, we attained 29 018 instances for training and 7255 instances for testing for the two classes, as shown in Table 3.

Table 3: Dataset splitting

Dataset	SSH Username Enumeration Attack	Non-Username Enumeration Attack	Total
Training	15075	13943	29018
Testing	3769	3486	7255

3.8.1 Training Phase – Ports Exclusive

Four machine-learning classifiers, Decision Tree, Random Forest, Naïve Bayes and K-Nearest Neighbor were trained on two classes, username enumeration attack and non-username enumeration attack without including ports information as feature sets.

(i) Decision Tree – Training and Hyperparameters Tuning

The decision tree classifier was used to develop a username enumeration attack detection model in the training phase. In training the model, different hyperparameters were used.

Hyperparameters are configurable points in a machine learning model that can be tailored to a specific job or dataset. The hyperparameters used in the training phase include criterion, maximum depth, maximum features, maximum leaf nodes and splitter. All the values in each hyperparameter used were selected using Randomized Search CV. Randomized Search CV is an optimization algorithm that selects the optimal hyperparameter to fit the model from a list of hyperparameters given, as shown in Fig. 15.

Criterion

This hyperparameter specifies how the impurity of a split will be measured. It has two options; Gini or Entropy as metrics. The default value is Gini.

max_dept

This hyperparameter defines the maximum depth of the decision tree to avoid over-fitting. The value is set to none by default.

maxi_features

This determines the number of features when considering the best split.

maxi_leaf_nodes

This hyperparameter specifies the maximum number of leaf nodes for the tree to take.

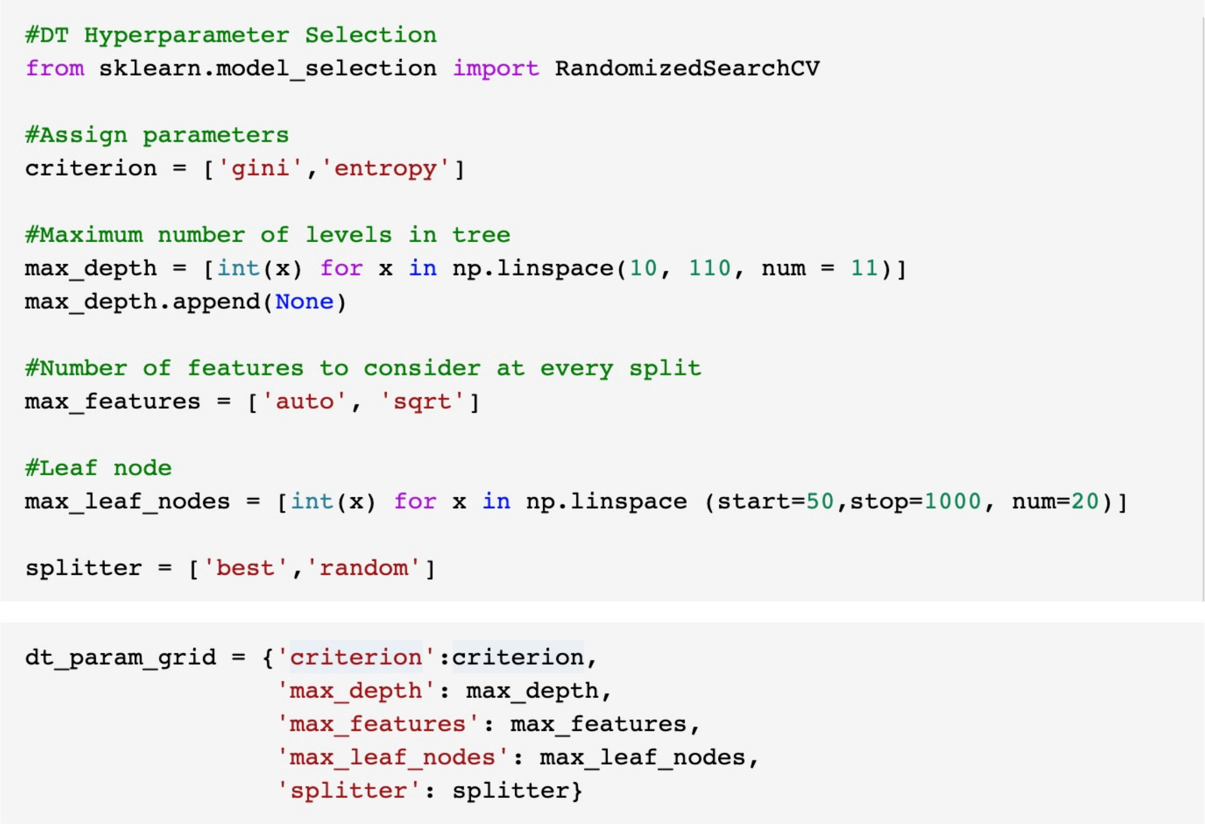
Split

This hyperparameter specifies how the decision tree looks for a feature split.

The criterion hyperparameter was set to Gini during training, which was the default value. Gini has a lower computing complexity than other metrics like Entropy, making it ideal for the training process. To avoid model over-fitting and to regularize the way the tree grows, the maximum depth parameter value was set to 50. The maximum features hyperparameter value was set to *auto* to determine the number of features to consider when splitting. The splitter hyperparameter value was set to *best* to decide how the model searches for the features. The model checks all of the features for each node and selects the best split. A tree's maximum number of leaf nodes was set to 950. Table 4 summarizes the hyperparameters used when training the Decision Tree classifier.

Table 4: Hyperparameters used when training the decision tree classifier

Classifier	Hyperparameter	Value
Decision Tree	Criterion	Gini
	Maximum depth	50
	Maximum features	Auto
	Maximum leaf nodes	950
	Splitter	Best



```
#DT Hyperparameter Selection
from sklearn.model_selection import RandomizedSearchCV

#Assign parameters
criterion = ['gini','entropy']

#Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)

#Number of features to consider at every split
max_features = ['auto', 'sqrt']

#Leaf node
max_leaf_nodes = [int(x) for x in np.linspace (start=50,stop=1000, num=20)]

splitter = ['best','random']
```

```
dt_param_grid = {'criterion':criterion,
                 'max_depth': max_depth,
                 'max_features': max_features,
                 'max_leaf_nodes': max_leaf_nodes,
                 'splitter': splitter}
```

Figure 15: Illustration of how Randomized Search CV is performed to get the best values for each parameter used in the Decision Tree classifier

(ii) Random Forest – Training and Hyperparameters Tuning

Training random forest classifier included six different hyperparameters. They were bootstrap, maximum depth, maximum features, minimum sample leaf, minimum sample split and a number of n estimators.

Bootstrap

This determines the sampling approach for data points

Maximum depth

It defines the maximum depth of the model.

Maximum features

Number of features when considering the best split.

Minimum sample split

The minimum number of data points inserted in a node before split

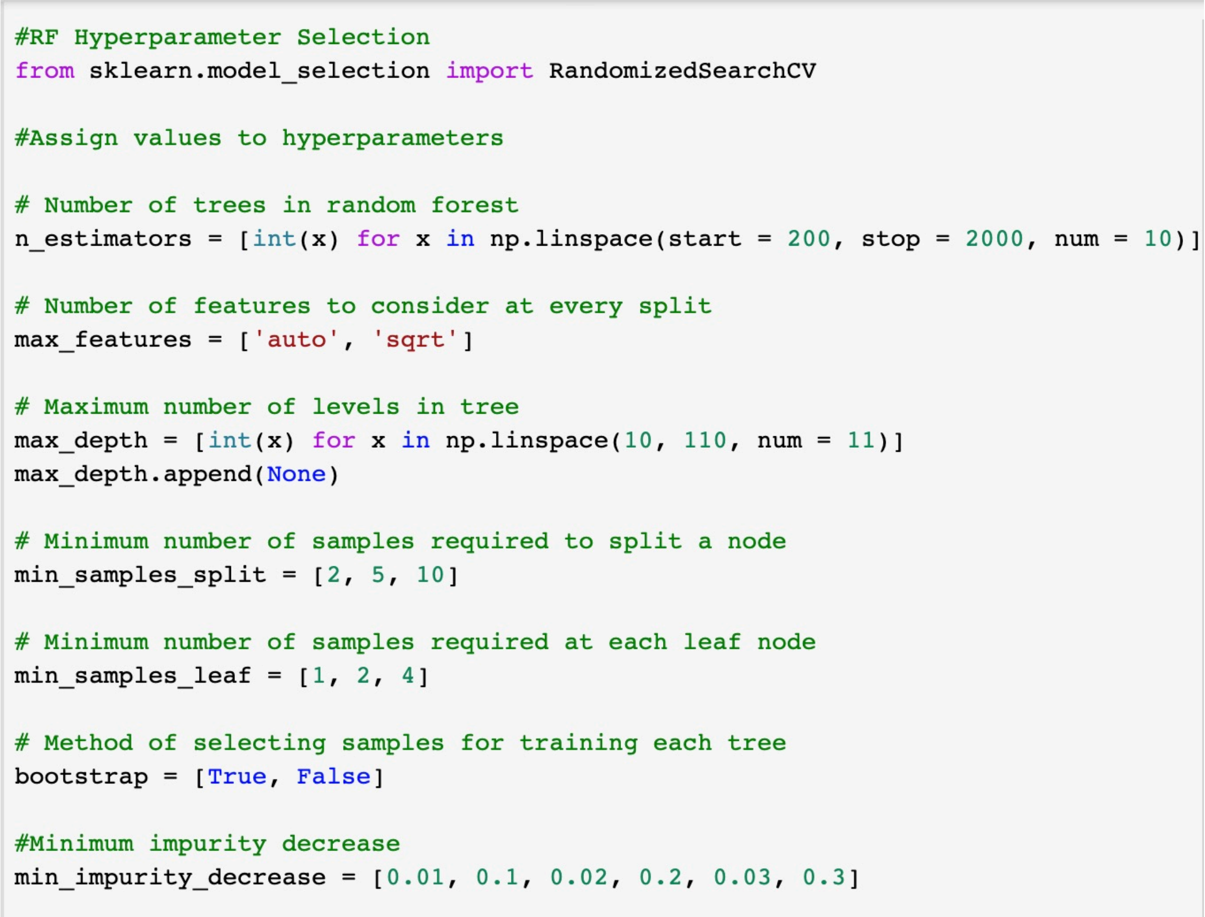
Minimum sample leaf

It defines the minimum number of data points required to be at a leaf node

N estimators

This hyperparameter specifies the number of trees in the forest.

In training the random forest classifier, the value of the n estimators hyperparameter was set to 1600. This hyperparameter's value is always higher to make the prediction stable and strong. The n estimator value was set to 1600 based on the result of the optimization algorithm used, The Randomized Search CV, as shown in Fig. 16. The minimum sample leaf value was set to 1 to guarantee the minimum number of samples required in every leaf node. The minimum sample split value was set to 5 to allow internal node split. The maximum depth parameter value was set to 90 to avoid model over-fitting and normalizing the tree’s evolution. The maximum features hyperparameter value was set to *Auto* to select the number of features while splitting. The sampling approach value was set to true in the bootstrap hyperparameter. The summary of hyperparameters used when training the random forest classifier is shown in Table 5.



```
#RF Hyperparameter Selection
from sklearn.model_selection import RandomizedSearchCV

#Assign values to hyperparameters

# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]

# Number of features to consider at every split
max_features = ['auto', 'sqrt']

# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)

# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]

# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]

# Method of selecting samples for training each tree
bootstrap = [True, False]

#Minimum impurity decrease
min_impurity_decrease = [0.01, 0.1, 0.02, 0.2, 0.03, 0.3]
```

Figure 16: **Illustration of how Randomized Search CV is performed to get the best values for each parameter used in the Random Forest classifier**

Table 5: **Hyperparameters used when training Random Forest classifier**

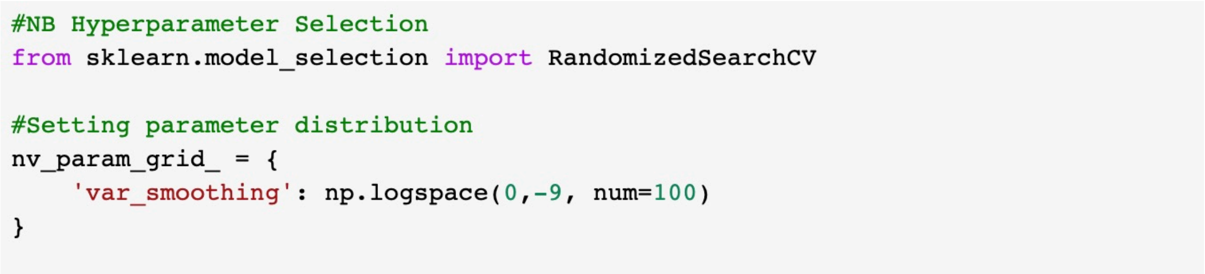
Classifier	**Hyperparameter**	**Value**
Random Forest	Bootstrap	True
	Maximum depth	90
	Maximum features	Auto
	Minimum sample leaf	1
	Minimum sample split	5
	N estimators	1600

(iii) Naïve Bayes – Training and Hyperparameters Tuning

Var_smoothing is the only hyperparameter for tuning in the naïve Bayes classifier. It stabilizes the calculation to smooth or widen the curve in order to accommodate more samples that are further distant from the distribution mean. Table 6 below show the summary hyperparameter used in the naïve Bayes classifier. The selected parameter value was obtained using Randomized Search CV, as shown in Fig. 17.

Table 6: **Hyperparameters used when training Naive Bayes classifier**

Classifier	Hyperparameter	Value
Naïve Bayes	Var_smoothing	$2.848035868435799 * 10^{-5}$



```
#NB Hyperparameter Selection
from sklearn.model_selection import RandomizedSearchCV

#Setting parameter distribution
nv_param_grid_ = {
    'var_smoothing': np.logspace(0,-9, num=100)
}
```

Figure 17: Illustration of how Randomized Search CV is performed to get the best values for each parameter used in the Naive Bayes classifier

(iv) K-Nearest Neighbor – Training and Hyperparameters Tuning

Training of K-Nearest Neighbor classifier included three hyperparameters n_neighbors, leaf_size and p.

n_neighbor

This hyperparameter defines the number of neighbors to be used.

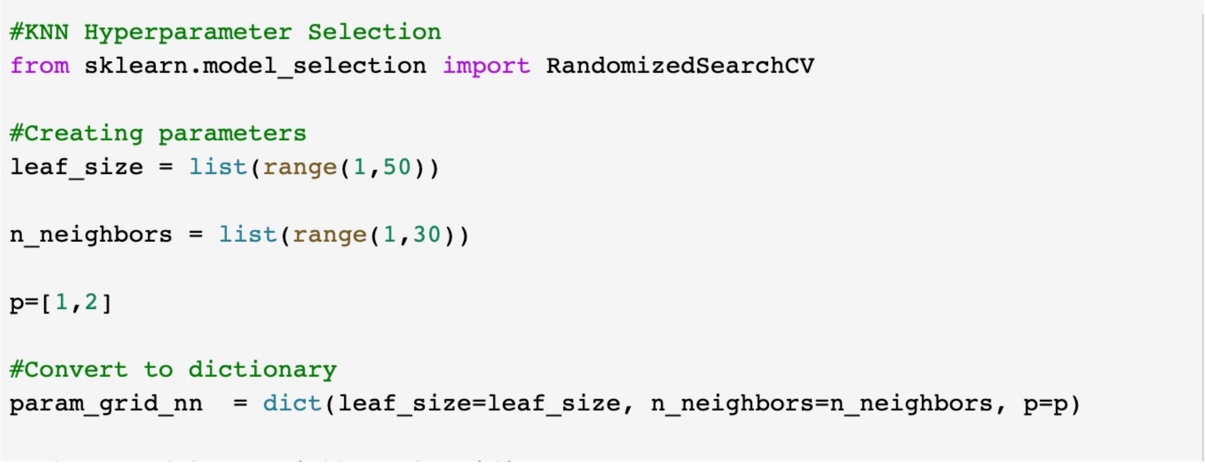
leaf_size

This hyperparameter determines tree construction and memory required to store the tree.

p

This parameter specifies the distance measure used in training the model. If set to 1 equal to Minkowski, 2 equals to Euclidean distance measures.

During the training, the p hyperparameters value was set to 2. It was set to 2 because the Euclidean distance measure was used. Next, the leaf size value was set to 7 to determine the speed of tree construction and the memory required to store the created tree. Finally, the N_neighbor hyperparameter value was set to 4. It determines the number of neighbors to be used. As shown in Fig. 18, all the hyperparameter values were selected using an optimization algorithm, Randomized Search CV. Table 7 shows the summary of hyperparameters used when training the K-Nearest Neighbor classifier.



```
#KNN Hyperparameter Selection
from sklearn.model_selection import RandomizedSearchCV

#Creating parameters
leaf_size = list(range(1,50))

n_neighbors = list(range(1,30))

p=[1,2]

#Convert to dictionary
param_grid_nn  = dict(leaf_size=leaf_size, n_neighbors=n_neighbors, p=p)
```

Figure 18: Illustration of how Randomized Search CV is performed to get the best values for each parameter used in the K-Nearest Neighbor classifier

Table 7: Hyperparameters used when training K-Nearest Neighbor classifier

Classifier	Hyperparameter	Value
Naïve Bayes	N	4
	Leaf size	7
	P	1

3.8.2 Training Phase – Port Inclusive

Towards studying the effect of including ports information as input feature sets, the same classifiers were used in the second fold of the experimentation/training phase to develop username enumeration attack detection models. The same experiment conducted when excluding source and destination ports has been repeated but with the inclusion of ports information in the training process. As discussed above, network administrators sometimes customize destination ports to other numbers than the default SSH port number 22. Therefore, the same dataset with the same split ratio was utilized in this case. Table 8 shows the configuration and hyperparameters used when training four classifiers, including ports information as input features.

Table 8: Hyperparameters used for models training - Ports inclusive

Classifier	Hyperparameter	Value
Decision Tree	Criterion	Entropy
	Maximum depth	60
	Maximum features	Auto
	Maximum leaf nodes	500
	Splitter	Best
Random Forest	Bootstrap	True
	Maximum depth	30
	Maximum features	Sqrt
	Minimum sample leaf	1
	Minimum sample split	5
	N estimators	400
Naïve Bayes	Var_smoothing	0.022328467394420651
K-Nearest Neighbor	N	8
	Leaf size	5
	P	2

3.9 Evaluation

Model evaluation is a degree of how well the trained model generalizes to a new unseen dataset. The unseen dataset for this case was 20% remaining of the dataset, the test subset. Model evaluations aimed at estimating the generalization accuracy of the new dataset. The performance of machine-learning models can be evaluated using different evaluation or performances metrics. The choice of performance metrics depends on a given machine-learning task such as classification, regression, and a few to mention. In this work, the performance metrics to evaluate the effectiveness of the developed models were computed in terms of precision, recall and overall accuracy. In addition, the Receiver Operating Characteristics (ROC) curve was also considered an additional performance metric. The metrics are defined below:

3.9.1 Precision

Precision is the metric that computes the number of positive class predictions that truly belong to the positive class. That is, how many True Positives are there among all positive class predictions? Thus, precision is the ratio of True Positives to the total number of positive predictions.

Formally, precision is defined as:

$$Precision = \frac{True\ Positive\ (TP)}{True\ Positive\ (TP) + FalsePositive\ (FP)}$$

Where:

True Positive (TP) is the number of positive samples accurately predicted as positive.

False Positive is the number of negative samples incorrectly predicted as positive.

Recall counts the number of positive class predictions made out of all positive examples in the dataset.

F-Measure provides a single score that balances both the concerns of precision and recall in one number.

3.9.2 Recall

Recall is the metric that computes the number of positive class predictions made out of all positive instances in the dataset. That is, how many True positive predictions are there out of all the actual positives? Hence, recall is the ratio of True Positives to the total number of instances that should have been classified as positive. The term recall is defined as:

$$Recall = \frac{True\ Positive\ (TP)}{True\ Positive\ (TP) + False\ Negative\ (FN)}$$

Where

True Positive (TP) is the number of positive samples accurately predicted as positive.

False Negative (FN) is the number of positive samples incorrectly predicted as negative.

3.9.3 Accuracy

Accuracy is the metric that computes the number of correct predictions made out of all number of predictions in the dataset. That is, how many correct predictions are out there out of all total predictions? Thus, accuracy is the ratio of the number of correct predictions to the total number of predictions. Mathematically, accuracy is defined as:

$$Accuracy = \frac{Correct\ Prediction}{All\ Prediction}$$

Where:

Correct Prediction is the number of samples correctly predicted.

All Prediction is the total number of all predictions in the dataset.

3.9.4 Receiver Operating Characteristics (ROC) Curve

The Receiver Operating Characteristics (ROC) curve is the evaluation metric that draws the graph showing the performance of the subsequent model. The curve plots two parameters True Positive Rate and False Positive Rate.

True Positive Rate is defined as:

$$True\ Positive\ Rate\ (TPR) = \frac{True\ Positive\ (TP)}{True\ Positive\ (TP) + False\ Negative\ (FN)}$$

Where:

True Positive (TP) is the number of positive samples accurately predicted as positive.

False Negative (FN) is the number of positive samples incorrectly predicted as negative.

False Positive Rate is defined as:

$$False\ Positive\ Rate\ (FPR) = \frac{False\ Positive\ (FP)}{False\ Positive\ (FP) + True\ Negative\ (TN)}$$

Where

False Positive (FP) is the number of negative samples incorrectly predicted as positive.

True Negative (TN) is the number of negative samples correctly predicted as negative.

The ROC curve shows the difference between True Positive Rate and False Positive Rate. The higher ROC value indicates a high True Positive Rate and low False Positive Rate, which is desirable in anomaly detection.

The models were trained and evaluated, observing the performance metrics in both training and testing subsets. The results of the developed models are presented and discussed in the next chapter.

3.10 Model Deployment Phase

The model deployment phase refers to the process of integrating a machine learning model into an existing production environment such as mobile applications, web applications or intrusion detection and prevention systems (IDS/IPS) to make real-world evaluations based on input data. The proposed model was deployed into SNORT (Chakrabarti *et al.,* 2010) to automatically detect and prevent username enumeration attacks.

The SNORT is an open-source software-based network detection and prevention system (IPS) that monitors the network traffic for unusual activities to determine whether it has been compromised. It contains one or more network-based sensors, monitoring and filtering all network traffic. When suspicious or malicious traffic is identified, the sensors assist in filtering network traffic and generating warnings. The SNORT IDS/IPS, in particular, assists in the detection and prevention of both external and internal attacks carried out by both attackers and benign users. It can be configured as a packet logger, sniffer or network intrusion detection and prevention system (IDS/IPS). If SNORT is configured as an intrusion prevention system (IPS), it monitors the network traffic and compares it against the defined rules. It issues alerts when it detects suspicious network traffic. All SNORT rules are defined and modified in *snort.conf* file.

CHAPTER FOUR

RESULTS AND DISCUSSION

4.1 Introduction

This Chapter presents the results of the study and discusses them in detail. The activities result in each specific objective were examined. Firstly, the evaluation of the data collection activity was presented. Then the performance metrics for the two-fold experimentations were conducted. The effectiveness comparison was examined when including and excluding ports information as feature sets. Lastly, the best performing model was adopted to the intrusion detection and prevention system.

4.2 Dataset

This work collected 18 844 and 17 429 instances for the SSH username enumeration attack and non-username enumeration attack, respectively, from the closed-environment network. For each class obtained, the dataset was split into a training subset and testing subset in a ratio of 80:20, respectively, as shown in Table 3. The training subset was used for models training while the testing subset evaluated the models developed. Table 9 illustrates the dataset distribution obtained for this study.

Table 9: Dataset distribution

Class	Instances
SSH Username Enumeration Attack	18 844
Non-Username Enumeration Attack	17 429

4.3 Performance Metrics Results

The performance metrics were evaluated for the two-fold experimentations conducted when including and excluding source and destination port as input features. Performance evaluation for each classification model developed in both cases used the same testing subset of the dataset shown in Table 3.

4.3.1 Precision

Table 10 presents precision values obtained in performance evaluation of the developed models in detecting and preventing username enumeration attacks. As indicated in the table, the precision values are for both cases, when including and excluding ports information as feature sets. The precision values of the KNN classifier is as higher as 99.95% when excluding ports information and 100% when including ports information, achieving the highest detection rate compared to other developed models. On the other hand, the performance of NB is relatively low, with the precision of 94.85% and 99.72%, respectively. It is likely because NB is a weak classifier in nature and other models outperformed it.

Table 10: Precision values obtained by different classifiers when including and excluding ports information

Classifier	Precision - Ports exclusive	Precision - Ports inclusive
DT	99.84	99.97
RF	99.87	99.89
NB	94.85	99.72
KNN	99.95	100

4.3.2 Accuracy

While evaluating the performance of the models developed on the dataset described in Table 3, the accuracy values for two-fold experimentations were recorded. Table 11 shows the accuracy values obtained by different classifiers when including and including source and destination ports as a feature set. The maximum accuracy was 99.93% when excluding ports information and 99.95% when including ports information. The minimum accuracy values obtained were 95.70% and 99.85% when excluding and including ports information. The accuracy values obtained imply that the models detect the attack with a good performance, as the values remained high for all models.

Table 11: Accuracy values obtained by different classifiers when including and excluding ports information

Classifier	Accuracy - Ports exclusive	Accuracy - Ports inclusive
DT	99.88	99.93
RF	99.92	99.94
NB	95.70	99.85
KNN	99.93	99.95

4.3.3 Receiver Operating Characteristic Curve

The area under receiver operating characteristic (ROC) curve summarized the performance of the developed models to detect and prevent username enumeration attacks. Table 12 indicates the roc curve values obtained by all classifiers used. As can be seen, the correctly classified rate is higher to the maximum value of 1, both when using and not using port information as a feature set. It implies that the classifiers can effectively detect username enumeration attacks with a high detection rate and low false alarm rate. Figure 19 and 20 show the roc curves for all modes when using and not using ports information.

Table 12: ROC values obtained by different classifiers when including and excluding ports information

Classifier	ROC - Ports exclusive	ROC - Ports inclusive
DT	0.997	0.998
RF	0.998	0.999
NB	0.994	0.997
KNN	0.999	1.000

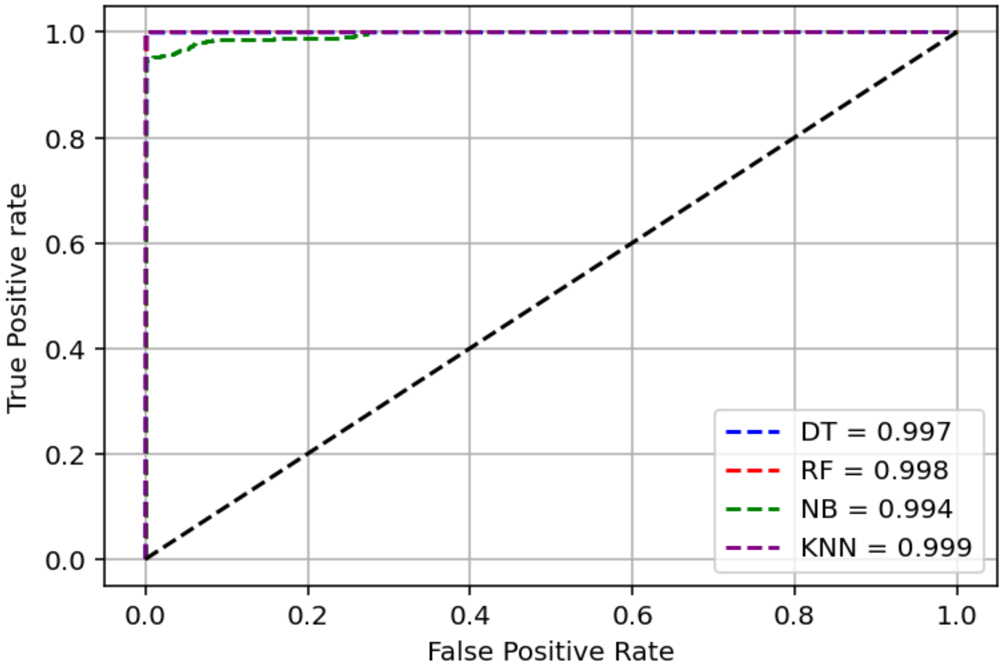


Figure 19: ROC AUC - Ports Exclusive

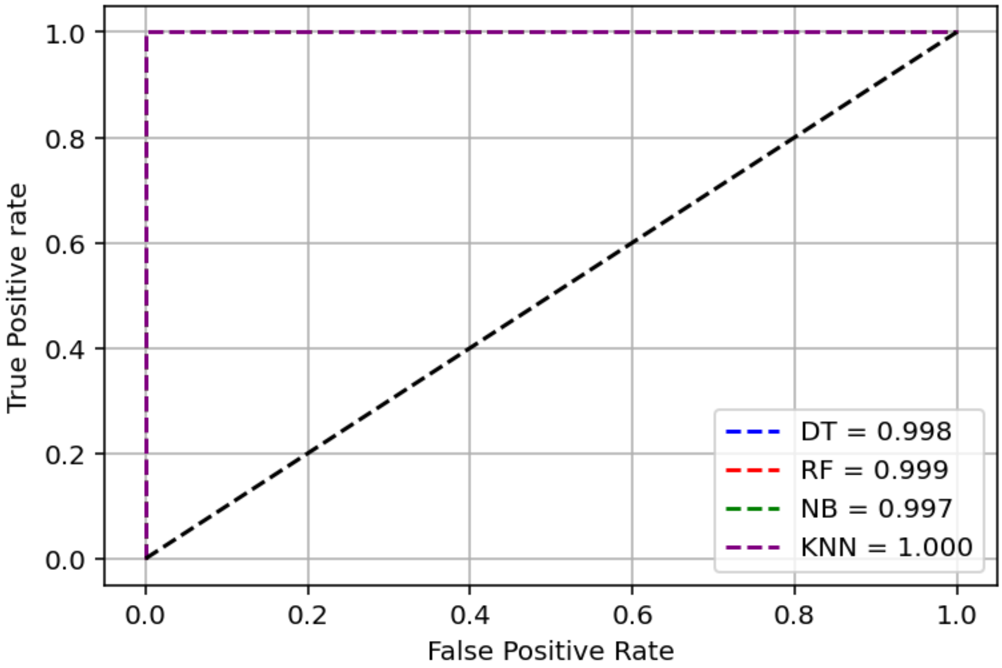


Figure 20: ROC AUC - Port Inclusive

Table 13: Summary of performance metrics for all models - Ports exclusive

Classifier	Precision	Accuracy	ROC
DT	99.84	99.88	0.997
RF	99.87	99.92	0.998
NB	94.85	95.70	0.994
KNN	99.95	99.93	0.999

Table 14: Summary of performance metrics for all model - Ports inclusive

Classifier	Precision	Accuracy	ROC
DT	99.97	99.93	0.998
RF	99.89	99.94	0.999
NB	99.72	99.85	0.997
KNN	100	99.95	1.000

If we observe our prediction results, we see all the classification models in both tables – when including and excluding ports information provide outstanding results as indicated by an accuracy of greater than 95.70%, which ensures the models effectiveness in the detection of username enumeration attack. The KNN classifier has the maximum performance metrics with an accuracy of 99.95% when including source and destination ports as input features and an accuracy of 99.93% while excluding source and destination ports as models input features.

Additionally, Fig. 19 and Fig. 20 show the ROC curves as the models' outcome results for two kinds of experiments conducted. They represent the True Positive rate versus False Positive rate of each classification model developed.

From the figures, we observe that the correctly classified rate is higher, close to the maximum value of 1. In contrast, the falsely classified rate is low for both cases – when including and excluding ports information. Therefore, from the outcome results in Table 13 and Table 14 together with ROC curves in Fig. 19 and 20, we can conclude that our machine-learning-based classification models effectively detect username enumeration attacks with high detection and low false alarm rate.

4.4 Effectiveness Comparison when Including and Excluding Ports Information

The effectiveness comparison between two kinds of experiments conducted shows that when including source and destination ports as input features, there are performance improvements compared to when the source and destination ports are excluded. Figure 21 and 22 show the relative comparison of precision, accuracy and roc-auc utilizing the dataset discussed in the earlier section. The classification performances of the DT, RF and KNN models slightly improve. The KNN model increases from an accuracy of 99.93% when it excludes source and destination ports as a feature set to an accuracy of 99.95% when it includes source and destination as a feature set. Similarly, the RF model slightly improves from an accuracy of 99.92% to 99.94% when including source and destination port as the model's input features. The Decision Tree improves its performance from an accuracy of 99.88% to 99.93%. Finally, the Naïve Bayes model significantly improves when including ports information as a feature set. It increases from an accuracy of 95.70% to 99.85%. Usually, Naïve Bayes is a weak classifier, and for the case of excluding ports information as input features in our study, other classifiers outperform it. However, by including source and destination port to its feature set, Naïve Bayes produces almost the same performance outcome results compared to DT, RF and KNN.

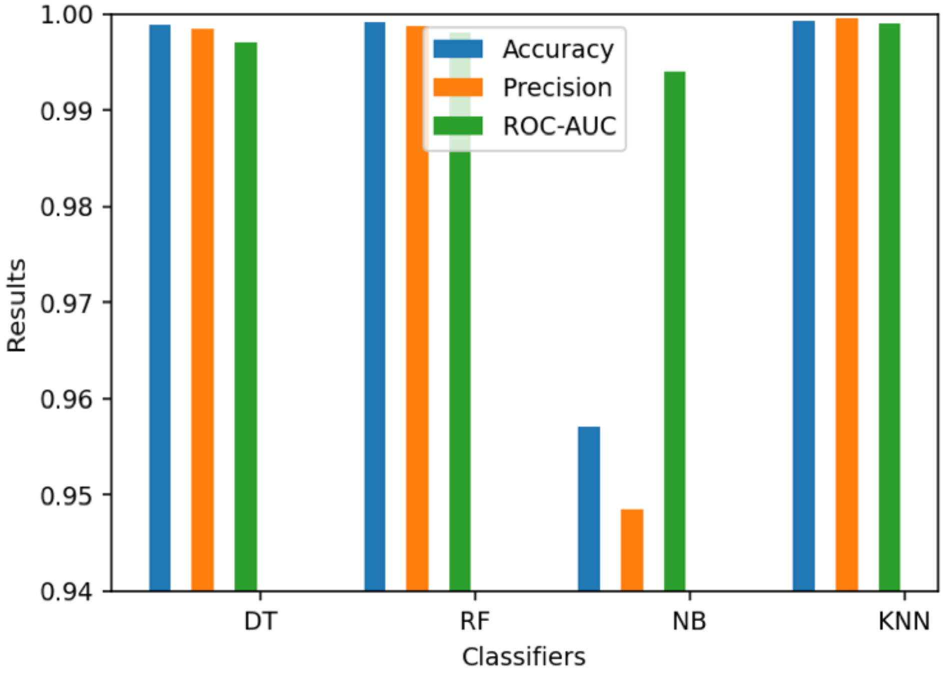


Figure 21: Effectiveness comparison – Ports exclusive

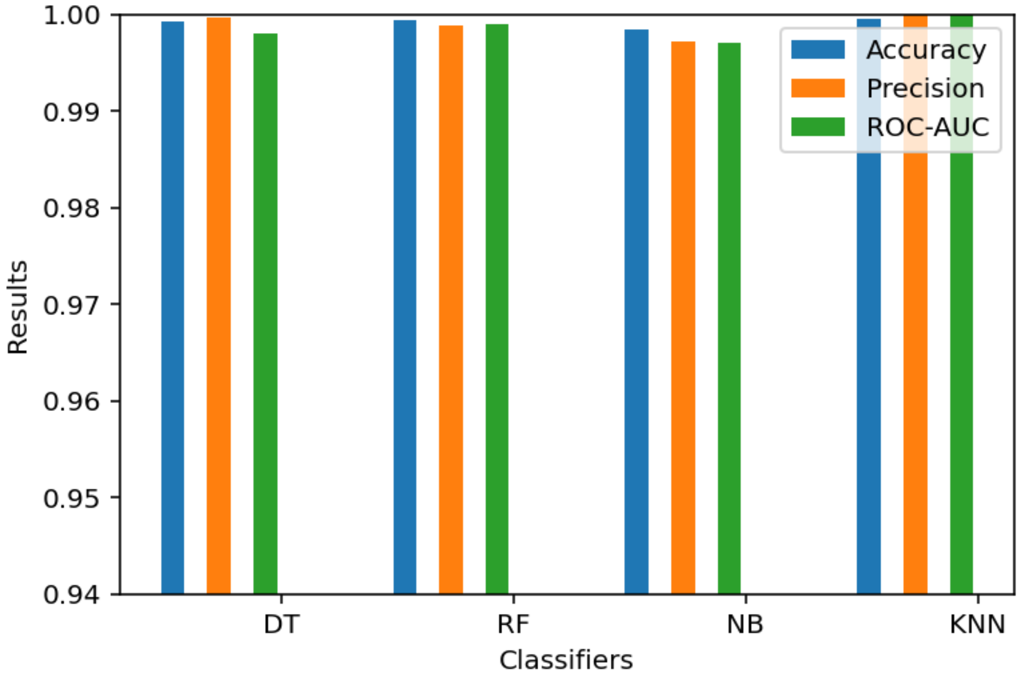


Figure 22: Effectiveness comparison – Ports inclusive

We observe that the DT, RF and KNN classification models produce almost the same classification performances regardless of whether port information is included or excluded in the feature set. It can be translated that even if source and destination ports are not included as the model's input features, the distribution of samples in the feature area is still a means that samples with a similar label are dispersed together.

We also observe that the Naïve Bayes classification model significantly enhances performance when including ports information as its input feature. This is due to the presumption that features in Naïve Bayes are completely independent. Therefore, it is rational to accept that the independent nature of Naïve Bayes' features can be recompensed with the inclusion of additional attributes to its attribute set and yields in performance improvement.

Thus, according to the results shown in Fig. 21 and 22 and the above experimental analysis, we can conclude that including source and destination ports as input features has various impacts on the developed classifiers depending on their type. However, it generally enhances the performances, ensuring the models' effectiveness in detecting username enumeration attacks.

4.5 Custom IDS/IPS

K-Nearest Neighbor model was chosen for the deployment because it has a simpler structure and requires fewer computing resources than the other training techniques used in this study.

The selected model was embedded into the SNORT intrusion detection and prevention system (IDS/IPS), as mentioned in Section 3.10. The proposed customized SNORT IDS/IPS was implemented by adding and editing KNN rules, specifically by modifying and fine-tuning */user/local/etc/* in *snort.conf.* After deploying the KNN model into SNORT IDS/IPS, its default action *"alert"* was changed to *drop* for dropping username enumeration attack traffic and allowing non-username enumeration attack traffic according to rules in customized SNORT IDS/IPS.

CHAPTER FIVE

CONCLUSION AND RECOMMENDATIONS

5.1 Conclusion

This study presents a novel username enumeration attack detection and prevention method on SSH protocol using machine-learning approaches. To achieve this, we collected the data from a closed-environment network, and the dataset is then labelled to generate a labelled dataset. We trained four distinct classifiers in a dataset containing two class labels: The username enumeration and non-username enumeration attack class instances. The former represented the normal class, while the latter represented the attack class. We evaluated the models' performance using accuracy, precision and ROC-AUC values. Our findings show that using machine-learning approaches to detect SSH username enumeration attacks achieves reasonable results, with KNN having an accuracy of 99.93%, NB 95%, RF 99.92% and DT 99.88%.

In addition, when training classification models, we investigated the impact of including ports information in the feature set. Our findings imply that including source and destination ports as input features improved performances without compromising computation power. However, the performance improvements vary from classifier to classifier based on their nature. The classifier such as Naïve Bayes, significantly enhances performance when including ports information. The nature of Naïve Bayes' features is completely independent; hence, including ports information yields significant performance improvements. The best selected model was then deployed into intrusion detection and prevention system (IDS/IPS) to automatically detect and prevent username enumeration attack

5.2 Recommendations

In future work, this research can be expanded into several directions. This study recommends a further stabilization of the developed models' robustness by gathering more data in a production-environment network and evaluating how developed models would perform on the real-world live dataset. Deep-learning techniques may also be incorporated in the future to detect username enumeration attacks. The study also recommends the use of machines with high computation power.

REFERENCE

Abubakar, A., & Pranggono, B. (2017). *Machine learning based intrusion detection system for software defined networks. The 2017 Seventh International Conference on Emerging Security Technologies (EST)*. https://www.google.com

Agghey, A. Z., Mwinuka, L. J., Pandhare, S. M., Dida, M. A., & Ndibwile, J. D. (2021). Detection of Username Enumeration Attack on SSH Protocol: Machine Learning Approach. *Symmetry,* 13(11), 2192.

Ahmad, T., & Aziz, M. N. (2019). Data preprocessing and feature selection for machine learning intrusion detection systems. *Express Letters*, *13*(2), 93–101.

Ahsan, M., Gomes, R., Chowdhury, M., & Nygard, K. E. (2021). Enhancing Machine Learning Prediction in Cybersecurity Using Dynamic Feature Selector. *Journal of Cybersecurity and Privacy*, 1(1), 199–218.

Alata, E., Nicomette, V., Kaâniche, M., Dacier, M., & Herrb, M. (2006). *Lessons learned from the deployment of a high-interaction honeypot. The 2006 Sixth European Dependable Computing Conference*. https://www.google.com

Aljuaid, T., & Sasi, S. (2016). *Proper imputation techniques for missing values in data sets. 2016 International Conference on Data Science and Engineering*. https://www.google.com

Alqahtani, H., Sarker, I. H., Kalim, A., Minhaz Hossain, S. M., & Hossain, S. (2020). *Cyber Intrusion Detection Using Machine Learning Classification Techniques*. https://www.google.com

Alshehri, H., & Meziane, F. (2017). Current state on internet growth and usage in Saudi Arabia and its ability to support e-commerce development. *Journal of Advanced Management Science*, *5*(2), 127–132.

Anandita, S., Rosmansyah, Y., Dabarsyah, B., & Choi, J. U. (2015). *Implementation of dendritic cell algorithm as an anomaly detection method for port scanning attack. The 2015 International Conference on Information Technology Systems and Innovation.* https://www.google.com

Apruzzese, G., Colajanni, M., Ferretti, L., Guido, A., & Marchetti, M. (2018). *On the* *effectiveness of machine and deep learning for cyber security. The 2018 10th International Conference on Cyber Conflict (CyCon)*. https://www.google.com

Bhagwat, R., & Kadwalkar, A. (2020). A Research on Secure Shell Protocol. *International Journal of Advanced Research in Science, Communication and Technology*, *9*(2), 1-6.

Bhatia, N. (2010). *Survey of nearest neighbor techniques. arXiv preprint arXiv:1007.0085.* https://www.google.com

Bhavani, T. T., Rao, M. K., & Reddy, A. M. (2020). *Network intrusion detection system using random forest and decision tree machine learning techniques. First International Conference on Sustainable Technologies for Computational Intelligence*. https://www.google.com

Bollegala, D. (2017). Dynamic feature scaling for online learning of binary classifiers. *Knowledge-Based Systems*, *129*, 97–105.

Buczak, A. L., & Guven, E. (2016). A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection. *Communications Surveys and Tutorials*, *18*(2), 1153–1176. https://doi.org/10.1109/COMST.2015.2494502

Čeleda, P., Velan, P., Král, B., & Kozák, O. (2019). *Enabling SSH Protocol Visibility in Flow Monitoring. The 2019 IFIP/IEEE Symposium on Integrated Network and Service Management.* https://www.google.com

Chakrabarti, S., Chakraborty, M., & Mukhopadhyay, I. (2010). *Study of snort-based IDS. Proceedings of the International Conference and Workshop on Emerging Trends in Technology.* https://www.google.com

Chandrasekar, P., Qian, K., Shahriar, H., & Bhattacharya, P. (2017). *Improving the prediction accuracy of decision tree mining with data preprocessing. The 2017 IEEE 41st Annual Computer Software and Applications Conference.* https://www.google.com

Charbuty, B., & Abdulazeez, A. (2021). Classification based on decision tree algorithm for machine learning. *Journal of Applied Science and Technology Trends*, *2*(01), 20–28.

Cherfi, A., Nouira, K., & Ferchichi, A. (2018). Very fast C4. 5 decision tree algorithm. *Applied Artificial Intelligence*, *32*(2), 119–137.

Dave, K. T. (2013). Brute-force Attack ‘Seeking but Distressing’. *International Journal of*

Innovations in Engineering and Technology Brute-force, 2(3), 75-78.

De Fuentes, J. M., Hernandez-Encinas, L., & Ribagorda, A. (2018). *Security Protocols for Networks and Internet: A Global Vision. In Computer and Network Security Essentials.* https://www.google.com

Dunford, R., Su, Q., & Tamang, E. (2014). *The pareto principle*. https://www.google.com

Eesa, A. S., Abdulazeez, A. M., & Orman, Z. (2017). A DIDS Based on The Combination of Cuttlefish Algorithm and Decision Tree. *Science Journal of University of Zakho*, 5(4), 313-318.

Eesa, A. S., Orman, Z., & Brifcani, A. M. A. (2015). A novel feature-selection approach based on the cuttlefish optimization algorithm for intrusion detection systems. *Expert systems with applications*, 42(5), 2670-2679.

Elmrabit, N., Zhou, F., Li, F., & Zhou, H. (2020). *Evaluation of machine learning algorithms for anomaly detection. The 2020 International Conference on Cyber Security and Protection of Digital Services (Cyber Security).* https://www.google.com

Exploits-db. (2018). *OpenSSH 2.3 < 7.7 - Username Enumeration*. https://www.exploit-db.com/exploits/45233

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J. T., Blum, M., & Hutter, F. (2019). *Auto-sklearn: Efficient and robust automated machine learning. In Automated Machine Learning*. https://www.google.com

Fiterău-Broştean, P., Lenaerts, T., Poll, E., De Ruiter, J., Vaandrager, F., & Verleg, P. (2017). *Model learning and model checking of SSH implementations. The 2017 - Proceedings of the 24th ACM SIGSOFT International SPIN Symposium on Model Checking of Software.* https://doi.org/10.1145/3092282.3092289

Gou, J., Ma, H., Ou, W., Zeng, S., Rao, Y., & Yang, H. (2019). A generalized mean distance-based k-nearest neighbor classifier. *Expert Systems with Applications*, *115*, 356–372.

Gupta, P., Mehrotra, S., Panwar, N., Sharma, S., Venkatasubramanian, N., & Wang, G. (2020). *Quest: Practical and oblivious mitigation strategies for COVID-19 using WiFi datasets. ArXiv Preprint ArXiv:2005.02510.* https://www.google.com

Hamid, Y., Sugumaran, M., & Journaux, L. (2016). Machine learning techniques for intrusion detection: A comparative analysis. *Proceedings of the International Conference on Informatics and Analytics*. https://www.google.com

Han, J., Pei, J., & Kamber, M. (2011). *Data mining: Concepts and techniques*. https://www.google.com

Hansman, S., & Hunt, R. (2005). A taxonomy of network and computer attacks. *Computers & Security*, *24*(1), 31–43.

Hewlett-Packard. (2010). *Top Cyber Security Risks Threat report for 2010.* http:// dvlabs. tippingpoint. com/toprisks2010.

Hofstede, R., Jonker, M., Sperotto, A., & Pras, A. (2017). Flow-based web application brute-force attack and compromise detection. *Journal of Network and Systems Management*, *25*(4), 735–758.

Hoque, N., Bhuyan, M. H., Baishya, R. C., Bhattacharyya, D. K., & Kalita, J. K. (2014). Network attacks: Taxonomy, tools and systems. *Journal of Network and Computer Applications*, *40*, 307–324.

Hossain, M. D., Ochiai, H., Doudou, F., & Kadobayashi, Y. (2020). SSH and FTP brute-force Attacks Detection in Computer Networks: LSTM and Machine Learning Approaches. *2020 5th International Conference on Computer and Communication Systems.* https://www.google.com

Huang, J., Li, Y. F., & Xie, M. (2015). An empirical analysis of data preprocessing for machine learning-based software cost estimation. *Information and Software Technology*, *67*, 108–127. https://doi.org/10.1016/j.infsof.2015.07.004

Hynek, K., Beneš, T., Čejka, T., & Kubátová, H. (2020). *Refined Detection of SSH Brute-Force Attackers Using Machine Learning. IFIP International Conference on ICT Systems Security and Privacy Protection*. https://www.google.com

Infante-Moro, A., Infante-Moro, J. C., Martínez-López, F. J., & García-Ordaz, M. (2016). The importance of internet and online social networks in the Spanish hotel sector. *Applied Computer Science,* 12(1), 57-86.

Internetworldstats. (2021). *World Internet Users Statistics and 2021 World Population Stats*. https://www.internetworldstats.com/stats.htm

Jain, G. (2021). *Application of SNORT and Wireshark in Network Traffic Analysis. IOP Conference Series: Materials Science and Engineering*. https://www.google.com

Jang-Jaccard, J., & Nepal, S. (2014). A survey of emerging threats in cybersecurity. *Journal of Computer and System Sciences*, *80*(5), 973–993.

Javed, M., & Paxson, V. (2013). Detecting stealthy, distributed SSH brute-forcing. *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*. https://www.google.com

John, G. H., & Langley, P. (2013). *Estimating Continuous Distributions in Bayesian Classifiers*. https://arxiv.org/abs/1302.4964v1

Jordan, M. I., & Mitchell, T. M. (2015). Machine learning: Trends, perspectives, and prospects. *Science*, *349*(6245), 255–260.

Joshi, A., Wazid, M., & Goudar, R. H. (2015). An efficient cryptographic scheme for text message protection against brute force and cryptanalytic attacks. *Procedia Computer Science*, *48*, 360–366.

Kannisto, J., & Harju, J. (2017). The time will tell on you: Exploring information leaks in ssh public key authentication. *International Conference on Network and System Security*. https://www.google.com

kaspersky. (2021). *Brute Force Attacks: Password Protection | Kaspersky*. https://www.kaspersky.com/resource-center/definitions/brute-force-attack

Kavitha, M. P. (2011). *Secured Password Hacking Process Using Multi Authentication Process*. https://www.google.com

Kaynar, K. (2016). A taxonomy for attack graph generation and usage in network security. *Journal of Information Security and Applications*, *29*, 27–56.

Khandait, P., Tiwari, N., & Hubballi, N. (2021). Who is Trying to Compromise Your SSH Server ? An Analysis of Authentication Logs and Detection of Bruteforce Attacks. *ACM International Conference Proceeding Series*. https://doi.org/10.1145/3427477.3429772

Kyaw, A. K., Sioquim, F., & Joseph, J. (2016). *Dictionary attack on Wordpress: Security and forensic analysis. The 2015 2nd International Conference on Information Security and Cyber Forensics, InfoSec 2015*. https://doi.org/10.1109/InfoSec.2015.7435522

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444.

Lee, J. K., Kim, S. J., Park, C. Y., Hong, T., & Chae, H. (2016). Heavy-tailed distribution of the SSH Brute-force attack duration in a multi-user environment. *Journal of the Korean Physical Society*, *69*(2), 253–258.

Li, P., & Qiu, X. (2012). NodeRank: an algorithm to assess state enumeration attack graphs. *2012 8th International Conference on Wireless Communications, Networking and Mobile Computing*. https://www.google.com

Li, X., Chen, W., Zhang, Q., & Wu, L. (2020). Building auto-encoder intrusion detection system based on random forest feature selection. *Computers & Security*, *95*, 101851.

Liang, J., Qin, Z., Xiao, S., Ou, L., & Lin, X. (2019). Efficient and secure decision tree classification for cloud-assisted online diagnosis services. *Transactions on Dependable and Secure Computing,* 18(4), 1632-1644.

Liu, Y., & Morgan, Y. (2018). Security against passive attacks on network coding system: A survey. *Computer Networks*, *138*, 57–76.

Mahesh, B. (2020). Machine learning algorithms: A review. *International Journal of Science and Research*, 9, 381-386.

Malhotra, S., Bali, V., & Paliwal, K. K. (2017). Genetic programming and K-nearest neighbour classifier based intrusion detection model. The *2017 7th International Conference on Cloud Computing, Data Science & Engineering-Confluence*. https://www.google.com

Mazraeh, S., Ghanavati, M., & Neysi, S. H. N. (2019). Intrusion detection system with decision tree and combine method algorithm. *International Academic Journal of Science and Engineering*, *6*(1), 167–177.

McGinnis, W. D., Siu, C., Andre, S., & Huang, H. (2018). Category encoders: A scikit-learn-contrib package of transformers for encoding categorical data. *Journal of Open Source Software*, *3*(21), 501.

Mehmood, T., & Rais, H. B. M. (2016). Machine learning algorithms in context of intrusion detection. The *2016 3rd International Conference on Computer and Information Sciences*. https://www.google.com

Nagamalai, D., Renault, E., & Dhanuskodi, M. (2011). *Trends in Computer Science,* *Engineering and Information Technology: First International Conference, CCSEIT 2011, Tirunelveli, Tamil Nadu, India, September 23-25, 2011, Proceedings*. https://www.google.com

Najafabadi, M. M., Khoshgoftaar, T. M., Calvert, C., & Kemp, C. (2015). *Detection of ssh brute force attacks using aggregated netflow data. The 2015 14th International Conference on Machine Learning and Applications.* https://www.google.com

Najafabadi, M. M., Khoshgoftaar, T. M., Kemp, C., Seliya, N., & Zuech, R. (2014). Machine learning for detecting brute force attacks at the network level. The *2014 International Conference on Bioinformatics and Bioengineering*. https://www.google.com

Nathan, A. J., & Scobell, A. (2020). 2020 Data Breach Investigations Report. *Verizon*. https://enterprise.verizon.com/resources/reports/2020-data-breach-investigations-report.pdf%0Ahttp://bfy.tw/HJvH

Nawir, M., Amir, A., Yaakob, N., & Lynn, O. B. (2019). Effective and efficient network anomaly detection system using machine learning algorithm. *Bulletin of Electrical Engineering and Informatics*, *8*(1), 46–51.

Ndibwile, J. D., Govardhan, A., Okada, K., & Kadobayashi, Y. (2015). Web server protection against application layer DDoS attacks using machine learning and traffic authentication. *Proceedings - International Computer Software and Applications Conference*, *3*, 261–267. https://doi.org/10.1109/COMPSAC.2015.240

OpenSSH. (2021). *OpenSSH* https://www.openssh.com/

Owens, J., & Matthews, J. (2008). A study of passwords and methods used in brute-force SSH attacks. *USENIX Workshop on Large-Scale Exploits and Emergent Threats*. https://www.google.com

Pahwa, K., & Agarwal, N. (2019). Stock market analysis using supervised machine learning. *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing.* https://www.google.com

Patil, A., Laturkar, A., Athawale, S. V., Takale, R., & Tathawade, P. (2017). A multilevel system to mitigate DDOS, brute force and SQL injection attack for cloud security. The *2017 International Conference on Information, Communication, Instrumentation and Control.* https://www.google.com

Patil, S., & Kulkarni, U. (2019). Accuracy prediction for distributed decision tree using machine learning approach. The *2019 3rd International Conference on Trends in Electronics and Informatics.* https://www.google.com

Pawar, M. V., & Anuradha, J. (2015). Network security and types of attacks in network. *Procedia Computer Science*, *48*, 503–506.

Portswigger, 2018. (n.d.). *Vulnerabilities in password-based login | Web Security Academy*. https://portswigger.net/web-security/authentication/password-based

Priyanka, & Kumar, D. (2020). Decision tree classifier: A detailed survey. *International Journal of Information and Decision Sciences,* 12(3), 246-269.

Rahmaninia, M., Moradi, P., & Jalili, M. (2020). A Multi-Objective Feature Selection Method based on the Conditional Mutual Information and Pareto Set Theory. *Tabriz Journal of Electrical Engineering*, *50*(3), 1225–1237.

Rapid7. (n.d.). *User Enumeration Explained: Techniques and Prevention Tips | Rapid7 Blog*. 2017. https://www.rapid7.com/blog/post/2017/06/15/about-user-enumeration/

Resende, P. A. A., & Drummond, A. C. (2018). A survey of random forest based methods for intrusion detection systems. *Computing Surveys (CSUR)*, *51*(3), 1–36.

Saito, S., Maruhashi, K., Takenaka, M., & Torii, S. (2016). Topase: Detection and prevention of brute force attacks with disciplined IPs from IDs logs. *Journal of Information Processing*, *24*(2), 217–226.

Satoh, A., Nakamura, Y., & Ikenaga, T. (2012). *SSH dictionary attack detection based on flow analysis. The 2012 12th International Symposium on Applications and the Internet*. https://www.google.com

Sharafaldin, I., Lashkari, A. H., & Ghorbani, A. A. (2018). *A detailed analysis of the cicids2017 data set. International Conference on Information Systems Security and Privacy*. https://www.google.com

Sharma, D., & Kumar, N. (2017). A review on machine learning algorithms, tasks and applications. *International Journal of Advanced Research in Computer Engineering & Technology*, *6*(10), 1548–1552.

Sharmin, S., Shoyaib, M., Ali, A. A., Khan, M. A. H., & Chae, O. (2019). Simultaneous feature selection and discretization based on mutual information. *Pattern Recognition*, *91*, 162–174.

Sheikh, A. F. (2020). CompTIA Security+ Certification Study Guide. *CompTIA Security+ Certification Study Guide*. https://doi.org/10.1007/978-1-4842-6234-4

Soofi, A. A., & Awan, A. (2017). Classification techniques in machine learning: Applications and issues. *Journal of Basic and Applied Sciences*, *13*, 459–465.

Srivastava, M. (2021). *An introduction to network security attacks. In Inventive Systems and Control (pp. 505-515).* Springer, Singapore. https://www.google.com

Stiawan, D. (2017). *Cyber-attack penetration test and vulnerability analysis*. https://www.google.com

Stiawan, D., Idris, M., Malik, R. F., Nurmaini, S., Alsharif, N., & Budiarto, R. (2019). Investigating brute force attack patterns in IoT network. *Journal of Electrical and Computer Engineering,* 2019, 1-14.

Stiawan, D., Sandra, S., Alzahrani, E., & Budiarto, R. (2017). Comparative analysis of K-Means method and Naïve Bayes method for brute force attack visualization. The *2017 2nd International Conference on Anti-Cyber Crimes.* https://www.google.com

Stratosphere IPS. (2019). *Malware Capture Facility Project: Normal Captures: Stratosphere IPS*. https://www.stratosphereips.org/datasets-normal

Techcesscyber. (2018). *Protecting Network Against Brute Force Password Attacks - TechCess*. https://www.techcesscyber.com/2018/09/protecting-network-against-brute-force-password-attacks/

Turner, P. R., Arildsen, T., & Kavanagh, K. (2018). *Applied Scientific Computing: With Python*. https://www.google.com

Virtue Security. (2021). *Username Enumeration*. https://www.google.com

Vizváry, M., & Vykopal, J. (2013). Flow-based detection of RDP brute-force attacks. *Proceedings of 7th International Conference on Security and Protection of Information.* https://www.google.com

Vykopal, J. (2011). A flow-level taxonomy and prevalence of brute force attacks. *International*

Conference on Advances in Computing and Communications. https://www.google.com

Vykopal, J., Plesnik, T., & Minarik, P. (2009). Network-based dictionary attack detection. The *2009 International Conference on Future Networks*. https://www.google.com

Wong, H. M., Chen, X., Tam, H. H., Lin, J., Zhang, S., Yan, S., Li, X., & Wong, K. C. (2021). Feature Selection and Feature Extraction: Highlights. *International Conference Proceeding Series*. https://doi.org/10.1145/3461598.3461606

Ylonen, T. (2019). *SSH key management challenges and requirements. The 2019 10th IFIP International Conference on New Technologies, Mobility and Security, NTMS 2019 - Proceedings and Workshop*. https://doi.org/10.1109/NTMS.2019.8763773

APPENDICES

Appendix 1: Data preprocessing

Import libraries

```
#Mount Google Drive with Google Colab

from google.colab import drive

drive.mount('/content/gdrive')

import numpy as np

np.random.seed(42)

import pandas as pd

#For visualization

import matplotlib.pyplot as plt

import seaborn as sns

%config InlineBackend.figure_format = 'retina'

#Metrics

from sklearn.metrics import accuracy_score

from sklearn.metrics import precision_score

from sklearn.metrics import recall_score

from sklearn.metrics import roc_auc_score

#DT Classifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.naive_bayes import GaussianNB

from sklearn.neighbors import KNeighborsClassifier

#Split data

from sklearn.model_selection import train_test_split
```

Check dataset encoding

```
import chardet

with open("../content/gdrive/MyDrive/research/data_unprocessed.csv",
'rb') as rawdata:

    result = chardet.detect(rawdata.read(10000))

check what the character encoding might be

print(result)
```

Load the dataset according to encoding obtained

```
raw_data =
pd.read_csv('/content/gdrive/MyDrive/research/data_unprocessed.csv',
encoding='latin-1')
```

Save the dataset to standard encoding UTF-8

```
raw_data.to_csv('/content/gdrive/MyDrive/research/raw_data.csv',
index=False)
```

Load the dataset with standard encoding UTF-8

```
raw_data = pd.read_csv('/content/gdrive/MyDrive/research/raw_data.csv'
```

Data preprocessing

```
#Drop unnecessary columns

raw_data = raw_data.drop(['Info', 'ArrivalTime',
'FrameLengthStoredIntoCaptureFile.1'], axis=1)
```

Columns with missing values

```
missing_values = [misval for misval in raw_data.columns

                  if raw_data[misval].isnull().any()]

print(missing_values)
```

Fill missing values in columns

```
#Filling missing values with zeros

raw_data['SourcePort'].fillna(0, inplace=True)
```

```
raw_data['DestinationPort'].fillna(0, inplace=True)

#Filling missing values with pre or post data entry

raw_data['Flags'].fillna('noflag', inplace=True)

raw_data = raw_data.fillna(method='bfill')
```

Categorical encoding

```
#First know all categorical columns in dataset

cat_cols = [ccol for ccol in raw_data.columns

            if raw_data[ccol].dtypes=='object']

print(cat_cols)

#Label Encoding

from sklearn.preprocessing import LabelEncoder

le=LabelEncoder()

for col in cat_cols:

    if col in proc_data.columns:

        i = proc_data.columns.get_loc(col)

        proc_data.iloc[:,i]          =          proc_data.apply(lambda
i:le.fit_transform(i.astype(str)), axis=0, result_type='expand')
```

Scaling

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

number_cols = [no for no in proc_data.columns

               if proc_data[no].dtypes in ['int64', 'float64']]

print(number_cols)

proc_data[['Time',    'SourceIP',    'SourcePort',    'DestinationIP',
'DestinationPort',      'Protocol',      'PacketLength',      'Delta',
'TimeShiftForPacket',   'FrameLengthStoredIntoCaptureFile',   'Flags',
'TCPFlags',       'TextItem',       'EpochTime',       'FrameNumber',
'FrameLengthOnTheWire', 'HeaderLength', 'TotalLength', 'TimeToLive',
'Label']] = scaler.fit_transform(proc_data[['Time', 'SourceIP',
'SourcePort', 'DestinationIP', 'DestinationPort', 'Protocol',
'PacketLength', 'Delta', 'TimeShiftForPacket',
'FrameLengthStoredIntoCaptureFile', 'Flags', 'TCPFlags', 'TextItem',
'EpochTime', 'FrameNumber', 'FrameLengthOnTheWire', 'HeaderLength',
'TotalLength', 'TimeToLive', 'Label']])

#Visualize

plt.figure(figsize=(50,25))

sns.scatterplot(data=proc_data)
```

Appendix 2: Optimization algorithm to select optimum hyperparameters Ports exclusive

Modeling

```
#Shuffle data

proc_data = proc_data.reindex(np.random.permutation(proc_data.index))
```

Models - Ports Exclusive

```
#Split target value

y = proc_data.Label

y.head()

#Split feature values

#Five Features: Time, PackeLength, Delta, Flags,TotalLength

X = proc_data[['Time','PacketLength', 'Delta', 'Flags', 'TotalLength']]

X.head()

#Split data

train_X, val_X, train_y, val_y = train_test_split(X, y, train_size=0.8,test_size=0.2, random_state=42)

#DT Classifier

#Optimization Algorithm to select optimum hyperparameters

#DT Hyperparameter Selection

from sklearn.model_selection import RandomizedSearchCV

#Assign parameters

criterion = ['gini','entropy']

#Maximum number of levels in tree

max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]

max_depth.append(None)

#Number of features to consider at every split
```

```
max_features = ['auto', 'sqrt']
#Leaf node
max_leaf_nodes = [int(x) for x in np.linspace (start=50,stop=1000,
num=20)]
splitter = ['best','random']
dt_param_grid = {'criterion':criterion,
                 'max_depth': max_depth,
                 'max_features': max_features,
                 'max_leaf_nodes': max_leaf_nodes,
                 'splitter': splitter}
dt_model = DecisionTreeClassifier()
dt_random = RandomizedSearchCV(estimator=dt_model,
param_distributions=dt_param_grid, n_iter=100, cv=3, verbose=2,
random_state=42, n_jobs=-1)
dt_random.best_params_
dt_model = DecisionTreeClassifier(random_state=42,
                                  criterion = 'gini',
                                  max_depth = 50,
                                  max_features = 'auto',
                                  max_leaf_nodes = 950,
                                  splitter = 'best')
#RF Classifier
#Optimization Algorithm for best hyperparameters selection
from sklearn.model_selection import RandomizedSearchCV
#Assign values to hyperparameters
# Number of trees in random forest
```

```
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000,
num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
#Minimum impurity decrease
min_impurity_decrease = [0.01, 0.1, 0.02, 0.2, 0.03, 0.3]
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'min_impurity_decrease': min_impurity_decrease,
               'bootstrap': bootstrap}
#Use the random grid to search for best hyperparameters
# First create the base model to tune
rf_model = RandomForestClassifier()
```

```
rf_random = RandomizedSearchCV(estimator = rf_model, param_distributions
= random_grid, n_iter = 100, cv = 3, verbose=2, random_state=42, n_jobs
= -1)

rf_random.best_params_

rf_model = RandomForestClassifier(random_state=42,
                                  bootstrap =True,
                                  max_depth = 90,
                                  max_features = 'auto',
                                  min_samples_leaf = 1,
                                  min_samples_split = 5,
                                  n_estimators = 1600)

#NB Classifier

#Optimization Algorithm for best hyperparameters selection

from sklearn.model_selection import RandomizedSearchCV

#Setting parameter distribution

nv_param_grid_ = {

    'var_smoothing': np.logspace(0,-9, num=100)

}

nv_model = GaussianNB()

nv_random = RandomizedSearchCV(estimator=nv_base_model,
param_distributions = nv_param_grid_, n_iter = 100, cv = 3, verbose=2,
random_state=42, n_jobs = -1)

#Check best parameters

nv_random.best_params_

nv_model = GaussianNB(var_smoothing =  2.848035868435799e-05)

#KNN Classifier

#Optimization Algorithm for best hyperparameters selection
```

```
from sklearn.model_selection import RandomizedSearchCV

#Creating parameters

leaf_size = list(range(1,50))

n_neighbors = list(range(1,30))

p=[1,2]

#Convert to dictionary

param_grid_nn  = dict(leaf_size=leaf_size, n_neighbors=n_neighbors, p=p)

nn_model = KNeighborsClassifier()

nn_random = RandomizedSearchCV(estimator=nn_model,
param_distributions=param_grid_nn, n_iter=100, cv=3, n_jobs=-1,
verbose=2, random_state=42)

nn_random.best_params_

nn_model = KNeighborsClassifier(leaf_size=7, n_neighbors =4, p = 1)
```

Appendix 3: Modeling - Ports exclusive

```
#Fit Models

dt_model.fit(train_X, train_y)

rf_model.fit(train_X, train_y)

nv_model.fit(train_X, train_y)

nn_model.fit(train_X, train_y)

#Prediction

dt_prediction = dt_model.predict(val_X)

rf_prediction = rf_model.predict(val_X)

nv_prediction = nv_model.predict(val_X)

nn_prediction = nn_model.predict(val_X)
```

Appendix 4: Evaluation - Ports exclusive

```
#Evaluation

print('DT Accuracy is:')

dt_accuracy = accuracy_score(val_y, dt_prediction)

print(dt_accuracy)

print('\t')

print('DT Precision is:')

dt_prec = precision_score(val_y, dt_prediction)

print(dt_prec)

print('\t')

print('RF Accuracy is:')

rf_accuracy = accuracy_score(val_y, rf_prediction)

print(rf_accuracy)

print('\t')

print('RF Precision is:')

rf_prec = precision_score(val_y, rf_prediction)

print(rf_prec)

print('\t')

print('NV Accuracy is:')

nv_accuracy = accuracy_score(val_y, nv_prediction)

print(nv_accuracy)

print('\t')



print('NV Precision is:')

nv_prec = precision_score(val_y, nv_prediction)
```

```
print(nv_prec)

print('\t')

print('KNN Accuracy is:')

nn_accuracy = accuracy_score(val_y, nn_prediction)

print(nn_accuracy)

print('\t')

print('KNN Precision is:')

nn_prec = precision_score(val_y, nn_prediction)

print(nn_prec)

print('\t')

#predict probabilities

pred_prob1 = dt_model.predict_proba(val_X)

pred_prob2 = rf_model.predict_proba(val_X)

pred_prob3 = nv_model.predict_proba(val_X)

pred_prob4 = nn_model.predict_proba(val_X)

# roc curve for models

from sklearn.metrics import roc_curve

fpr1, tpr1, thresh1 = roc_curve(val_y, pred_prob1[:,1], pos_label=1)

fpr2, tpr2, thresh2 = roc_curve(val_y, pred_prob2[:,1], pos_label=1)

fpr3, tpr3, thresh3 = roc_curve(val_y, pred_prob3[:,1], pos_label=1)

fpr4, tpr4, thresh4 = roc_curve(val_y, pred_prob4[:,1], pos_label=1)

#roc curve for tpr = fpr

random_probs = [0 for i in range(len(val_y))]

p_fpr, p_tpr, _ = roc_curve(val_y, random_probs, pos_label=1)

from sklearn.metrics import roc_auc_score
```

```
#auc scores

auc_score1 = roc_auc_score(val_y, pred_prob1[:,1])

auc_score2 = roc_auc_score(val_y, pred_prob2[:,1])

auc_score3 = roc_auc_score(val_y, pred_prob3[:,1])

auc_score4 = roc_auc_score(val_y, pred_prob4[:,1])

print('ROC for DT:')

print(auc_score1)

print('\t')

print('ROC for RF:')

print(auc_score2)

print('\t')

print('ROC for NB:')

print(auc_score3)

print('\t')

print('ROC for KNN:')

print(auc_score4)

print('\t')

print(auc_score1, auc_score2)

# matplotlib

import matplotlib.pyplot as plt

%config InlineBackend.figure_format = 'retina'

#.style.use('seaborn')

plt.grid()

# plot roc curves

plt.plot(fpr1, tpr1, linestyle='--',color='blue', label='DT = 0.997')
plt.plot(fpr2, tpr2, linestyle='--',color='red', label='RF = 0.998')

plt.plot(fpr3, tpr3, linestyle='--',color='green', label='NB = 0.994')

plt.plot(fpr4, tpr4, linestyle='--',color='purple', label='KNN = 0.999')

plt.plot(p_fpr, p_tpr, linestyle='--', color='black')

# title

plt.title('ROC curve')

# x label

plt.yticks ([0.94, 0.96,0.98,1.0])

plt.xticks([0.94, 0.96,0.98,1.0])

plt.xlabel('False Positive Rate')

# y label

plt.ylabel('True Positive Rate')

plt.legend(loc='best')

plt.savefig('ROC',dpi=500)

plt.show()
```

Models – Ports Inclusive

```
#Split target value

y = proc_data.Label

y.head()

#Split feature values

#Seven Features: Time, PackeLength, Delta, Flags,TotalLength

X     =     proc_data[['Time','PacketLength',     'Delta',     'Flags',
'TotalLength''SourcePort', 'DestinationPort']]

X.head()

#Split data
```

```
train_X,    val_X,    train_y,    val_y    =    train_test_split(X,    y,
train_size=0.8,test_size=0.2, random_state=42)

#DT Classifier

#Optimization Algorithm to select optimum hyperparameters
```

Appendix 5: Optimization Algorithm to select optimum hyperparameters Ports inclusive

```
#DT Hyperparameter Selection

from sklearn.model_selection import RandomizedSearchCV

#Assign parameters

criterion = ['gini','entropy']

#Maximum number of levels in tree

max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]

max_depth.append(None)


#Number of features to consider at every split

max_features = ['auto', 'sqrt']

#Leaf node

max_leaf_nodes = [int(x) for x in np.linspace (start=50,stop=1000, num=20)]

splitter = ['best','random']

dt_param_grid = {'criterion':criterion,

                 'max_depth': max_depth,

                 'max_features': max_features,

                 'max_leaf_nodes': max_leaf_nodes,

                 'splitter': splitter}

dt_model = DecisionTreeClassifier()

dt_random = RandomizedSearchCV(estimator=dt_model, param_distributions=dt_param_grid, n_iter=100, cv=3, verbose=2, random_state=42, n_jobs=-1)

dt_random.best_params_

dt_model = DecisionTreeClassifier(random_state=42,
                                  criterion = 'entropy',
                                  max_depth = 60,
                                  max_features = 'auto',
                                  max_leaf_nodes = 500,
                                  splitter = 'best')
#RF Classifier
#Optimization Algorithm for best hyperparameters selection
from sklearn.model_selection import RandomizedSearchCV
#Assign values to hyperparameters
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000,
num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
#Minimum impurity decrease
min_impurity_decrease = [0.01, 0.1, 0.02, 0.2, 0.03, 0.3]
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,

               'max_depth': max_depth,

               'min_samples_split': min_samples_split,

               'min_samples_leaf': min_samples_leaf,

               'min_impurity_decrease': min_impurity_decrease,

               'bootstrap': bootstrap}

#Use the random grid to search for best hyperparameters

# First create the base model to tune

rf_model = RandomForestClassifier()

rf_random = RandomizedSearchCV(estimator = rf_model, param_distributions
= random_grid, n_iter = 100, cv = 3, verbose=2, random_state=42, n_jobs
= -1)

rf_random.best_params_

rf_model = RandomForestClassifier(random_state=42,

                                  bootstrap =True,

                                  max_depth = 30,

                                  max_features = 'sqrt',

                                  min_samples_leaf = 1,

                                  min_samples_split = 5,

                                  n_estimators = 400)

#NB Classifier

#Optimization Algorithm for best hyperparameters selection

from sklearn.model_selection import RandomizedSearchCV

#Setting parameter distribution

nv_param_grid_ = {

    'var_smoothing': np.logspace(0,-9, num=100)
```

```
}
nv_model = GaussianNB()
nv_random = RandomizedSearchCV(estimator=nv_base_model,
param_distributions = nv_param_grid_, n_iter = 100, cv = 3, verbose=2,
random_state=42, n_jobs = -1)
#Check best parameters
nv_random.best_params_
nv_model = GaussianNB(var_smoothing =  0.022328467394420651)
#KNN Classifier
#Optimization Algorithm for best hyperparameters selection
from sklearn.model_selection import RandomizedSearchCV
#Creating parameters
leaf_size = list(range(1,50))
n_neighbors = list(range(1,30))
p=[1,2]
#Convert to dictionary
param_grid_nn  = dict(leaf_size=leaf_size, n_neighbors=n_neighbors, p=p)
nn_model = KNeighborsClassifier()
nn_random = RandomizedSearchCV(estimator=nn_model,
param_distributions=param_grid_nn, n_iter=100, cv=3, n_jobs=-1,
verbose=2, random_state=42)
nn_random.best_params_
nn_model = KNeighborsClassifier(leaf_size=8, n_neighbors =5, p = 2)
```

Appendix 6: Modeling - Ports inclusive

```
#Fit Models

dt_model.fit(train_X, train_y)

rf_model.fit(train_X, train_y)

nv_model.fit(train_X, train_y)

nn_model.fit(train_X, train_y)
```

Appendix 7: Evaluation - Ports inclusive

```
#Prediction

dt_prediction = dt_model.predict(val_X)

rf_prediction = rf_model.predict(val_X)

nv_prediction = nv_model.predict(val_X)

nn_prediction = nn_model.predict(val_X)

#Evaluation

print('DT Accuracy is:')

dt_accuracy = accuracy_score(val_y, dt_prediction)

print(dt_accuracy)

print('\t')

print('DT Precision is:')

dt_prec = precision_score(val_y, dt_prediction)

print(dt_prec)

print('\t')

print('RF Accuracy is:')

rf_accuracy = accuracy_score(val_y, rf_prediction)

print(rf_accuracy)

print('\t')


print('RF Precision is:')

rf_prec = precision_score(val_y, rf_prediction)

print(rf_prec)

print('\t')

print('NV Accuracy is:')
```

```
nv_accuracy = accuracy_score(val_y, nv_prediction)

print(nv_accuracy)

print('\t')

print('NV Precision is:')

nv_prec = precision_score(val_y, nv_prediction)

print(nv_prec)

print('\t')

print('KNN Accuracy is:')

nn_accuracy = accuracy_score(val_y, nn_prediction)

print(nn_accuracy)

print('\t')

print('KNN Precision is:')

nn_prec = precision_score(val_y, nn_prediction)

print(nn_prec)

print('\t')

#predict probabilities

pred_prob1 = dt_model.predict_proba(val_X)

pred_prob2 = rf_model.predict_proba(val_X)

pred_prob3 = nv_model.predict_proba(val_X)

pred_prob4 = nn_model.predict_proba(val_X)

# roc curve for models

from sklearn.metrics import roc_curve

fpr1, tpr1, thresh1 = roc_curve(val_y, pred_prob1[:,1], pos_label=1)

fpr2, tpr2, thresh2 = roc_curve(val_y, pred_prob2[:,1], pos_label=1)

fpr3, tpr3, thresh3 = roc_curve(val_y, pred_prob3[:,1], pos_label=1)
```

```
fpr4, tpr4, thresh4 = roc_curve(val_y, pred_prob4[:,1], pos_label=1)
#roc curve for tpr = fpr
random_probs = [0 for i in range(len(val_y))]
p_fpr, p_tpr, _ = roc_curve(val_y, random_probs, pos_label=1)
from sklearn.metrics import roc_auc_score
#auc scores
auc_score1 = roc_auc_score(val_y, pred_prob1[:,1])
auc_score2 = roc_auc_score(val_y, pred_prob2[:,1])
auc_score3 = roc_auc_score(val_y, pred_prob3[:,1])
auc_score4 = roc_auc_score(val_y, pred_prob4[:,1])
print('ROC for DT:')
print(auc_score1)
print('\t')
print('ROC for RF:')
print(auc_score2)
print('\t')
print('ROC for NB:')
print(auc_score3)
print('\t')
print('ROC for KNN:')
print(auc_score4)
print('\t')
print(auc_score1, auc_score2)
# matplotlib
import matplotlib.pyplot as plt
```

```
%config InlineBackend.figure_format = 'retina'

style.use('seaborn')

plt.grid()

# plot roc curves

plt.plot(fpr1, tpr1, linestyle='--',color='blue', label='DT = 0.998')

plt.plot(fpr2, tpr2, linestyle='--',color='red', label='RF = 0.999')

plt.plot(fpr3, tpr3, linestyle='--',color='green', label='NB = 0.997')

plt.plot(fpr4, tpr4, linestyle='--',color='purple', label='KNN = 1.000')

plt.plot(p_fpr, p_tpr, linestyle='--', color='black')

# title

plt.title('ROC curve')

# x label

plt.yticks ([0.94, 0.96,0.98,1.0])

plt.xticks([0.94, 0.96,0.98,1.0])

plt.xlabel('False Positive Rate')

# y label

plt.ylabel('True Positive Rate')

plt.legend(loc='best')

plt.savefig('ROC',dpi=500)

plt.show()
```

RESEARCH OUTPUTS

Research output 1: Publications

Agghey, A. Z., Mwinuka, L. J., Pandhare, S. M., Dida, M. A., & Ndibwile, J. D. (2021). Detection of Username Enumeration Attack on SSH Protocol: Machine Learning Approach. *Symmetry*, *13*(11), 2192.

Research output 2: Poster Presentation