ANOMALY-BASED INTRUSION DETECTION USING MACHINE LEARNING: A CASE STUDY ON PROBING ATTACKS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF INFORMATICS OF
THE MIDDLE EAST TECHNICAL UNIVERSITY
BY

EMRAH TUFAN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN
THE DEPARTMENT OF CYBER SECURITY

JULY 2020

ANOMALY-BASED INTRUSION DETECTION IN AN INSTITUTIONAL NETWORK USING MACHINE LEARNING: A CASE STUDY ON PROBING ATTACKS

Submitted by EMRAH TUFAN in partial fulfillment of the requirements for the degree of **Master of Science in Cyber Security Department, Middle East Technical University** by,

Prof. Dr. Deniz Zeyrek Bozşahin _____
Dean, Graduate School of Informatics

Assoc. Prof. Dr. Aysu Betin Can _____
Head of Department, Information Systems

Assoc. Prof. Dr. Cengiz Acartürk _____
Supervisor, Cognitive Science Dept., METU

Asst. Prof. Dr. Cihangir Tezcan _____
Co-Supervisor, Cyber Security Dept., METU

Examining Committee Members:

Prof. Dr. Ali Aydın Selçuk _____
Computer Engineering Dept., TOBB ETÜ

Assoc. Prof. Dr. Cengiz Acartürk _____
Cognitive Science Dept., METU

Asst. Prof. Dr. Cihangir Tezcan _____
Cyber Security Dept., METU

Asst. Prof. Dr. Aybar Can Acar _____
Health Informatics Dept., METU

Assoc. Prof. Dr. Banu Günel Kılıç _____
Information Systems Dept., METU

Date: **28 July 2020**

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name : EMRAH TUFAN

Signature : _____

ABSTRACT

ANOMALY-BASED INTRUSION DETECTION IN AN INSTITUTIONAL NETWORK USING MACHINE LEARNING: A CASE STUDY ON PROBING ATTACKS

TUFAN, EMRAH

MSc., Department of Cyber Security
Supervisor: Assoc. Prof. Dr. Cengiz ACARTÜRK
Co-Supervisor: Asst. Prof. Dr. Cihangir TEZCAN

July 2020, 80 pages

Cyber attacks constitute a serious threat to organizations with implications ranging from economic, reputational and legal consequences. As the techniques employed by cyber criminals get more sophisticated, information security professionals face a greater challenge protecting the famous triangle of confidentiality, integrity and availability (CIA). In today's interconnected realm of computer systems, every attack vector has a network dimension. Therefore, this study aims to detect network intrusion attempts with an anomaly-based machine learning model to provide better protection than the conventional misuse-based models. Two different models were built and implemented on a data set gathered from a production environment, ensemble learning and convolutional neural network respectively. To demonstrate the models' reliability and validity, they were applied on UNSW-NB15 benchmarking data set as well. To keep the scope of the study manageable, probing type of attack was focused on and models were trained accordingly. The results suggested that both models detect the chosen type of intrusion attempts with an F1 score of more than 0.97. Convolutional neural network model scored slightly higher with 0.99. Similar results were obtained in UNSW-NB15 data set pointing the proposed model's validity.

Keywords: Intrusion Detection Systems, Anomaly-Based, Machine Learning

ÖZ

MAKİNA ÖĞRENMESİ İLE KURUMSAL BİR AĞDA ANOMALİ TABANLI SİBER İHLAL TESPİT SİSTEMİ: KEŞİF SALDIRILARI ÜZERİNDE BİR VAKA ÇALIŞMASI

TUFAN, EMRAH
Yüksek Lisans, Siber Güvenlik Bölümü
Tez Yöneticisi: Doç. Dr. Cengiz ACARTÜRK
Yardımcı Tez Yöneticisi: Dr. Öğr. Üyesi Cihangir TEZCAN

Temmuz 2020, 80 sayfa

Siber saldırılar ekonomik, itibari ve yasal açılardan sonuçlar doğurarak organizasyonlara yaşamsal bir tehdit oluşturmaktadır. Siber suçlular tarafından kullanılan teknikler geliştikçe, bilgi güvenliği uzmanları popüler gizlilik, bütünlük ve erişilebilirlik üçgeninin bileşenlerini sağlamakta daha çok zorluk yaşamaktadır. Günümüzün birbirine bağlı bilgisayar dünyasında, her saldırının ağ ile ilgili bir boyutu bulunmaktadır. Bu yüzden bu çalışma, siber ihlal girişimlerini anomali tabanlı bir makina öğrenmesi modeli kullanarak geleneksel kural tabanlı metotlardan daha başarılı bir şekilde tespit etmeyi ve daha iyi koruma sağlamayı hedeflemektedir. Gerçek ortam sistemleri üzerinden toplanan veri üzerinde iki farklı model oluşturulmuş olup, bunlar sırasıyla topluluk öğrenmesi ve evrişimli sinirsel ağ modelleridir. Oluşturulan modellerin güvenirlirlik ve geçerliliğinin gösterilmesi amacıyla bu modeller aynı zamanda bir kıyaslama veri seti olan UNSW-NB15 veri seti üzerinde uygulanmıştır. Çalışmanın kapsamını aşmaması maksadıyla sadece keşif saldırıları üzerine yoğunlaşılmış ve modeller bunları tespit edecek şekilde eğitilmiştir. Sonuçlar göstermiştir ki; her iki model de seçilen siber ihlal tipini 0.97’den fazla bir F1 skoru ile doğru tahmin edebilmiştir. Evrişimli sinirsel ağ modeline ait sonuçlar bir miktar daha iyi olup, 0.99 seviyesindedir. UNSW-NB15 veri seti üzerinde alınan benzer sonuçlar oluşturulan modelin geçerliliğini göstermektedir.

Anahtar Sözcükler: Saldırı Tespit Sistemi, Anomali-Tabanlı, Makine Öğrenmesi.

ACKNOWLEDGMENTS

First of all, I would like to thank the academic personnel in the Department of Cyber Security who have contributed to my education. My academic and thesis advisor Assoc. Prof. Dr. Cengiz ACARTURK and thesis co-advisor Asst. Prof. Dr. Cihangir TEZCAN have been great supervisors with their fast correspondence and invaluable comments on my drafts.

I owe a special gratitude to Col.Adnan GURBUZ for his provision of visionary insights. During the master education, he supported and praised the work that have been done to encourage new achievements.

Another appreciation is needed for The Scientific and Technological Research Council of Turkey (TUBITAK) for their support on this work with 2210-C postgraduate scholarship.

And the last but never least, I have been always grateful to my family, my wife Burcu and son Ege because of their spiritual support and love. The thesis writing process has been no exception in that sense.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ACCS	Australian Center for Cyber Security
AUC	Area Under the Curve
CIA	Confidentiality Integrity Availability
CIC	Canadian Institute for Cybersecurity
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CVE	Common Vulnerabilities and Exposures
DARPA	Defense Advanced Research Projects Agency
DoS	Denial of Service
ENISA	European Agency for Network and Information Security
FTP	File Transfer Protocol
GPU	Graphical Processor Unit
GUI	Graphical User Interface
HIDS	Host-based Intrusion Detection System
HTTP	Hyper Text Transfer Protocol
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
IoT	Internet of Things
IP	Internet Protocol
IPS	Intrusion Prevention System
KDD	Knowledge Discovery and Data Mining
KNN	K Nearest Neighbours
LAN	Local Area Network
ML	Machine Learning
MLP	Multi Layer Perceptron
NAT	Network Address Translation
NIDS	Network-based Intrusion Detection System
ROC	Receiver Operating Characteristics
SNMP	Simple Network Management Protocol
SQL	Structured Query Language
SVM	Support Vector Machines
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UNSW	University of New South Wales
VANET	Vehicular Ad-hoc Network
WAN	Wide Area Network

CHAPTER 1

INTRODUCTION

1.1. Motivation

Humans are more dependent on computer-based technologies than ever, and this trend rapidly grows with time (Bowker, 2019; Hurwitz, 2018). Moreover, one characteristic that makes computer science such a game changer technology is computers' ability to inter-communicate in a network. It has been five decades since the first message of "lo" between two computers was sent. In the 1990s, Wi-Fi was invented and the idea of connecting personal computers without cables started becoming widespread for end-users. Today, with the introduction of internet of things (IoT), wearable technologies, vehicular ad-hoc network (VANET) and other cutting edge networking technologies, interconnected computers have become embedded in our daily life. On the other hand, the dilemma of availability versus security introduces risks to those who are responsible for protecting information assets or a critical infrastructure based on computer networks. Criminal hackers target governmental or non-governmental organizations with the intention of causing financial, political, physical and psychological damage (Goodrich & Tamassia, 2011; Stallings & Brown, 2017). This threat is immense recently. For example, Cisco announced their prevention of six trillion threats in 2018 which means more than 23,000 threats per second (Robbins, 2019). Reported global financial damage caused by cyber attacks cost 3 trillion dollars in 2015 and this amount is expected to be doubled in 2021 (Morgan, 2019).

European Agency for Network and Information Security (ENISA) announced top fifteen attack types in 2018 in comparison with 2017 as illustrated in Table 1 (Sfakianakis, Douligeris & Marinos, 2019). All categories more or less involve network component as a part of attack vector. More interestingly, ENISA's report emphasized the popularity of critical infrastructure as targets among attackers. In such a risky environment for businesses and organizations, detecting and preventing cyber attacks is more critical than ever.

This study will propose a model for detecting attacks against an institutional network using machine learning. Every organization needs a robust intrusion detection/prevention mechanism to guarantee the existence of fundamental information security triangle's components. Namely; confidentiality, integrity and availability (CIA). As it could be seen from previous statistics, cyber threat landscape is so complicated that information assets can no longer be protected solely with deterministic methods such as creating a rule for every possible intrusion attempt type. On the contrary, learning models are needed to defend against continuously evolving threats.

Table 1 ENISA 2018 cyber threat landscape report.

Top Threats 2017	Top Threats 2018	Change in ranking
1.Malware	1.Malware	
2.Web Based Attacks	2.Web Based Attacks	
3.Web Application Attacks	3.Web Application Attacks	
4.Phishing	4.Phishing	
5.Spam	5.Denial of Service	
6.Denial of Service	6.Spam	
7.Ransomware	7.Botnets	
8.Botnets	8.Data Breaches	
9.Insider Threat	9.Insider Threat	
10.Physical manipulation/damage	10.Physical manipulation/damage	
11.Data Breaches	11.Information Leakage	
12.Identity Theft	12.Identity Theft	
13.Information Leakage	13.Cryptojacking	NEW
14.Exploit Kits	14.Ransomware	
15.Cyber Espionage	15.Cyber Espionage	

1.2. Research Question

Rule based intrusion detection systems are fast and can operate on low resources. This feature makes rule-based detection tempting in an environment with thousands of sessions per second in a busy network. Therefore, ML models should introduce a significant improvement to compete with deterministic methods. In this research, the main question is that by employing a ML model, is it possible to enhance detection and reduce false detection rates significantly in an institutional network?

1.3. Scope of the Study

The terms of intrusion and intrusion detection needs to be explained in information security context to comprehend intrusion detection systems (IDS) better. The former indicates the unauthorized attempt to access an information asset on which the requesting agent (client) is not allowed. Access ranges from viewing to manipulation (even destruction) of the asset in this context. For instance, a user with a stolen password of another user can login to a system in which (s)he does not have access with his/her own user. Moreover, a legitimate user might try to consume a service in which (s)he is not warranted with his/her own privileges. Furthermore, another user might seek a way to place and execute a script on a remote computer to take control of it. The latter is the action or mechanism which aims to identify such attempts against computer systems. Intrusion detection was resembled to sounding of a

burglar alarm such that whenever a suspicious activity occurs, it goes off to warn authorities about it (Axelsson, 2000).

Deriving from the points above, intrusion detection systems are structural entities that aim to detect threats by looking at some symptoms either predetermined or learned. IDS do that in a methodological manner and became a component of the information security ecosystem. Rather than a gatekeeper rejecting attempts that do not suit an organization's criteria, IDS might be resembled to an intelligence agent who inspects the target by gathering information and reports his/her compliance status to the authorities. IDS have various types for accomplishing this task and a comprehensive taxonomy of IDS will be provided in the next chapter.

IDS, as the other information security components, exist to provide confidentiality, integrity and availability. By detecting the various types of network-based attacks, they may contribute the presence to all these three components. For instance, if an IDS detects a denial of service or ransomware attack, it provides availability. Moreover, if it spots a data breach or malware injection attempt, depending on the intruder's intention, confidentiality or integrity is assured. Deriving from these points, this study has the intention of building an anomaly-based IDS model to detect attacks on an institutional network.

To achieve this, data will be collected from a production environment. Then, features will be extracted from raw data using some tools and custom scripts. After the completion of preprocessing stage and creating a data frame with combined features, they will be eliminated to find the most contributing ones to the model. Filter and wrapper methods will be combined while making feature selection. After that, two different supervised ML models will be built. First, ensemble learning methods and convolutional neural network (CNN). After harvesting results from the two models, they will be compared with each other to find out which one is more successful. And finally, the same pipeline will be implemented on UNSW-NB15 data set to validate results found on institutional data set.

Rest of the thesis will be as follows. In chapter two, a brief literature review will be included with a deductive strategy. First, intrusion detection systems in general, then machine learning methods for anomaly-based IDS will be discussed. Chapter three constitutes the most ambitious part of the study. It first provides a taxonomy for IDS data sets, then selected data set for the research will be discussed and justified. After that, proposed methodology will be argued in details. Chapter four will be devoted for explaining the performance evaluation criteria and revealing the results. Finally, chapter five and six will provide a discussion and conclusion part in which how well did the study achieved in answering the research question will be evaluated.

CHAPTER 2

RELEVANT WORK

In the field of cyber security, intrusion detection constitutes a well-recognized problem and different approaches have been taken to overcome it. Artificial intelligence is yet another tool which proposes a solution to this issue with its subdomains such as machine learning and deep learning. This chapter provides a quick review on intrusion detection systems and machine learning models in relation with them.

2.1. Intrusion Detection Systems

Intrusion attempts constitute one of the major threats in network security. Having the ultimate goal of getting full or at least partial control of the targeted system, hackers might try various tactics and tricks. In contrast, information security professionals undertake the responsibility to protect digital assets of the enterprise. Defending side also has its arsenal of network/host security applications such as intrusion detection/prevention system (IDS/IPS), firewall, data loss prevention and anti-malware software. Traditionally, IDS and firewalls either standalone or integrated, form the outer perimeter, in other words the border wall of the protected area. Figure 1 demonstrates a simplified example of an enterprise network (Crichigno, Bou-Harb, & Ghani, 2019).

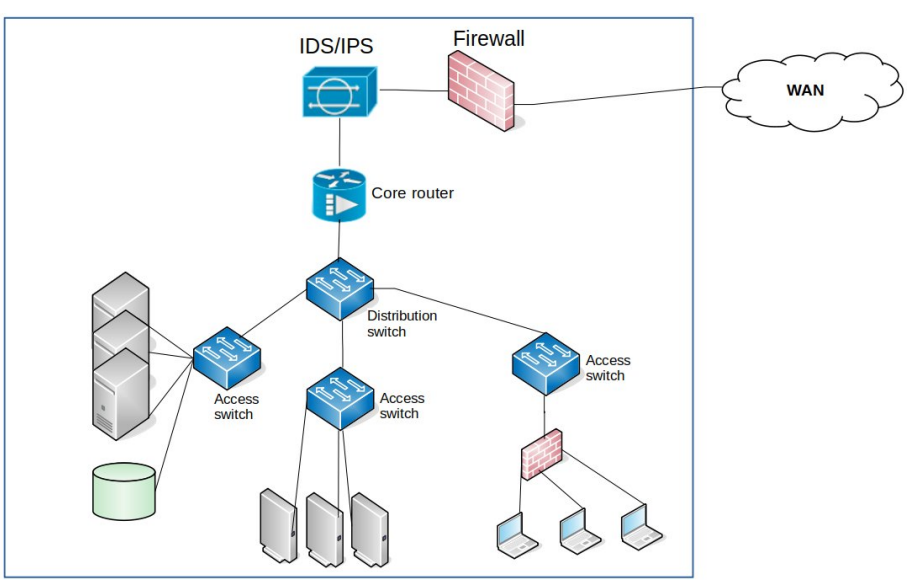

Figure 1 Logical topology of an enterprise network.

Recent defense strategy evolves in favor of detecting intrusions rather than preventing since network attacks proliferate in terms of variety and become more sophisticated to prevent at outer perimeter of the enterprise. Moreover, direct prevention of the suspicious activity might lead denial of the service to legitimate traffic due to false positives. To elaborate the concept of IDS, Figure 2 provides a general taxonomy which differentiates IDS, based on data collection methods and attack detecting techniques (Singh & Singh, 2012).

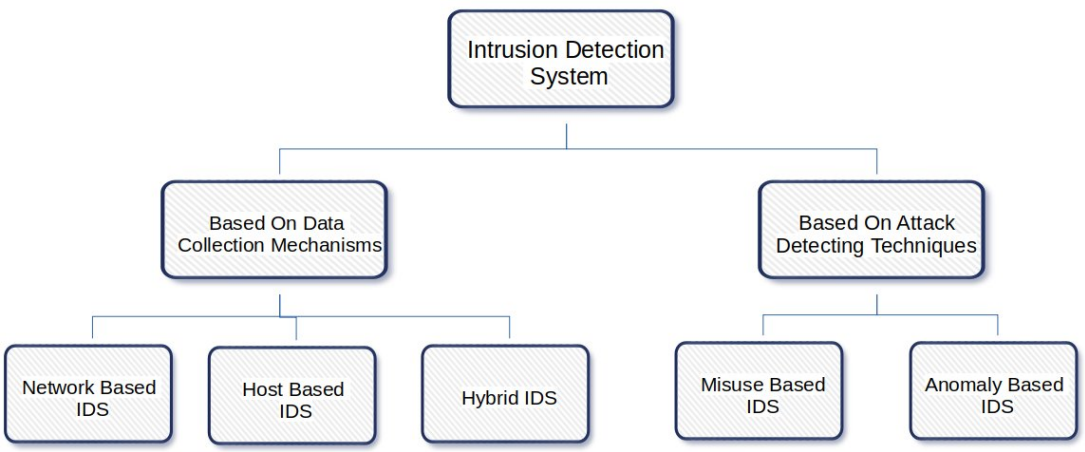

Figure 2 Intrusion Detection Systems taxonomy based on data collection methods and attack detecting techniques

Evaluating the concept based on the data collection mechanism, host-based IDS (HIDS) analyzes behavior on endpoints, namely hosts. In other words, HIDS monitor system resources such as memory consumption, CPU utilization or disk I/O and track processes to detect an abnormal activity. To that extent, their area of responsibility covers the host on which they operate. Majority of HIDS look for indicators of a threat in places such as operating system log files or another specific software‘s audit logs (Bhattacharyya & Kalita, 2013). Alarms are created according to the predefined rules. Since they operate on rules, their performance of detecting known intrusion attempts are quite high. However, they are not quite effective on spotting novel attacks.

On the other hand, network-based IDS (NIDS) captures data flows through the network with some sensors and subsequently analyzes them to recognize suspicious incidents (Liao, Richard Lin, Lin, & Tung, 2013). Such systems might work standalone at the entry point of protected network to capture and inspect every incoming and outgoing packet. Alternatively, they could be placed in another network segment. By using tap devices planted in critical nodes, local area network (LAN) and wide area network (WAN) traffic might be sent to them for inspection. NIDS might work either rule-based or anomaly-based as will be explained. Both types have their strengths and weaknesses.

Hybrid IDS intend to combine the strengths of both HIDS and NIDS. This approach advocates the placement of IDS both on hosts to inspect local incidents and on the network to analyze traffic flowing through. This idea might sound plausible at first. However,

management of both systems brings a higher administrative overhead to information security professionals and system administrators.

Another distinction is made based on attack detecting techniques. Misuse-based IDS operate to recognize known attacks by using specific behavior created by them, in other words, their signatures (Buczak & Guven, 2016). They constitute an effective solution against the intrusion attempts which have been well-recognized and documented. Rules need to be created addressing each attack type and required actions are instructed to the IDS in existence of a match to a rule. However, misuse-based IDS has its disadvantages. They simply cannot detect unknown or zero-day attacks. Even worse, every attack type needs to be converted to a rule to ensure detection which means frequent updates are required for an effective protection. Snort, Suricata and Zeek (previously named Bro) are the most popular and open-source examples of misuse-based IDS. To illustrate the rule concept, a Snort rule is given in Figure 3. Interpreting this rule, it creates an alert if there exists a connection attempt from any external IP and port to SQL Servers' tcp port 1433 and if the payload includes string "sa" and the intrusion attempt takes place more than five times within 2 seconds. Previously stated weakness could be seen here that if the same attempt takes place once in a second (or in any frequency lower than five times in two seconds), misuse-based IDS cannot detect that and brute force login attempt might be successful. Since network scanning and packet crafting tools are easily available now, it is trivial for a seasoned intruder to evade such rules.

```
alert tcp $EXTERNAL_NET any -> $SQL_SERVERS 1433
(msg:"Attack Detected"; flow:to_server,established; content:"|02|";depth:1;
content:"sa";depth:2;offset:39; nocase; detection_filter:track_by_src,count 5,seconds 2;)
```

Figure 3 Snort rule to detect MS SQL Server unauthorized login attempts.

On the contrary, anomaly-based IDS works on the principle of detecting a deviation from the expected system behavior. For instance, the more common type of anomaly-based network IDS observe irregular congestion or occurrences of unexpected behavior such as too many packet re-transmissions in the traffic. To elaborate, overarching goal of a anomaly-based IDS is to analyze and categorize network flow to discriminate malicious attempts from normal occurrences (Bhattacharyya & Kalita, 2013). In contrast to misuse-based approach, there is no certain rules to classify what constitutes an intrusion attempt. Anomaly-based IDS achieves this distinction by two methods on a broad categorization. First, statistical methods which creates a model by applying statistical tests on previous instances. Then, each instance is being tested against the model for classification (Liao et al., 2013). Second, machine learning techniques which will be evaluated in detail in the following paragraphs.

2.2. Machine Learning Techniques for Anomaly Detection

Machine learning (ML) has its place in intersection of many scientific fields such as computer sciences, statistics and natural sciences. It constitutes one of the most popular fields of study with many state-of-art applications and prospective opportunities. Information security and network security in particular is no exception to this trend that ML proposes solutions to many longstanding problems. In early days of the concept, it was defined as

“field of study that gives computers the ability to learn without being explicitly programmed” (Samuel, 1959). The main claim of machine learning is being able to execute human-level tasks by computers. Since they are all popular, the terms artificial intelligence, machine learning and deep learning might be used interchangeably. Figure 4 clarifies the conceptual hierarchy between them as it has also been well recognized by previous studies (Mitchell, 1997; Russel & Norvig, 2016; Tsai & Yu, 2009; Witten, Frank, Hall, & Pal, 2017).

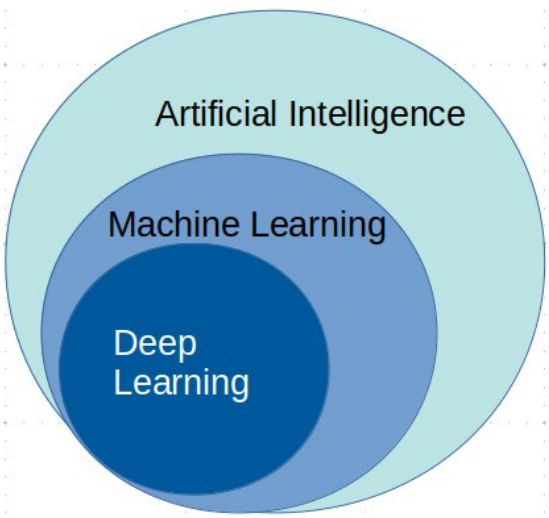

Figure 4 Artificial intelligence and its subdomains.

Since this study’s focus is on anomaly detection using machine learning, previous studies will be evaluated in that context. There is no consensus on the classification of ML methods for IDS. However, in a broad taxonomy, there are five types of machine learning methods for anomaly-based intrusion detection as it is shown in Figure 5 (Bhattacharyya & Kalita, 2013). Providing samples from each and every method is far beyond the scope of this study. However, prominent sub-methods and related work will be evaluated.

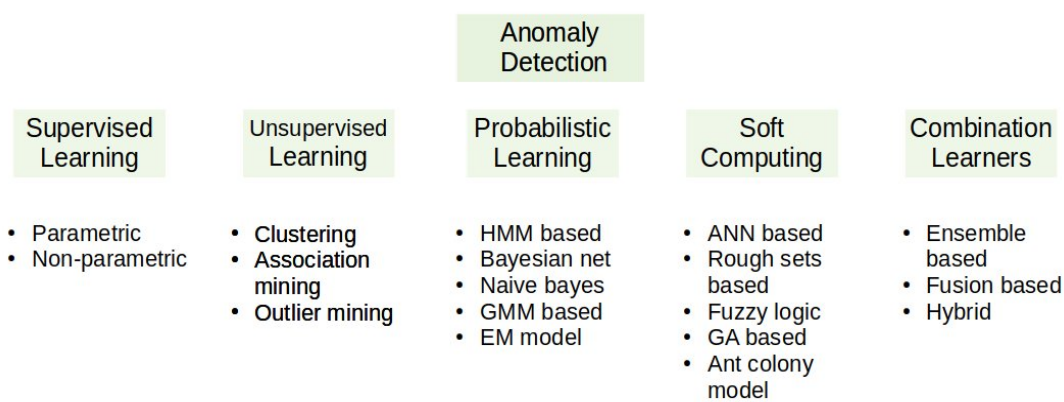

Figure 5 Machine learning methods for anomaly-based intrusion detection.

- **Supervised learning** assumes the existence of a distinction in data set on which a flow/packet's status can be seen as attack or not. Or sometimes more, which type of attack that was occurred. Then, a classifier is trained based on the labeled data set. After that, new instances are tested in accordance with the classifier to determine their membership to a class. The model's success is measured on the basis of accurate predictions versus false positives and false negatives. Based on the parameters being fixed in size or not, there are two types of supervised learning methods. Parametric models summarizes data with a fixed set of parameters no matter what the sample size is. However, in non-parametric models, parameters do exist but they are subjected to change according to data set size (Russel & Norvig, 2016). Machine learning methods such as logistic regression and naive bayes are placed in the first category whereas k-nearest neighbors (KNN), decision trees and support vector machines are in the second.

Ubiquitous work is available using supervised learning for anomaly-based IDS. To summarize some, Chikrakar and Chuanhe proposed a model in which similar data samples are grouped by k-nearest neighbor techniques and then output clusters are classified by SVM classifiers. In this model, KNN and SVM methods are employed and a high accuracy is obtained. On the other hand, authors acknowledged the complexity issue if the data set grows larger (Chitrakar & Chuanhe, 2012). Singh et al. suggested a model in which naive bayes and decision tree methods are used together to balance detection and false positive rate (Harbi, Singh, & Zahidur Rahman, 2010). It demonstrated satisfying false positive and detection rate, whereas remote to local attack class prediction needs further improvement. Peddabachigari et al. proposed a model which combines decision tree and SVM methods. They first sent the data through the decision tree model for initial feature generation. And then generated features along with the original ones were sent to SVM model to get the final classification (Peddabachigari, Abraham, Grosan, & Thomas, 2007). This model demonstrated good performance on KDD cup dataset. On the other hand, compared to SVM-only model results, no significant improvement could be seen.

- **Unsupervised learning** models has the advantage of being able to be built without labeled data sets. Since labeling process often requires human expertise and commonly includes manual steps, this advantage becomes even more important. There are three types of unsupervised learning models. First, clustering based methods does not require classes besides labels. Model discriminates classes evaluating the given training data set. Second, association mining seeks to identify events that occur together frequently. This methods utilizes statistical methods effectively such that for building an association rule, it calculates two parameters: support and confidence. After that, inferences can be made for future instances using those associations. Third, outlier mining looks for the patterns that does not match with the majority of data. Compared to the first sub-method of clustering, outliers can be categorized as the instances which do not belong any of the clusters. Critical point in outlier mining approach is to determine what constitutes an outlier. While doing this, statistical and other machine learning methods are employed.

To enumerate previous work, Syarif et al proposed a model using five different clustering algorithms and compared results with misuse-based models' efficiency. Four of their models performed better than their comparatives (Syarif, Prugel-Bennett, & Wills, 2012). However, high false positive rates are acknowledged as an area which is in need of further

development by the authors. Zhang and Zulkernine suggested a model based on outlier mining with random forest algorithm (Zhang & Zulkernine, 2006). The model is claimed to be more flexible and performing better than supervised models since it is not dependent on labeled data or the training context of a particular network. However, the authors emphasized a weakness of unsupervised models which is the performance deterioration under high rate of attack instances such as a DOS attack occurrence. To alleviate this problem, misuse-based models can be used in combination with the proposed model.

- **Probabilistic learning** models provides an estimation on new instances which are affected by randomness and other probabilistic uncertainty (Bhattacharyya & Kalita, 2013). Most distinctive feature of probabilistic models can be stated as being able to update the previous estimates based on new evidence learned from training data. Tapiador et al proposed a model whose states and transitions between them are specified from HTTP protocol traffic (Estévez-Tapiador, García-Teodoro, & Díaz-Verdejo, 2005). It obtained probabilities of the symbols to the Markovian source during the training phase by using simulated attack-free traffic. The results demonstrated acceptable detection and false positive rates. However, authors underlined the need for retraining frequently as a downside of the model. Aung and Oo suggested a model which used frequent pattern growth algorithm (Moh, Aung, & Oo, 2015). This algorithm is an association algorithm derivative which is frequently used for extraction of item sets from large data sets. Then, authors employed item sets to build frequent pattern trees which will be used to detect anomalies in the traffic. The results demonstrated capability of the model being able to handle the traffic and detect anomalies.

- **Soft computing** models emphasize utilization of randomness factor in contrast to conventional computing's exact communication. It can be achieved by using probabilistic models and fuzzy sets. Soft computing methods can be listed as fuzzy sets, rough sets, ant colony algorithms, artificial immune systems and genetic algorithms. Hamamoto et al proposed a model which combines fuzzy logic and genetic algorithm (Hamamoto, Carvalho, Sampaio, Abrão, & Proença, 2018). The first was used to generate dynamic signatures from network flow and the second classifies the instances as anomalous or not. The authors claimed a high detection rate with low false positive rate of below 1% which is quite accurate in anomaly-based IDS. A downside of the model could be high complexity stemming from combination of numerous methods. Amin et al suggested a model employing rough sets theory and compared it with four other machine learning algorithms (Amin, Anwar, Adnan, Khan, & Iqbal, 2015). The authors underlined the results showing that rough sets based model outperforms the other four being tested on KDD Cup data set.

- **Combination learners** as the name implies, combines multiple learning techniques which were discussed previously. The main goal for doing that is to alleviate weaknesses of each model and increase the overall capability of the combined model. Ensemble methods, fusion methods and hybrid methods can be placed under this category. Pham et al suggested a model which employs an ensemble method (Pham, Foo, Suriadi, Jeffrey, & Lahza, 2018). This model compared two ensemble methods which are bagging and boosting. J48 decision tree classifier is used as the

base classifier. Results are evaluated on NSL-KDD data set and bagging method's classifier could be seen as more accurate in both detection and false positive rates.

Deriving from the discussion, it could be argued that ML methods for anomaly-based intrusion detection demonstrate neither superiority nor inferiority to each other in general. However, there are driving factors which encourage researchers to employ a specific method. For instance, data's being labeled or not, proposed system's logic of work being real time or offline, computation power available to use, field experts being available to manually intervene in the data and volume of the data to be processed for a second are some of the questions for a researcher to ask to determine the method of choice. In short, a case basis evaluation should be done to evaluate the success of chosen method.

In this chapter, intrusion detection methods were discussed with respect to their categorization. Then, anomaly-based IDS was focused on. Machine learning techniques for anomaly-based IDS were enumerated and each method was discussed based on some seminal previous work. In the next chapter, this study's methodology will be explained and justified.

CHAPTER 3

METHODOLOGY

In this chapter, methodology of the study will be explained with its justifications. First, well-known public data sets will be discussed. Then, chosen data set and the way it was collected will be evaluated. Third, the proposed pipeline will be explained and supported with arguments.

3.1. Intrusion Detection System Data Sets

Regardless of the approach, machine learning models need data to be trained with. ML models for anomaly-based IDS is no exception in that sense. There are mainly two categories, public and private data sets. Since the private category is beyond the scope of this study, the well-known public data sets will be evaluated. These are generally used by researchers and for benchmarking purposes.

The Defense Advanced Research Projects Agency (DARPA) 1998-1999 data set is created in a lab environment with artificial attacks for network security analysis. It includes application protocol traffic such as FTP, Telnet, IRC and SNMP. The attack types of probing, rootkit, buffer overflow and DoS are present within DARPA 1998-1999. Weaknesses of it can be listed as being outdated and lacking modern attacks, containing real life irregularities such as absence of false positives and old-fashioned network architecture of the lab environment (MIT Lincoln Laboratory, 1998). Another popular data set is Knowledge Discovery and Data Mining (KDD) 1999 which is an updated version and first derivative of DARPA 1998-1999. It includes different attack instances than its parent such as smurf and neptune. It also has a categorization that there are four classes of attacks which are DoS, user to root, remote to local and probing attacks. However, it is criticized of being highly redundant and having corrupted data (Stolfo, Fan, Lee, & Prodromidis, 1999). To address those deficiencies, NSL-KDD was proposed (Tavallaee, Bagheri, Lu, & Ghorbani, 2009). Although it mitigates statistical weaknesses in the data set, NSL-KDD still suffers from the lack of modern attacks and real-world samples.

On the other hand, there are modern public data sets for IDS research created by universities and/or research groups. Recognizing the need for a modern IDS data set, Moustafa and Slay proposed UNSW-NB15 data set (Moustafa & Slay, 2015). It was also created in a lab environment as previous examples with artificial malicious and benign instances created by a commercial network simulation device. Specifically, such tools enable the simulation of modern attacks by automatically collecting information from Common Vulnerabilities and

Exposures (CVE) web sites, then creating and sending network attacks accordingly. Moreover, the data set provides both packet capture (pcap) files from flowing traffic and extracted features from raw data. These features are extracted by using publicly available argus tool and some custom scripts by the authors. Another set of public data sets for IDS studies are prepared by Canadian Institute for Cybersecurity (CIC). Those are named in CICIDS[Year] format such as CICIDS2017 (Sharafaldin, Lashkari, & Ghorbani, 2019). CICIDS data sets are prepared within a lab environment. Employing an abstraction methodology which is called B-Profile, researchers created twenty five users with simulated human behavior. Eleven criteria are defined to make data set reliable and complete such as complete network configuration, attack diversity, heterogeneity and metadata availability (Sharafaldin et al., 2019). Third, another institution which prepares data sets are Stratosphere Lab. Artificial traffic with malicious, background and normal profiles are created and captured. Although their area of focus is on malware and DoS attacks primarily, it is possible to find modern attack instances within these data sets (Garcia, 2020).

The publicly available IDS data set list is not limited to beforementioned ones. Although not being popular as previously stated examples, DEFCON data set by Shmoo group (2000-2002), Lawrence Berkeley National Laboratory (LBNL) data set (2004-2005), CDX data set (2009) by United States Military Academy, Kyoto data set (2009) by Kyoto University and Twente data set (2009) by University of Twente are other public examples of IDS data sets.

3.2. Selected Data Sets for the Research

As previously discussed, not too many studies exist on real world data in IDS research despite some exceptions (Bhattacharyya & Kalita, 2013). Majority of the previous work tends to be on public or private data sets prepared within lab environments with the help of network traffic simulators. However, limitations caused by using artificially created data sets can be summarized as:

(1) Lack of real-life behavior profiling for a specific enterprise network,
(2) Lack of representation on real-life threats by the model,

First point could be explained such that, every organization has its way of providing services or designing network architecture. Service X might be provided by organization A but not by B and C. However, artificially created data sets tries to cover all types of attacks since being in a controlled environment. Thus, if one trains a model with a simulated data set and put this model on his own enterprise IDS, this model would be trained against some attack types that is impossible to be exposed. In short, enterprise IDS model needs to be trained with real-world traffic to provide more reliability. Not only the malicious simulated traffic but also normal traffic does not reflect specific enterprise user's behavior since it was created by simulators. Second, attack simulators try their best to create attacks in accordance with discovered vulnerabilities. However, since there always will be other alternatives to exploit a weakness, different payloads might be created in a real-life scenario. For instance, Hulk is a well-known and recent example for application layer DoS attacks. And its most popular implementation is by Shteiman's python script which is also available online. But even in the GitHub's public repositories, it is possible to find numerous derivatives from the original script. The wildness of the real-world has the possibility to be attacked with all these derivatives whereas simulated environments does not.

3.2.1. Institutional Data Set

Because of the reasons stated in the previous chapter, network data which was collected from institutional network will be analyzed in this study. Collection was made from the pop-up network interface of the organization with tcpdump tool. Sessions initiated from outside was focused on to decrease complexity with the assumption of expecting the threat from outside is more probable. Figure 6 demonstrates the simplified setting where the data is being collected. In here, traffic captures are taken from network interface A. Five captures were made within both daytime/working hours and weekends or after work hours. Sessions/flows were extracted from the packet level data. Later, those data sets were shuffled and randomly sampled to produce a representative and manageable one. The final data set has 100,000 flows/sessions.

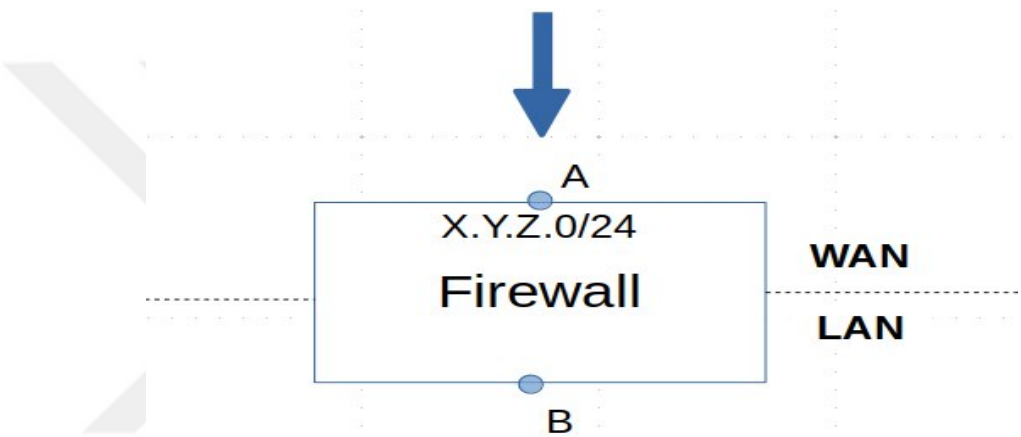

Figure 6 Logical representation of packet capture methodology with respect to network interfaces on firewall appliance in the institutional network

To provide more insight about the institutional data set, some statistical information will be provided. There were around 481,000 sessions before random sampling was made. These were initiated by 38,804 unique hosts. Although this number seems high, majority of traffic came from the frequent clients such that top 20 of them initiated 36% of all the sessions. The traffic that was sourcing from those clients was legitimate as far as probing attacks concerned. Furthermore, there were 256 unique destinations in the data set. However, a similar behavior to source was present such that the top 5 destinations received 57% of the whole sessions. Destination port and service demonstrated heterogeneity as well. Independent of their availability, http, dns, telnet, smb, sql, smtp and ssh became the mostly demanded applications in the data set so that their weight reached to %55 of the all sessions. Average duration was 6.06 seconds, average exchanged bytes were 41,385 and average packet count were 62 for a session throughout the data set.

On the other hand, there are some limitations of choosing the real world data. Firstly, since it is captured from a production environment to ensure validity, there is a significant risk to break things. Because, even the tcpdump process consumes a significant amount of resources. One remedy for this could be the employment of tap devices if any available. Second, operating on production systems reduces flexibility, in other words, it is not possible to try too many alternatives on the target systems as it could be possible in a simulated environment.

3.2.2. UNSW-NB15 Data Set

The other data set to be used for benchmarking and validation purposes was UNSW-NB15. It was prepared by the cyber security research group at the Australian Center for Cyber Security (ACCS) (Moustafa & Slay, 2015, 2016). A network simulation tool was used to generate both normal and abnormal traffic. Attacks which were crafted by the simulator could be grouped in nine such that fuzzers, analysis, backdoors, DoS, exploits, generic, reconnaissance, shellcode and worms. Simulator updates itself with the information from a CVE web site and created related attacks accordingly. For capturing the traffic, tcpdump tool was used as it was done in institutional data set. All capturing period lasted for two days; 16 hours on Jan 22, 2015 and 15 hours on Feb 17, 2015 totaling to 31 hours. The total size of pcap files was 100GB and they were divided into files of 1GB each. Then, argus and bro tools were used for feature extraction. Moreover, twelve custom scripts were also utilized to harvest more features. Framework architecture for UNSW-NB15 data set generation is presented in Figure 7. Resulting feature set consists of 48 features as it could be seen in Appendix A. Argus and Bro files with extracted feature sets in .csv format were made publicly available for non-commercial purposes. Total data set is fairly large with the total number of 2,500,000 flows/sessions. As it was done in institutional data set, 100,000 flows were extracted randomly.

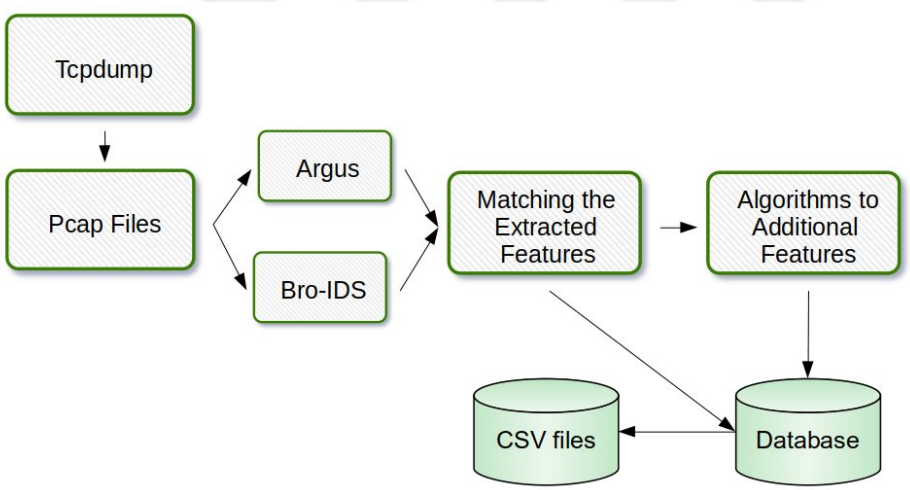

Figure 7 Framework architecture and workflow of generating UNSW-NB15 data set.

3.3. Proposed Methodology

Anomaly-based IDS using supervised machine learning method was chosen for this thesis. There are four attack types which were previously categorized as DoS, user to root, remote to local and probing attacks. Training a model for all attack types is beyond the scope of this study. Therefore, probing type of intrusion attempts was focused on. This type consists of attacks such as ping sweep to discover alive hosts and port scan to determine the services provided by them. As it is demonstrated in Figure 8, probing constitutes a part of reconnaissance phase of cyber kill chain which is “where the story begins” (Lockheed

Martin, 2020). To accomplish benchmarking and validation, corresponding attack type of “reconnaissance” was chosen from the UNSW-NB15 data set.

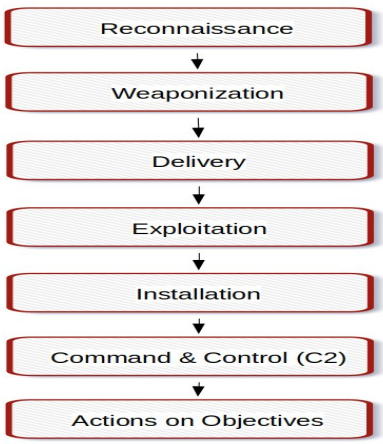

Figure 8 Cyber kill chain proposed by Lockheed Martin.

Training and testing the model was done offline. This is because of the beforementioned limitations such that the author does not have authorization to apply tests or install new features on production systems. Therefore, packet captures were taken and all the work was done offline as it will be explained. There are two pipelines built for the research. The first is for the data set preparation and the second is for model training and testing. After the model was trained and tested on the institutional data set, the same process was applied on the UNSW-NB15 data set which was recognized as a modern IDS bench-marking data set for validation (Divekar, Parekh, Savla, Mishra, & Shirole, 2018). Since the data set preparation was done from scratch, it involves more steps compared to the similar research done with ready data sets. Moreover, this stage consumes the most time in whole research with nearly 80% as it was recognized in similar studies (Gil, 2019). Figure 9 illustrates data preparation pipeline whose steps will be explained one by one and Appendix C has the related python script for the feature selection phase.

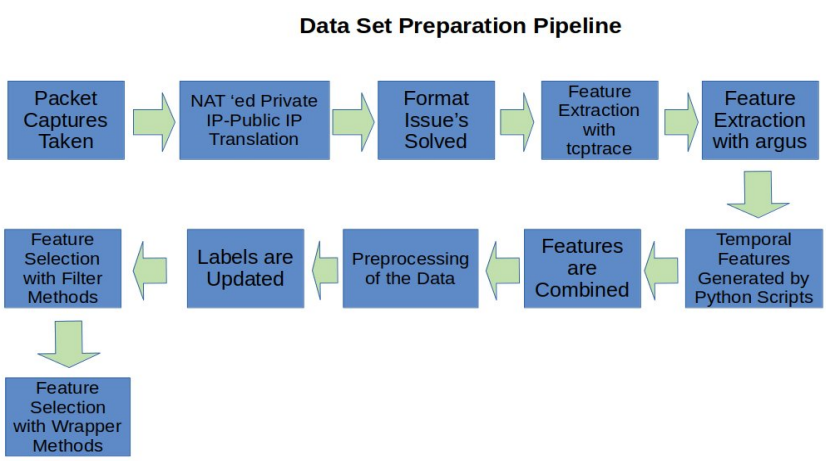

Figure 9 Data set preparation pipeline to apply on institutional packet captures for obtaining data frames with selected features.

3.3.1. Data Collection and Formatting Phase

After packet captures were taken, it was divided into parts of 2 million packets each to process conveniently. Then, network address translation (NAT)'ed IP addresses were converted to public addresses (Figure 10). This is because clients who are coming from WAN knows about the public IP of the server (A.B.C.D). However, border router of the institution translates this address to a private network address since public addresses are limited but there are ubiquitous servers in LAN. This causes problem because in later steps, both argus and tcptrace tools cannot extract flows from packet level data because IP addresses within a flow are not consistent. To overcome this problem, tcpwrangler and tcprewrite tools were used. These two have the same functionality apart from the first has a graphical user interface (GUI).

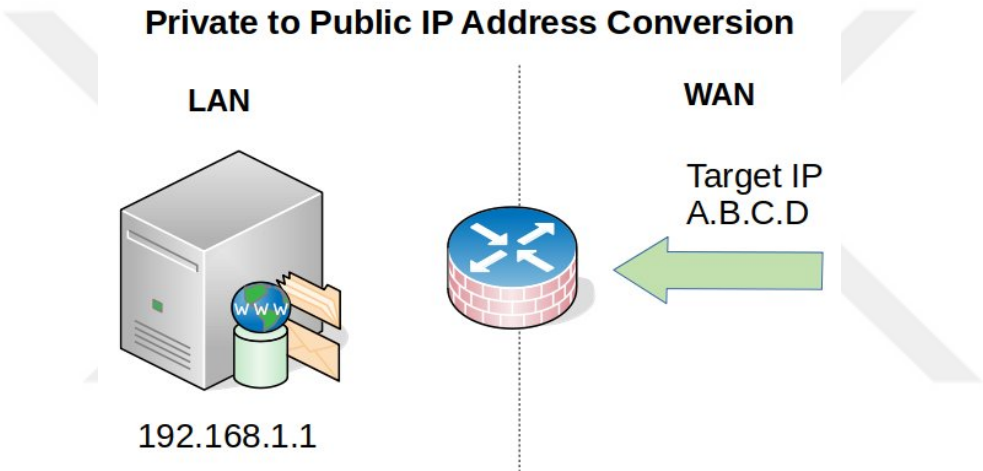

Figure 10 Private to public IP address translation realized on firewall appliance.

After that, format issues were solved. The packets were captured with tcpdump tool on a production firewall's related interfaces and default format given by underlying libpcap library of tcpdump was linux cooked capture (Linux_SLL). These packet captures could not be read by wireshark and argus in later stages. The problem was discovered that there was no layer 2 (OSI model data link layer) addresses in Linux_SLL format. Therefore, pseudo-random MAC addresses were assigned by tcprewrite tool to overcome this issue. The probability of this arbitrary assignment would cause a bias is evaded since there will be no feature extraction from layer two in the models.

3.3.2. Feature Extraction from the Data Set

Then, feature extraction step was started. Building a model based on flow-level intrusion detection is more common in the field since it has two important advantages. First, much more computational power and time is needed to build a packet-level IDS model. This is because a packet capture of 250MBs includes around 2 million packets whereas the same has only around 15,000 flows. Sample size linearly affects model training time and resources at the best case. Second, some attacks only become visible when packets are evaluated in their

sessions. This is because, in some cases, the payload is only meaningful when it is complete after a session. There were two command line tools used for flow-level feature extraction from packet-level data which are tcptrace and argus. Alternative methods for flow-level feature extraction do exist such as using NetFlow protocol and tools. However, NetFlow data, depending on the version, has only around ten features by design. Conversely, utilized tools in this research can give a much more detailed picture of each session by providing more than two hundred features in total. Supporting with the related parameters, tcptrace could extract 97 features whereas argus could do 131 for each flow. A sample raw feature output from each tool (argus and tcptrace) is illustrated in Table 2 and Table 3 respectively, official documentation could be examined for the comprehensive explanation of column headers (Argus, 2020; Ostermann, 2020).

Table 2 Sample raw features extracted with argus tool from institutional network packet captures.

StartTime	**Flgs**	**Proto**	**SrcAddr**	**Sport**	**Dir**	**Dport**	**ToPkts**	...
1580023501.44141	e	tcp	123.123.123.123	1234	-->	80	3007	...
1569582856.7375	M	tcp	111.111.111.111	2345	---	25	2193	...
1569582857.73946	e S	tcp	222.222.222.222	5432	<--	1433	6543	...
...	...	...	...	...		...	...	...

Table 3 Sample raw features extracted with tcptrace tool from institutional network packet captures.

first_packet	**host_a**	**host_b**	**port_a**	**port_b**	...
1580023501.44141	123.123.123.123	111.111.111.111	1234	80	...
1569582856.7375	111.111.111.111	231.231.231.231	2345	25	...
1569582857.73946	222.222.222.222	110.110.110.110	5432	1433	...
...	...	...	...	...	...

After that, custom features were created for each flow. These are temporal features as it was seen in popular public data sets such as NSL-KDD and UNSW-NB15. Those features intended to catch anomalies by evaluating source behavior with the time. Pyshark python library which was built on command line tool tshark along with pandas and numpy libraries were used to write required scripts. Specific TCP header flags were considered as signs of probing behavior using nmap's scanning documentation. These are ICMP, syn, syn-ack, null, fin, xmas (psh-urg-fin) and fin-ack. In short, script extracts the packets with one of the values in their TCP header. Then, evaluating by each flow, it counts the occurrence of each header value by source IP for a given time period of 2 seconds. These values are expected to be an estimator for the source being malicious or normal. A sample output of temporal features is illustrated in Table 4.

Table 4 Sample temporal features extracted with custom scripts from institutional network packet captures.

StartTime	**SrcIP**	**ICMPCount**	**SYNCount**	**SYNACKCount**	...
1569581994.93719	123.123.123.123	0	32	0	...
1569581991.5029	111.111.111.111	64	2	0	...
1569581993.46201	222.222.222.222	0	3	1	...
...	...	...	...	...	...

At that stage, there were three sets of features present, from tcptrace, argus and temporal features. Finally, these sets were combined with an outer join principle taking the session start time as the index column as illustrated in Figure 11. One significant limitation here is that since tcptrace tool and temporal feature extracting script can only extract features from TCP sessions, other flow types (UDP, ICMP) recorded by argus tool have their related columns as missing values for those two feature sets. This issue will be addressed in missing value handling phase.

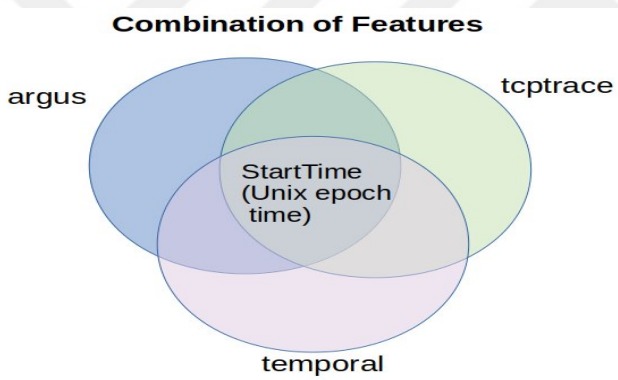

Figure 11 Combination of three sets of features which are gathered from argus and tcptrace tools and the custom script based on start time column.

3.3.3. Preprocessing and Labeling

Preprocessing step constitutes a critical phase in any data analytic project. At this stage, three main steps were executed with the help of pandas and numpy python libraries. First, redundant/unwanted columns were dropped. There were three types of unwanted features. To begin with, repeating columns were handled. For instance, source/destination port was extracted by both tcptrace and argus with different column names as it can be seen in Table 2 and Table 3 (SrcPort and port_a respectively). Second, some features present no variation at all. For instance, argus output's "retrans" field was always valued zero. And finally, some fields were empty throughout the data set. These three groups of features were cleared at this phase to prevent noise. Second step of preprocessing was transforming categorical features to numerical ones. This is because machine learning models need numerical values to be trained and tested. This process is called one-hot encoding. For instance, there was the

feature of “State” indicating a session’s status which had nine possible values of ACC, CON, ECO, FIN etc. It was turned into State_ACC, State_CON, State_ECO etc. and each session had a binary value for each of these generated features. Table 5 and Table 6 illustrates this transformation. Third, missing values were handled. While doing this, two different steps were taken. First, if a value is missing and it is probable for this sample to get a value, statistical summary functions such as mean and median were used to fill those missing values. For instance, a sample session could miss “TotalBytes” feature due to a problem in the tools used. Because every session has a “TotalBytes” value, this could be replaced with the mean of the whole sample space. On the contrary, all UDP sessions missed “TCPBaseAddress” feature which is the intended behavior since UDP sessions do not have sequence numbers or base addresses. For such missing values, an impossible value was given as a placeholder.

Table 5 Combined feature set for institutional data set before one-hot encoding.

StartTime	**SrcIP**	...	**State**	**Dur**	...
1569581994.93719	123.123.123.123	...	RST	0.002239	...
1569581991.5029	111.111.111.111	...	CON	0.986554	...
1569581993.46201	222.222.222.222	...	REQ	5.075901	...
...	...	...	...	...	...

Table 6 Combined feature set for institutional data set after one-hot encoding.

StartTime	**SrcIP**	...	**State_REQ**	**State_CON**	**State_ECO**	...
1569581994.93719	123.123.123.123	...	0	0	0	...
1569581991.5029	111.111.111.111	...	0	1	0	...
1569581993.46201	222.222.222.222	...	1	0	0	...
...	...	...	...	...		...

After that, labeling was done. For this step, three major sources were used. First, the firewall has a misuse-based IDS component whose rules are live updated from internet. A sample alert by firewall IDS component is illustrated in Table 7. Second, packet capture files were transferred into an environment with two open source IDS software, Snort and Suricata installed. They were scanned with Snort and Suricata and created alerts were taken as labels. Test environment is illustrated in Figure 12. Suricata and Snort rules are managed with Pulled Pork which downloads and installs rules in accordance with emerging threats. After that, Barnyard2 tool was used as a standard output medium of two systems. It provides a MySQL database environment for Snort and Suricata in which rules, alerts, configurations and other meta data could be stored. Another advantage to keep records in a database is that they can be queried with SQL by filtering with complicated criteria. This would not be possible if plain text log files were used. And third, field expertise and architectural knowledge was used. Such as, the author has knowledge of which target hosts are up or which services they are providing. Therefore, a request targeting an unused IP address or port

could be labeled as malicious. Moreover, session-level investigation was made when doubt was present to judge a session is malicious or not. After all, to make this process automated, it was included in a python script. Combining these three label-sets, final labels were created.

Table 7 A sample alert by firewall IDS component.

Time	**Source**	...	**Attack Name**	**Industry Reference**	...
1569581994.93719	123.123.123.123	...	Invalid TCP flag combination	CAN-2002-1071	...
1569581991.5029	111.111.111.111	...	Illegal header format	CVE-2015-0007	...
...	...	...	...	...	...

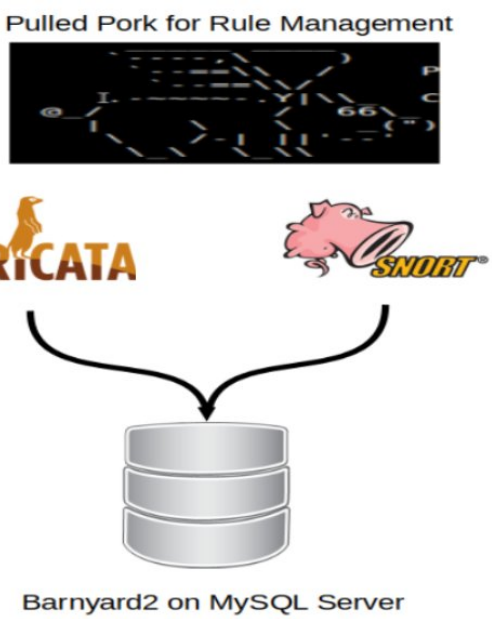

Figure 12 Test environment for labeling packet capture files taken from the institutional network.

3.3.4. Feature Selection

Then, feature selection phase commenced. It is a critical step that affects ML model performance directly. Previous work emphasized the significance of feature selection and proposed various methods for its execution (Alkasassbeh, 2017; Ambusaidi, He, Nanda, & Tan, 2016; Bolón-Canedo, Sánchez-Maroño, & Alonso-Betanzos, 2016; Ganapathy et al., 2013; Nguyen, Franke, & Petrović, 2012). Since more than two hundred features were present in combined data frame for the institutional data set, a robust feature selection process was required. Reducing the number of features has two main benefits in machine

learning. First, extracting a subset from whole feature list helps building a more accurate model by removing the noise caused by unnecessary features (Bolón-Canedo et al., 2016). And second, model training time significantly decreases as the number of features does. The decrease is linear in the worst case depending on training algorithm (Khammassi & Krichen, 2017). Stemming from those points, using too many features in a ML model is defined as "curse of dimensionality" (Iglesias & Zseby, 2015).

There are primarily two methods in feature selection. Filter methods measure features' significance independent from the ML model. More specifically, they look for the correlation between each feature (independent variable) and the target (dependent) variable separately (Ambusaidi et al., 2016). The most prominent advantages of filter method feature selection are that, first they are not computationally expensive. Therefore, regardless of the feature count, a relatively fast output could be harvested. And second, they are less prone to over-fitting. On the contrary, the most significant drawback is that they cannot detect correlation and redundancy between the features. For instance, let A and B be features and X is the target feature. A and B on their own might have high correlation with X which is a desirable thing. Filter method can detect that successfully. However, A and B could have high correlation between each other as well which deteriorates the model's accuracy. And filter methods cannot discover that. Second, there are wrapper methods. They have two different approaches, forward selection proposes bottom-up and backward elimination top-down strategy. In both ways, a ML model is built by selecting a learning algorithm. Then based on its accuracy, features are either added or subtracted from the model until the best feature subset is generated (Ganapathy et al., 2013). The largest advantage of this method is that it provides a more accurate feature subset comparing to filter method. However, there are two main disadvantages. First, they require high computational power such that it is an NP-hard problem this methodology is solving (Khammassi & Krichen, 2017). And second, they are more prone to over-fitting.

Proposed feature selection method for this study consists of both approaches to combine their strengths. Ideal approach to have the best feature subset would be applying a wrapper based method on the whole feature set. However, due to the high number of features and sample size, it is not computationally feasible. Therefore first, a threshold was determined and filter methods was applied to select the features satisfying that. And then, wrapper method is used to make a second selection among that subset to generate the final feature subset. Figure 13 demonstrates the proposed workflow for the feature selection process. This approach has its application on previous work and demonstrated its efficiency (Ambusaidi, He, Tan, et al., 2015; Mohammadi, Mirvaziri, Ghazizadeh-Ahsaee, & Karimipour, 2019).

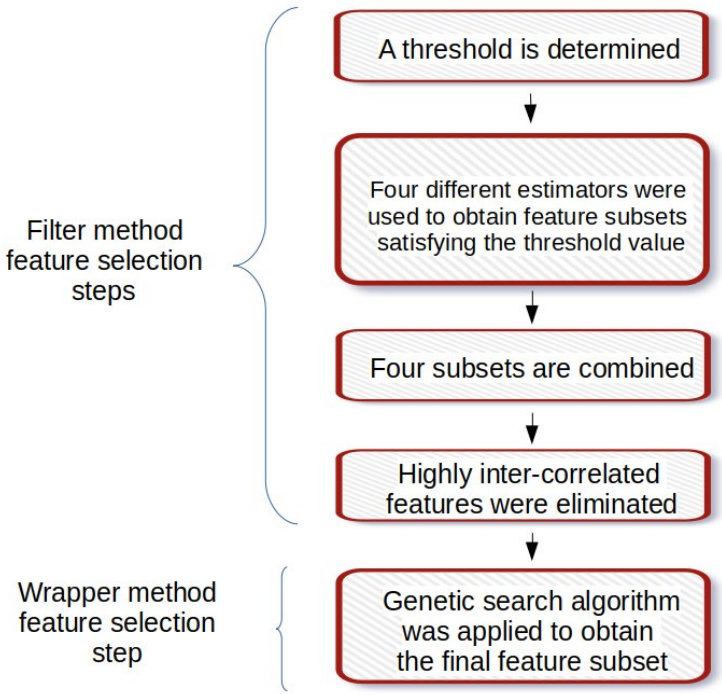

Figure 13 Proposed workflow for the feature selection process.

Numpy, pandas, sklearn and genetic selection python libraries were utilized throughout the feature selection process. First, in filter feature selection method, four different estimators were used. In the first estimator, features with more than 0.3 correlation were selected. In other three estimators, the best 25 features were selected using chisquare, ANOVA f-value and extra-trees predictive methods. It was seen that most of the selected features using four estimators overlap. After that, these four subsets were combined. This aggregated subset consists of thirty two features. However, as it was emphasized before, filter methods cannot discover correlation between features. To overcome this drawback, a correlation test was applied on selected features to eliminate ones which have more than 0.75 correlation among themselves. After this test, 14 features were eliminated with 18 of them remaining as the final subset from filter methods feature selection process. These features are provided in Table 8.

Table 8 Selected subset after filter method feature selection step for institutional data set.

No	Feature Name	Information
1	mss_requested_a2b	The Maximum Segment Size (MSS) requested as a TCP option in the SYN packet opening the connection.
2	DstTCPBase	Destination TCP base address.
3	min_segm_size_a2b	Minimum segment size from source to destination.
4	PCRatio	Producer/consumer ratio. A normalized value indicating directionality of application information transfer, independent of data load or rate.
5	idletime_max_a2b	Maximum idle time from source to destination.
6	state_CON	State connected.
7	state_REQ	This indicates that this is the initial state report for a TCP session.
8	state_RST	State reset.
9	state_FIN	State fin.
10	sTtl	Source TTL value.
11	dTtl	Destination TTL value.
12	Dport	Destination port.
13	Dur	Session duration.
14	sMeanPktSz	Mean size in source packets.
15	dMeanPktSz	Mean size in destination packets.
16	max_segm_size_a2b	The maximum segment size observed during the lifetime of the connection.
17	FIN_pkts_a2b	The count of all the packets seen with the FIN bits set in the TCP header respectively.
18	adv_wind_scale_a2b	The advertised window scaling factor used.

Second feature selection phase consists of a wrapper approach to select the best feature subset among the previous selection which was made with filter based approach. Genetic search algorithm which was used in similar previous studies was chosen for this process since its efficiency was presented before (Ambusaidi, He, & Nanda, 2015; Khammassi & Krichen, 2017). Before delving into its implementation, a brief introduction to genetic search will be provided. The idea stems from Darwinian natural selection and genetics theory in biology. Its ML derivative adopted the same terminology as gene, chromosome, population, cross-over and mutation. Let every feature be represented with a binary value of being selected or not, gene corresponds to a single feature, chromosome a possible feature subset and population is a set of possible feature subsets as illustrated in Figure 14 (Mallawaarachchi, 2017).

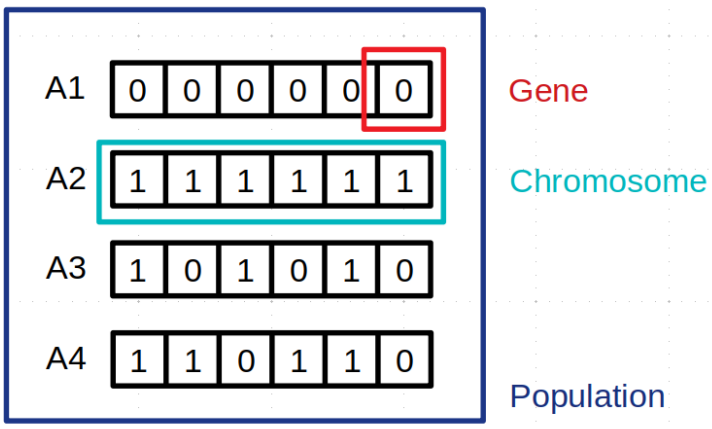

Figure 14 Gene, chromosome and population definitions of genetic algorithm in ML context.

The whole process begins with a random population. Then, according to a predefined fitness function, fitness value is calculated. The overall aim is to minimize the fitness value. After that in selection step, chromosomes with the lowest fitness value within that population are selected. Two new chromosomes are created based on genes of these two to be used in the next generation in crossing-over. And last step of the loop requires a random bit change which is called as mutation. Each one of this cycle is labeled as a "generation". Figure 15 illustrates the workflow of genetic search (Casas, 2019).

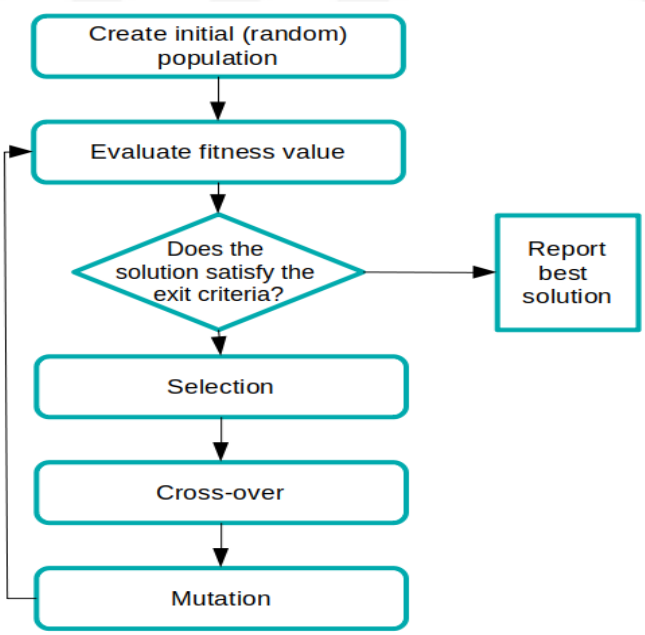

Figure 15 Genetic search feature selection workflow to obtain the optimum feature subset.

The genetic search algorithm was explained in high level. It was implemented by using python's sklearn-genetic library (Calzolari, 2019). Random forest algorithm was used as the estimator and the model was run through 50 generations. Related python script is given in Appendix C. As the result of wrapper-based genetic search feature selection implementation, 5 features were eliminated and there were 13 features left as illustrated in Table 9.

Table 9 Final feature subset as the result of filter and wrapper method feature selection steps combined for institutional data set.

No	Feature Name	Information
1	mss_requested_a2b	The Maximum Segment Size (MSS) requested as a TCP option in the SYN packet opening the connection.
2	DstTCPBase	Destination TCP base address.
3	PCRatio	Producer/consumer ratio. A normalized value indicating directionality of application information transfer, independent of data load or rate.
4	state_CON	State connected.
5	state_RST	State reset.
6	sTtl	Source TTL value.
7	dTtl	Destination TTL value.
8	Dur	Session duration.
9	sMeanPktSz	Mean size in source packets.
10	dMeanPktSz	Mean size in destination packets.
11	FIN_pkts_a2b	The count of all the packets seen with the FIN bits set in the TCP header respectively.
12	adv_wind_scale_a2b	The advertised window scaling factor used.
13	Dport	Destination port.

After that, the before-mentioned feature selection process was applied on UNSW-NB15 data set. Initial feature set consists of 48 features and preprocessing made this number rise to 69 due to one hot encoding. Resulting from the application of filter method feature selection process, 15 features was selected as it can be seen in Table 10.

Table 10 Selected subset after filter method feature selection for UNSW-NB15 data set.

No	Feature Name	Information
1	ct_dst_sport_ltm	No of connections of the same destination address and the source port in 100 connections according to the last time.
2	ct_state_ttl	No. for each state according to specific range of values for source/destination time to live.
3	smeansz	Mean of the row packet size transmitted by the source.
4	dmeansz	Mean of the row packet size transmitted by the destination.
5	ackdat	TCP connection setup time, the time between the SYN_ACK and the ACK packets.
6	state_INT	This indicates that this is the initial state report for a UDP session.
7	Dload	Destination bits per second.
8	service_dns	Application layer protocol for the related flow is DNS.
9	dsport	Destination port number.
10	ct_ftp_cmd	No of flows that has a command in ftp session.
11	dttl	Destination to source time to live value
12	state_CON	State connected.
13	proto_udp	Transport layer protocol for for the related flow is UDP.
14	service_-	Application layer protocol for the related flow is not applicable or not captured.
15	Spkts	Source to destination packet count.

Then, these 15 features were fed into the wrapper-based genetic search feature selection process. Final feature set was reduced to 10 features as the result of feature selection phase for UNSW-NB15 data set as it was presented in Table 11.

Table 11 Final feature subset as the result of filter and wrapper method feature selection steps combined for UNSW-NB15 data set.

No	Feature Name	Information
1	smeansz	Mean of the row packet size transmitted by the source.
2	dmeansz	Mean of the row packet size transmitted by the destination.
3	ackdat	TCP connection setup time, the time between the SYN_ACK and the ACK packets.
4	Dload	Destination bits per second.
5	service_dns	Application layer protocol for the related flow is DNS.
6	dsport	Destination port number.
7	dttl	Destination to source time to live value
8	state_CON	State connected.
9	service_-	Application layer protocol for the related flow is not applicable or not captured.
10	Spkts	Source to destination packet count

3.3.5. Model Training and Testing

Next, utilizing the selected features, ML models will be built. In this phase, there will be two different machine learning methods which will be applied on two different data sets to make comparisons. They were selected on the basis of frequent use in supervised ML models in anomaly-based IDS. First, ensemble methods will be applied on institutional data set and convolutional neural network model will follow that. After evaluation of these models to discover which one was more successful according to predefined criteria, they will also be applied on the UNSW-NB15 data set for validation. This is because UNSW-NB15 constitutes a modern benchmarking data set alternative to KDD Cup-99 and DARPA 98-99 data sets (Divekar et al., 2018). Proposed pipeline is illustrated in Figure 16 and related python scripts are given in Appendix D, E and F.

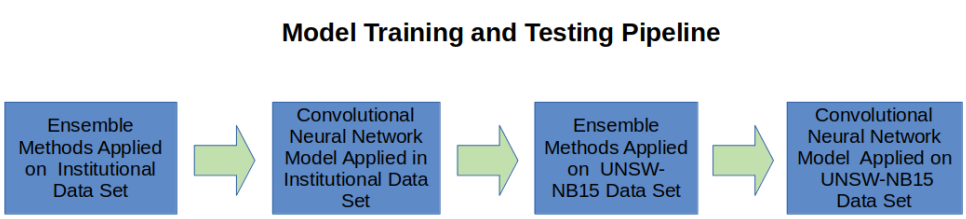

Figure 16 ML training and testing pipeline for both data sets and models.

Ensemble learning methods have their application in anomaly-based IDS frequently. Previous research emphasizes success of the models trained with ensemble methods (Didaci, Giacinto, & Roli, 2002; Gaikwad & Thool, 2015; Panda & Patra, 2009; Peddabachigari et al., 2007; Vanerio & Casas, 2017). In short, ensemble learning methods are meta-learners

which utilize multiple ML models on the same data set to get more accurate results aiming to reduce bias, noise and variance. There are basically three types of ensemble learning models which do similar things with slightly different approaches. Bagging (bootstrap aggregating) chooses one base estimator algorithm e.g decision tree, KNN or SVM. Then by using this algorithm, it trains multiple models on randomly selected subsets of the data. Data samples are taken using bootstrap sampling and with replacement. Then, a classifier is trained on each model and final output is determined by averaging all outputs for regression and voting among them for classification (Borji, 2007). On the other hand, boosting category aims to convert weak learners to strong ones by repeating rounds. The main principle is that it takes weak predictors which are slightly better than random guessing, then in each round more weight is given to misclassified instances to be able to predict them correctly in the next round (Vanerio & Casas, 2017). The main difference between bagging and boosting is that bagging trains multiple models on parallel with randomly selected sample subset, whereas boosting follows a sequential approach on the same data set in each round with tweaked weights for misclassified instances. Stacked ensemble method combines multiple models with a meta-regressor or meta-classifier. In this approach, there are multiple base learners and they are trained with a complete training set without performing any selection, then a meta-learner taking the outputs of base learners as features trained to make final predictions (Smoltakov, 2017).

Adopted approach for ensemble training for this study is bagging method. This is because first, boosting method is for combination of weak learners which predict slightly better than random guessing and stacked method introduces a level of complexity to the model by adding one more layer. And second, bagging methods have the advantage of working in parallel which reduces training duration significantly. Although thesis' model works offline due to the beforementioned limitations, anomaly detection is ideally to be done in real time and increased speed means less delay in model training and testing processes. Bagging ensemble method was used in previous research and proved its efficiency (Borji, 2007; Gaikwad & Thool, 2015; Vanerio & Casas, 2017). The choice of base classifier is critical in bagging methods. Popular methods of choice in previous studies were also chosen for this work. Naive bayes, k-nearest neighbours, logistic regression and SVM were utilized in that sense. Then, their performance will be compared and evaluated to choose the best fit. Throughout the implementation process, python sklearn's related libraries were used.

Second model to be used in this study is convolutional neural networks. Although being significantly successful in fields such as image and video recognition, recommender systems and natural language processing, CNN and its derivative models have their frequent implementation in the field of anomaly detection as well (Kravchik & Shabtai, 2018; Kwon, Natarajan, Suh, Kim, & Kim, 2018; Zhu, Ye, Wang, & Xu, 2018). Promising results from recent research encouraged the author to build a CNN model for this study. As CNN demonstrated its efficiency in image recognition, this strength was used for a robust classification. Python's keras and sklearn libraries were used to build the model.

Both data sets needed a second preprocessing to fit into the CNN model. The first problem was that CNN needs to get input in the form of 2D images. However, institutional data set consists of final feature set consisting of 139 features for each flow which means the input for each sample is a 1x139 vector rather than a 2D matrix. To overcome this issue and converting 1D vector to 2D matrix, Naseer and Saleem's approach was taken (Naseer & Saleem, 2018). 139 features were repeated seven times in such a way to form a 1x973 vector

in which 1st, 140th, 279th, … , 835th element were the same. The second problem was that there needs to be 51 more features to get 1024 features which will later be converted to 32x32 2D feature matrix. Remaining features were added randomly for avoiding the introduction of bias. This was called as “padding” in previous studies. Then, 1x1024 vector was converted to 32x32 matrix to feed into the CNN model. An example flow which was converted to 32x32 grayscale image is illustrated in Figure 17. The same process was applied for UNSW-NB15 data set as well.

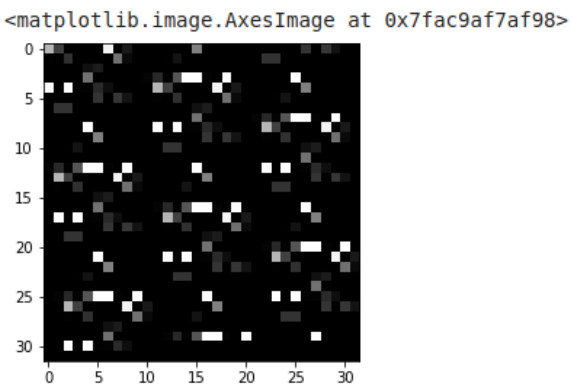

Figure 17 An example flow which was converted to 32x32 grayscale image.

Hyper-parameter tuning constitutes a crucial step in building any kind of neural network. They were chosen in accordance with previous studies and popular trends in similar research. Number of layers were chosen following Heaton’s methodology (Heaton, 2008). He argues that two hidden layers are enough for most of the deep neural networks and more needs to be added by caution. Therefore, starting from two convolutional layers, one layer was added at a time and performance of the model is observed until it reached the optimum point. Other than the convolutional layer depth, there were other hyper-parameters such as batch size, number of epochs, filter size, optimization algorithm, dropout rate etc. Employing a trial and error method is not practical in such a scenario since the high number of hyper-parameters generated too many combinations. To overcome this complexity, python’s hyperas library which works as an extension of keras library was employed. With the suggestions provided by hyperas, automated hyper-parameter optimization was possible. As the result, CNN model was built as it could be seen in Figure 18.

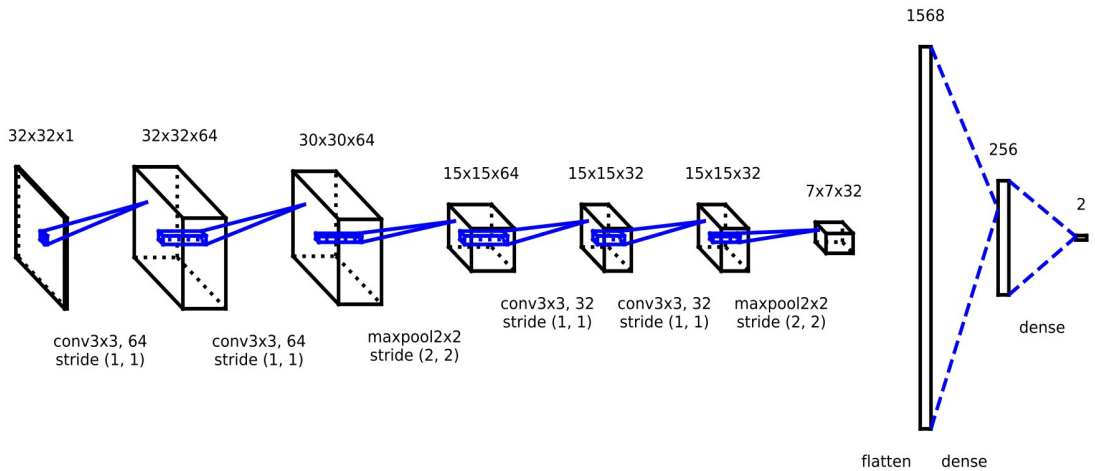

Figure 18 CNN model architecture.

Keras library could also provide a summary for neural network models. It gives a quick review about layers and trainable/non-trainable parameters. As it is presented in Figure 19, all parameters are trainable which means they have been updated during the training and contributed accuracy of the overall model. On the other hand, a significant pitfall in any ML model is its being biased. The most common occurrence of bias is overfitting which means that a model memorizes the training data and adapts itself too much to it. So that it loses the ability to predict any other sample. Overfitting has been a well-recognized problem for long in the literature (Dietterich, 1995; Schaffer, 1993). Solutions against overfitting could be listed as adding more data, generalizable data collection and inclusion to the model, building a simple model and regularization (Ruizendaal, 2017). These were taken into consideration building the model. More specifically, regularization with dropout was applied after each convolutional and fully connected (dense) layer.

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 32, 32, 64)	640
conv2d_5 (Conv2D)	(None, 30, 30, 64)	36928
max_pooling2d_2 (MaxPooling2	(None, 15, 15, 64)	0
dropout_3 (Dropout)	(None, 15, 15, 64)	0
conv2d_6 (Conv2D)	(None, 15, 15, 32)	18464
conv2d_7 (Conv2D)	(None, 15, 15, 32)	9248
max_pooling2d_3 (MaxPooling2	(None, 7, 7, 32)	0
dropout_4 (Dropout)	(None, 7, 7, 32)	0
flatten_1 (Flatten)	(None, 1568)	0
dense_2 (Dense)	(None, 256)	401664
dropout_5 (Dropout)	(None, 256)	0
dense_3 (Dense)	(None, 2)	514

Total params: 467,458
Trainable params: 467,458
Non-trainable params: 0

Figure 19 Summary for the CNN model.

Another precaution to prevent overfitting was dividing the data set into three parts with no overlapping element. This was done for both data sets and models. The common practice of having 60%, 20% and 20%; training, validation and testing parts were applied with random selection. Validation part was only used for hyperparameter tuning, hyperopt and its keras derivative hyperas python libraries were utilized during this process. Hyperopt improves hyperparameter optimization by decreasing training time compared to the other common methods of grid search and random search (Bergstra, Yamins, & Cox, 2013). After that, models were trained with those hyperparameters on training data. And finally, trained models were tested on the testing part to evaluate their accuracy. Train-validaliton-test split is illustrated in Figure 20.

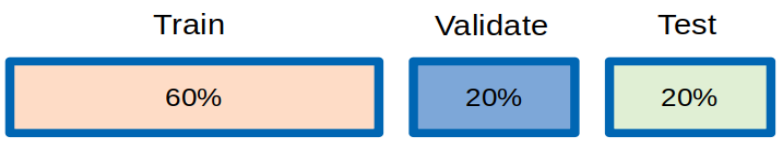

Figure 20 Validation-Train-Test split to prevent overfitting.

CHAPTER 4

RESULTS

In this section, the outcome of the models will be discussed. First, environment in which the models were trained and tested will be explained. Then, results will be given for both models and data sets. The results will be discussed in the next chapters of discussion and conclusion.

4.1. Experiment Environment

A personal computer was used for most of the implementation throughout the study. It has the specifications such that i7-6700HQ 2.6 GHz CPU, NVIDIA GeForce GTX-950M GPU, 24 GBs of memory and Linux Ubuntu 18.04 operating system. For required python script development, three platforms were used. First, PyCharm IDE installed on the local PC. Second, anaconda jupyter notebook environment which was run on local PC. And for those tasks that require robust GPU power, Google Colab's web based jupyter notebook development environment were used.

4.2. Evaluation Criteria

Results' performance were evaluated based on two criteria. First, F1 score was employed. F1 score consists of precision and recall values which on themselves are performance evaluation metrics. Precision measures the ratio of correctly identified positive cases against all positive predicted cases. It can be formulated as in Equation 1.

$$\text{Precision} = \frac{\text{True Positive}}{(\text{True Positive} + \text{False Positive})}$$

Equation 1 Precision formulation.

Recall measures the ratio of correctly identified positive cases from all the actual positive cases. It can be formulated as in Equation 2.

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

Equation 2 Recall formulation.

Based on these two metrics, F1 score can be better explained. It demonstrates the harmonic mean of precision and recall metrics and accepted as a more accurate value for model evaluation. F1 score can be formulated as in Equation 3.

$$F_1 = \left(\frac{2}{\text{recall}^{-1} + \text{precision}^{-1}} \right) = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Equation 3 F1 score formulation.

F1 and accuracy scores are frequently compared to evaluate which one constitute a better performance metric for a ML model. Accuracy score is formulated in Equation 4. There are two justifications for the F1 score to be chosen instead of accuracy in this study. First, the rate of true negatives which in most real life scenarios are not significant as other classes in confusion matrix directly contributes the accuracy score (Brownlee, 2014; Huilgol, 2019; Shung, 2018). However, F1 score is only affected by true positives, false positives and false negatives as being equal to the harmonic mean of precision and recall values. Since the aim of this study is to classify intrusion attempts and they are labeled as positive in the models, F1 score constitutes a more suitable performance evaluation metric. Second, F1 score is more suitable with data sets with imbalanced distributions of positive and negative cases (Berger, 2013; Powers, 2011; Reeker, 2001). In an imbalanced data set, more frequent class manipulates model if accuracy score is chosen as performance evaluation metric since this value needs to be optimized. On the other hand, less frequent class would be given less importance and classification accuracy for the sample belonging to that class would decrease. This is important because, positive instances which represent the occurrences of attacks/intrusion attempts generally constitute the minority of the all cases in IDS data sets and ignorance of them heavily distorts model's reliability.

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

Equation 4 Accuracy score formulation.

Second evaluation criterion was Receiver Operating Characteristics (ROC). Along with F1 score, it is widely accepted as a metric for ML models' performance assessment. It evaluates the relation between recall (sensitivity) and precision (specificity) and depicts the model's performance for every possible threshold value for classification. In other words, ROC is a dynamic evaluation of the model whereas F1 may reflect the prediction capability in a specific threshold point. Area under curve (AUC) constitutes the outcome of ROC evaluation and higher AUC means better performance of the model being evaluated. Evaluation metrics were calculated with python sklearn metrics libraries throughout the implementation.

4.3. Results

First, ensemble models on institutional data set were evaluated. As it was stated in the previous section, four base learners which were SVM, KNN, naive bayes and logistic regression were used to compare their performance. Table 12a, Table 12b, Table 12c and Table 12d demonstrates confusion matrix, F1 and ROC AUC scores for SVM, KNN, naive bayes and logistic regression base learners respectively. Figure 21a, Figure 21b, Figure 21c and Figure 21d illustrates ROC AUC for models built with SVM, KNN, naive bayes and logistic regression base learners respectively.

Table 12a Institutional data set results in ensemble model for SVM base learner.

Results for Support Vector Machine (SVM) Base Learner		
Confusion Matrix		
	0	1
0	10094	160
1	175	9571
F1 Score		
0.9828002259074805		
ROC AUC Score		
0.9832201242953904		

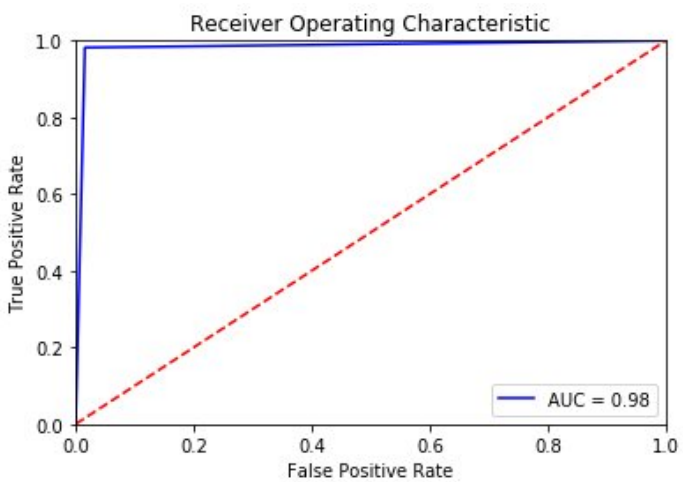

Figure 21a ROC and AUC graphical representation for institutional data set in ensemble model and SVM base learner.

Table 12b Institutional data set results in ensemble model for KNN base learner.

Results for K Nearest Neighbours (KNN) Base Learner		
Confusion Matrix		
	0	1
0	10133	137
1	152	9578
F1 Score		
0.9859653601797176		
ROC AUC Score		
0.9861792167826919		

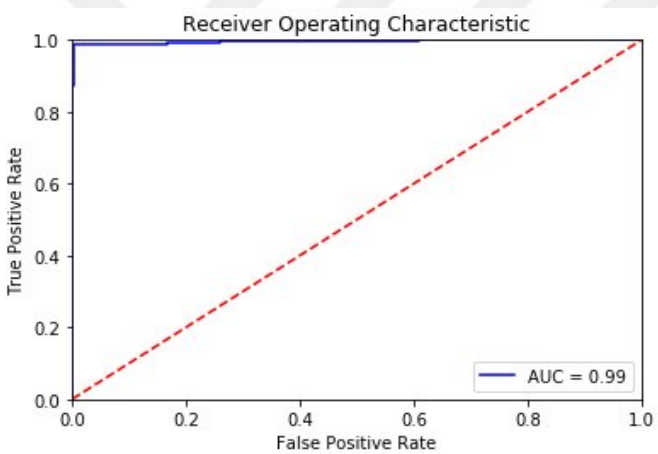

Figure 21b ROC and AUC graphical representation for institutional data set in ensemble model and KNN base learner.

Table 12c Institutional data set results in ensemble model for naive bayes base learner.

Results for Naive Bayes Base Learner		
Confusion Matrix		
	0	1
0	10084	229
1	222	9465
F1 Score		
0.9767297869047006		
ROC AUC Score		
0.9774388520689834		

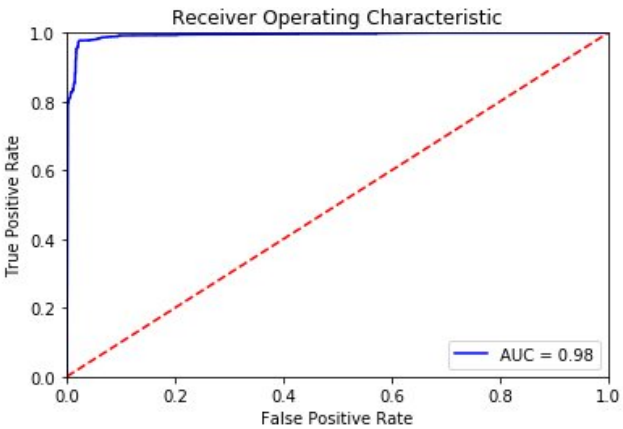

Figure 21c ROC and AUC graphical representation for institutional data set in ensemble model and naive bayes base learner.

Table 12d Institutional data set results in ensemble model for logistic regression base learner.

Results for Logistic Regression Base Learner		
Confusion Matrix		
	0	1
0	10046	232
1	134	9588
F1 Score		
0.981271108381465		
ROC AUC Score		
0.981822171446981		

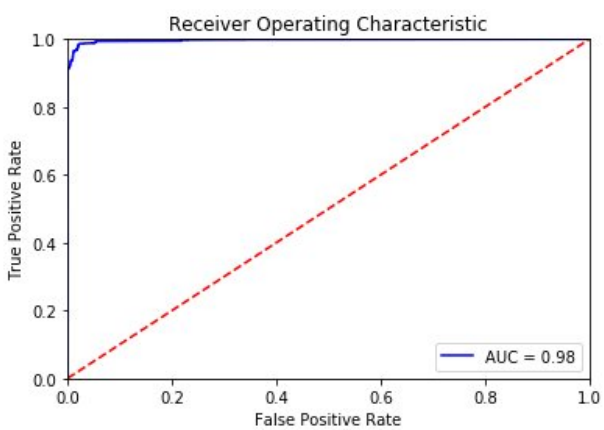

Figure 21d ROC and AUC graphical representation for institutional data set in ensemble model and logistic regression base learner.

Then, UNSW-NB15 data set results with application of these four base learners with bagging ensemble learning method are presented. Table 13a, Table 13b, Table 13c and Table 13d demonstrate confusion matrix, F1 and ROC AUC scores for SVM, KNN, naive bayes and logistic regression base learners respectively. Figure 22a, Figure 22b, Figure 22c and Figure 22d illustrate ROC AUC for models built with SVM, KNN, naive bayes and logistic regression base learners respectively.

Table 13a UNSW-NB15 data set results in ensemble model for SVM base learner.

Results for Support Vector Machines (SVM) Base Learner		
Confusion Matrix		
	0	1
0	17799	209
1	33	1959
F1 Score		
0.9418269230769232		
ROC AUC Score		
0.9859138910149707		

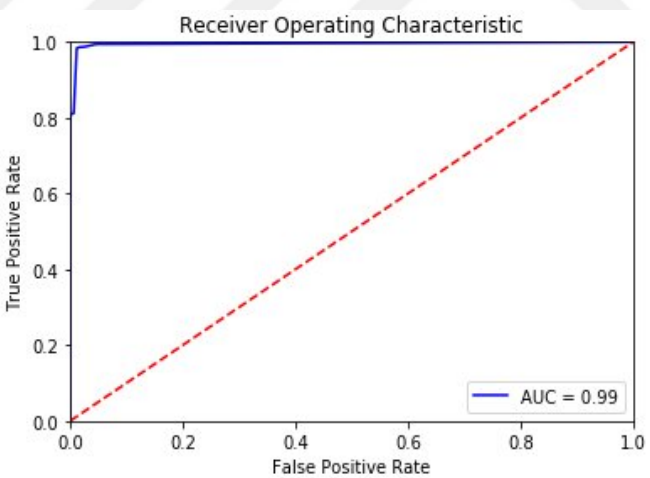

Figure 22a ROC and AUC graphical representation for UNSW-NB15 data set in ensemble model and SVM base learner.

Table 13b UNSW-NB15 data set results in ensemble model for KNN base learner.

Results for K Nearest Neighbours (KNN) Base Learner		
Confusion Matrix		
	0	1
0	17927	90
1	78	1905
F1 Score		
0.95776772247360484		
ROC AUC Score		
0.97793518793001755		

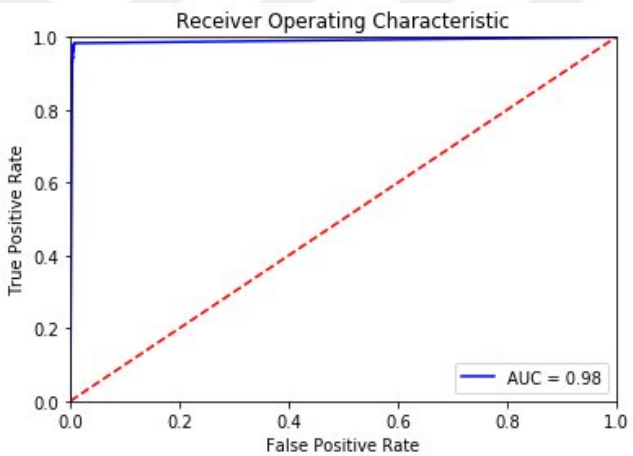

Figure 22b ROC and AUC graphical representation for UNSW-NB15 data set in ensemble model and KNN base learner.

Table 13c UNSW-NB15 data set results in ensemble model for naive bayes base learner.

Results for Naive Bayes Base Learner		
Confusion Matrix		
	0	1
0	17690	327
1	16	1967
F1 Score		
0.91980360065446646		
ROC AUC Score		
0.98689094579834682		

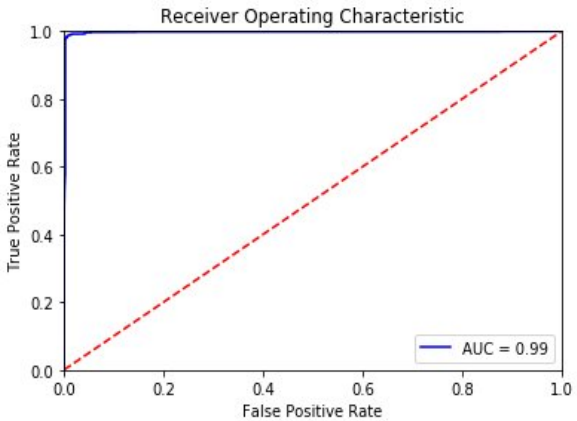

Figure 22c ROC and AUC graphical representation for UNSW-NB15 data set in ensemble model and naive bayes base learner.

Table 13d UNSW-NB15 data set results in ensemble model for logistic regression base learner.

Results for Logistic Regression Base Learner		
Confusion Matrix		
	0	1
0	17872	145
1	25	1958
F1 Score		
0.9583945178658835		
ROC AUC Score		
0.9896724422115931		

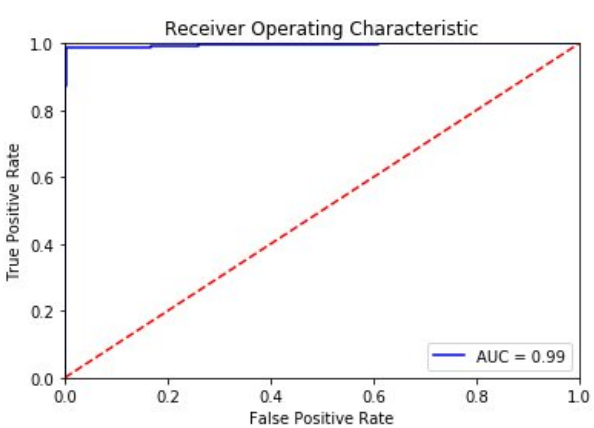

Figure 22d ROC and AUC graphical representation for UNSW-NB15 data set in ensemble model and logistic regression base learner.

Summary of the ensemble learning model's results for both data sets is presented in Table 14. The results from both evaluation metrics can be seen as similar for both data sets. However, there are some points worth emphasizing. First, F1 scores are higher in institutional data set than the UNSW-NB15 data set. Second, apart from KNN classifier, ROC AUC scores from UNSW-NB15 data set are slightly higher than the institutional data set. Third, F1 scores in UNSW-NB15 data set are lower than the ROC AUC scores from the same data set compared with the results with institutional data set.

Table 14 Summary of results for both data sets with ensemble learning model.

Data Set	Base Learner	F1 Score	ROC AUC Score
Institutional	SVM	0.9828	0.9832
	KNN	0.9859	0.9862
	Naive Bayes	0.9767	0.9774
	Logistic Regression	0.9813	0.9818
UNSW-NB15	SVM	0.9418	0.9859
	KNN	0.9578	0.9778
	Naive Bayes	0.9198	0.9869
	Logistic Regression	0.9584	0.9897

Second, convolutional neural network model's performance based on predefined evaluation criteria was demonstrated on institutional data set. Apart from the previous ensemble model, model loss and accuracy metrics were given as another indicators of performance by epochs on which the model was trained and tested. The model was trained and tested for 5 epochs to obtain maximum accuracy and minimum loss. Table 15 demonstrates confusion matrix, F1 and ROC AUC scores for CNN classifier. Figure 23a, Figure 23b and Figure 23c illustrate model's loss, model's accuracy and ROC AUC respectively.

Table 15 Institutional data set results for CNN model.

Institutional Data Set Results for CNN Classifier		
Confusion Matrix		
	0	1
0	10335	27
1	44	9594
F1 Score		
0.9963134119113141		
ROC AUC Score		
0.9998327458234368		

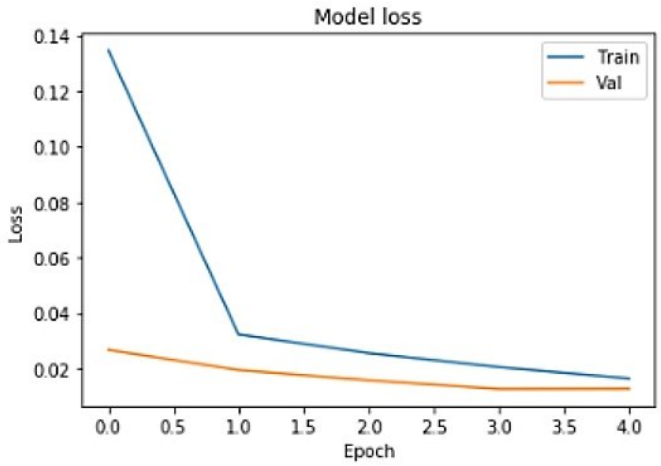

Figure 23a Institutional data set model loss by epochs for CNN model.

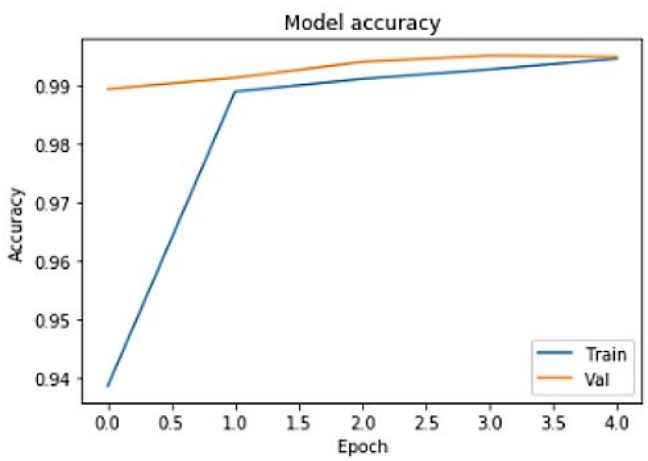

Figure 23b Institutional data set model accuracy by epochs for CNN model.

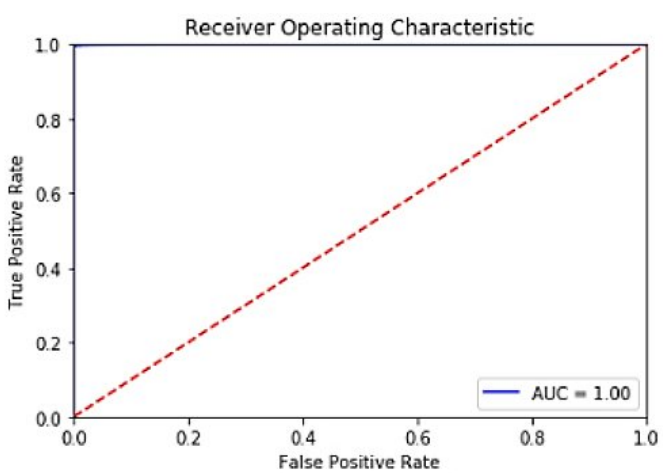

Figure 23c ROC AUC graphical representation for institutional data set with CNN model.

After that, results for UNSW-NB15 data set with CNN model is presented. Table 16 demonstrates confusion matrix, F1 and ROC AUC scores for CNN classifier. Figure 24a, Figure 24b and Figure 24c illustrates model's loss, model's accuracy and ROC AUC respectively. Different from the institutional data set, the model was trained and tested for 7 epochs to reach a convergence between train and test splits' accuracy and loss results.

Table 16 UNSW-NB15 data set results for CNN model.

UNSW-NB15 Data Set Results for CNN Classifier		
Confusion Matrix		
	0	1
0	17923	125
1	57	1895
F1 Score		
0.9902316501821967		
ROC AUC Score		
0.9990000055634883		

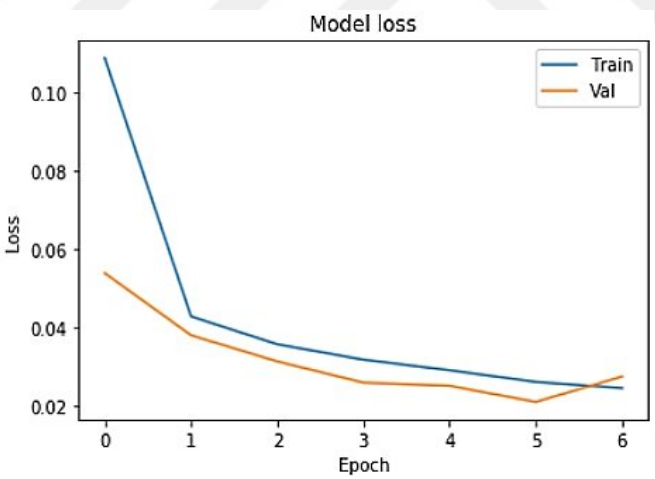

Figure 24a UNSW-NB15 data set model loss by epochs for CNN model.

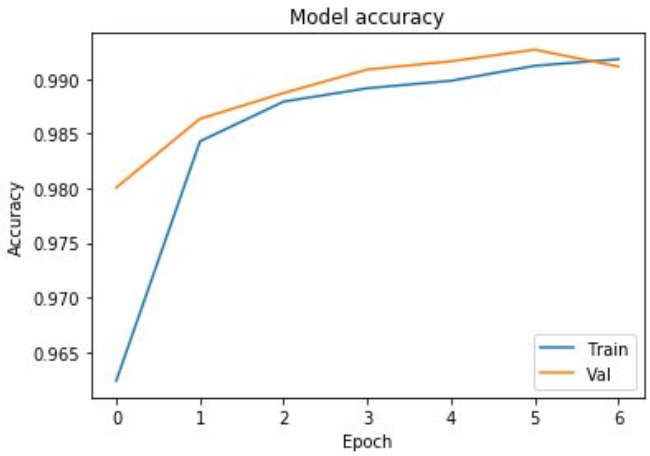

Figure 24b UNSW-NB15 data set model accuracy by epochs for CNN model.

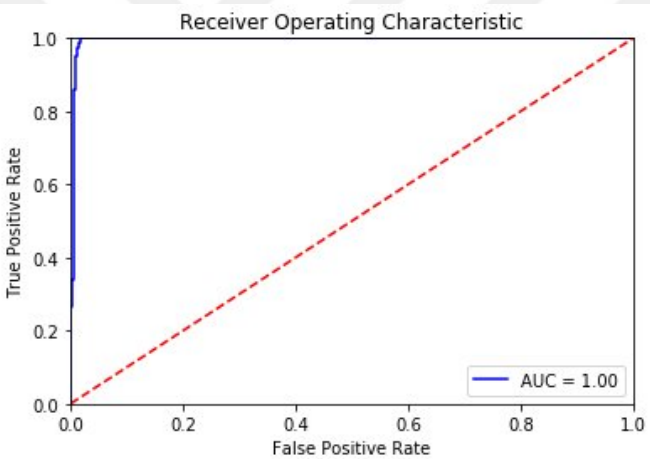

Figure 24c ROC and AUC graphical representation for UNSW-NB15 data set with CNN model.

Summary of the CNN model's results for both data sets' test split is presented in Table 17. They demonstrate that both performance evaluation metrics are higher than 99% for both data sets. Scores from the institutional data set is slightly higher than UNSW-NB15 data set results in both F1 and ROC AUC. Although they are not in this study's evaluation criteria, minimum loss and maximum accuracy scores from the institutional data set are slightly higher indicating the same direction.

Table 17 Summary results of test data for both data sets with CNN model.

Data Set	Min Loss	Max Accuracy	F1 Score	ROC AUC Score
Institutional	0.0126	0.9952	0.9963	0.9998
UNSW-NB15	0.0211	0.9927	0.9902	0.9990

CHAPTER 5

DISCUSSION

This study has the objective of building an anomaly-based ML model to detect network intrusions. In the previous chapter, results from both models and data sets were presented separately for comparative evaluation of each data set in each model. In this chapter, first, a more complete picture will be given with comprehensive results containing both models and data sets. Then, alternative models and methodology in building current models will be discussed. After that, the results from recent studies on UNSW-NB15 benchmarking data set will be evaluated.

Overall results for both data sets and models are presented below in Table 18 according to the evaluation criteria of F1 and ROC AUC scores. There are three significant points to deduce from the table. First, CNN model demonstrates higher scores for both data sets and evaluation metrics. Therefore, it can be argued from the experimental results that CNN model is more successful in detecting network intrusions than ensemble models. Second, for both models, F1 scores from the institutional data set are slightly higher than UNSW-NB15 benchmarking data set. However, the difference between two data sets is not significant with the maximum of 5,69% in ensemble model and 0.08% in CNN model. Third, ROC AUC scores are similar for both data sets and models. They are slightly higher in UNSW-NB15 data set in three models of ensemble classifier. On the other hand, institutional data set's ROC AUC scores are higher in CNN model and one model of ensemble classifier. Resulting from the second and third points, it could be proposed that both ML models designed for anomaly-based network intrusion detection did not demonstrate a significant difference between institutional and benchmarking data sets. Therefore, they are valid and reliable.

Another significant point to deduce from Table 18 is the difference in F1 and ROC AUC scores in UNSW-NB15 data set. In institutional data set, this difference was below 1% at the highest. However, in UNSW-NB15 data set, the largest difference was 6.71%. Arguably, this is because of the distributional difference between data sets. Institutional data set has the positive class ratio of 44% whereas UNSW-NB15 data set has it 11%. Although more significant imbalances were presented (1% class 0, 99% class 1 or vice versa), Saito and Rehmsmeier emphasized the potential problems of using ROC AUC score in heavily imbalanced data sets (Saito & Rehmsmeier, 2015). This point assisted to elaborate the difference between F1 and ROC AUC scores of UNSW-NB15 data set. Therefore, F1 score could be seen as a more reliable performance metric for UNSW-NB15 data set.

Table 18 Comparative summary results for both data sets and models.

Data Set	Model	F1 Score	ROC AUC Score
Institutional	Ensemble with SVM	0.9828	0.9832
	Ensemble with KNN	0.9859	0.9862
	Ensemble with Naive Bayes	0.9767	0.9774
	Ensemble with Logistic Regression	0.9813	0.9818
	CNN	0.9963	0.9998
UNSW-NB15	Ensemble with SVM	0.9418	0.9859
	Ensemble with KNN	0.9578	0.9778
	Ensemble with Naive Bayes	0.9198	0.9869
	Ensemble with Logistic Regression	0.9584	0.9897
	CNN	0.9902	0.9990

Although the CNN model proposes better results in an overall evaluation compared with the ensemble model, it could be argued that it has two significant drawbacks. First, classifications from the CNN model cannot be explained. In other words, the output cannot be interpreted to understand the inner working mechanism and decision logic of the model. This issue is being actively researched in the field of artificial intelligence such that the term of "explainable AI" emerged. Although there are slightly different interpretations of it, “explainable” was defined in AI context such that the ability to satisfactorily answer the question of why? (Miller, 2019). In other words, level of clarity in the interpretation of causal chain between the input and output determines the degree of explainability. Explainable AI provides verification and improvement of the system, compliance to legislation and learning from the system (Samek, Thomas, & Klaus-Robert, 2017).

Previous studies emphasize that some ML methods provide a more explainable structure about their inner mechanism while others do not (Arrieta, 2020; Došilović, Mario, & Nikica, 2018). The former category is called "white box" whereas the latter is "black box". As Riberio et al. argued, explainability is better to be evaluated in a scale rather than a binary classification (Ribeiro, Singh, & Guestrin, 2016). Namely, most methods are "grey-box" rather than black or white with some degree of explainability. Methods such as decision trees, logistic regression, KNN and SVM are placed closer to the white-box since it is possible to make inferences about decision mechanism looking at those models. As an example from this study, importance value of the selected features as result of the two step feature selection process enables the reader to make assumptions about the ensemble model. Contrarily, methods such as neural networks are generally labeled as black-box since their logic of work cannot be interpreted (Adadi & Berrada, 2018).

Narrowing down the focus to the field of this study, Vigano and Magazzeni argued that security is one of the critical fields in which less explainable ML models should be chosen with more caution (Viganò & Daniele, 2018). This is because, accountability becomes even more significant in a process in which what is at stake could be the people’s freedom of

information as a result of a mistaken classification. A trade-off between prediction capability of a model and overall degree of its explainability is well-recognized in previous studies (Hacker et al, 2020; Turek, 2017). To avoid legal consequences of employing less explainable models, it was argued that the use of them will be limited in fields such as health and law despite their higher performance. As a countermeasure, there are proposed methods to increase the level of explainability in black-box methods (Guidotti, 2018; Hase & Mohit, 2020). However, utilization and practicality of them are beyond the scope of this study.

Second drawback of employing a CNN model is computational and temporal costs. Table 19 illustrates the hyperparameter optimization and model training times of both models in institutional data set. It could be deducted from the table that CNN model requires significantly more time than ensemble models. The best resulting ensemble model with KNN base classifier needed 16 minutes for the whole process of hyperparameter optimization, training and testing without GPU support. However, CNN model running on the Google Colab jupyter notebook environment with GPU support took 122 minutes for the same work. Although the models proposed in this study are designed as offline due to the limitations stated previously, anomaly-based IDS models ideally work in real-time. Therefore, employing a model with decreased training time would constitute a significant step in converting it to online.

Table 19 Hyperparameter optimization and training/testing duration of both models in institutional data set.

Model	Environment	Hyperparameter Optimization	Model Training and Testing
Ensemble with SVM	Local PC jupyter notebook with CPU	36 minutes	89 seconds
Ensemble with KNN		15 minutes	37 seconds
Ensemble with Naive Bayes		54 seconds	48 seconds
Ensemble with Logistic Regression		4 minutes	33 seconds
CNN	Google Colab with GPU	66 minutes	56 minutes

After demonstrating the results according to the success criteria and underlining the drawbacks of the CNN model, it could be argued that using ensemble model with KNN base classifier is a better choice in an overall evaluation. This is because, ensemble model did slightly worse than CNN model in classification with 0.5% lower score of F1 and 1% lower score of ROC AUC. However, it has more significant advantages of being more explainable and needing less time and computation power to be trained.

There might be criticism to proposed models in this study since there are ubiquitous approaches to reach the same goal in ML. Two alternative ideas from the field will be discussed here pointed one at each model. First, an alteration scenario to feature selection process in ensemble learning model will be presented with its results. Second, with the use of Multi Layer Perceptron (MLP) instead of CNN model, outcome will be compared in terms of success in evaluation criteria.

Significance of the feature selection in building ML models was well-recognized. It was argued that elimination of a relevant feature weakens the model and inclusion of an irrelevant one might deteriorate the prediction capability (Witten et al., 2017). Apart from the prediction performance, computational power requirements and time complexity of the model is affected from feature selection process. While implementing feature selection in this study, two different techniques which were filter and wrapper methods were combined as it was stated before. In the first elimination by filter method, a threshold was set for reducing the large number of features. Features with the minimum correlation of 0.3 with target for the correlation method and the most 25 significant features for the statistical methods were combined. Then, the combined set was filtered by eliminating ones which have more than 0.75 inter-correlation. These three parameters were determined by combining the inspiration from previous studies and trial and error method.

Table 20 demonstrates the results from experiments with different values for these parameters. As it can be seen from the table, filter method's threshold becomes more inclusive as the number of trial increases. There are both filter method-only and combined feature selection results for each threshold. The values of performance metrics are averaged from five random train-test sampling for each trial. There are four points to infer from these results. First, number of the selected features significantly increases as the threshold becomes more inclusive. First trial selected sixteen features whereas tenth selected thirty nine which corresponds to an increase of two and a half times. Second, time spent on hyperparameter optimization and training/testing significantly raises as the threshold becomes more inclusive and the selected number features increase. This is not a desired outcome since an ideal model for anomaly-based IDS works in real time. Third, there were not a significant improvement in the performance metrics as the thresholds becomes more inclusive and more features are included in the models. Fourth, with the same threshold, applying a combined feature selection method instead of a filter method-only one brings a minor improvement to performance metrics. However, it provides a remarkable time reduction. Resulting from these points, it can be argued that keeping filter method's threshold as in this study's proposed values gives optimized results and employing a combined feature selection methodology provides an advantage with less process time with a slightly better performance.

Second, with the proliferation of neural network usage in the field of anomaly-based IDS, related research proposed results comparing the performances of different neural network models. The most popular choices are multi layer perceptrons, convolutional neural networks, long short term memory networks, recurrent neural networks and hybrid models which were built by combining at least two of them. CNN model was chosen in this study since it was seen displaying satisfactory performance in previous studies compared to other neural network models (Kravchik & Shabtai, 2018; Mohammadpour, Ling, Liew, & Chong, 2018; Naseer & Saleem, 2018). On the other hand, there are some previous work on which results demonstrated no significant performance difference between CNN and the preliminary neural network models such as multi layer perceptrons (MLP) (Kwon et al., 2018). Providing a comprehensive comparison of all popular neural network models' performance is beyond the scope of this study. However, the CNN model will be compared with MLP on the institutional data set in terms of their performance according to the evaluation criteria.

Table 20 Ensemble learning model results with different feature selection threshold and methods

Number of Trial	Min Correlation with Target	Number of Best Features	Max Intercor relation	Methods Used	Final Feature Count	Time Spent on Hyperparameter Optimization	Time Spent on Training/Testing	F1 Score	ROC AUC Score
1	0.35	20	0.7	Filter	16	16 minutes	37 seconds	0.9805	0.9811
2	0.35	20	0.7	Filter+Wrapper	13	15 minutes	37 seconds	0.9859	0.9862
3	0.3	25	0.75	Filter	18	16 minutes	38 seconds	0.9823	0.9831
4	0.3	25	0.75	Filter+Wrapper	13	15 minutes	37 seconds	0.9859	0.9862
5	0.25	30	0.8	Filter	24	28 minutes	44 seconds	0.9816	0.9821
6	0.25	30	0.8	Filter+Wrapper	15	21 minutes	39 seconds	0.9870	0.9876
7	0.2	35	0.85	Filter	30	32 minutes	64 seconds	0.9774	0.9779
8	0.2	35	0.85	Filter+Wrapper	22	28 minutes	44 seconds	0.9801	0.9808
9	0.15	40	0.9	Filter	39	43 minutes	71 seconds	0.9775	0.9779
10	0.15	40	0.9	Filter+Wrapper	23	29 minutes	46 seconds	0.9823	0.9826

To provide a fair comparison, a MLP model was built with two hidden layers. This is because the CNN model proposed in this study had two fully connected layers with similar number of nodes. All 139 features were fed into the model from the input layer and classification of probing/not probing was left to output layer with two nodes. Hyperparameters such as dropout rate, activate function and node count were optimized with hyperopt library as it was done in other models. Figure 25 illustrates the logical representation of the proposed MLP model.

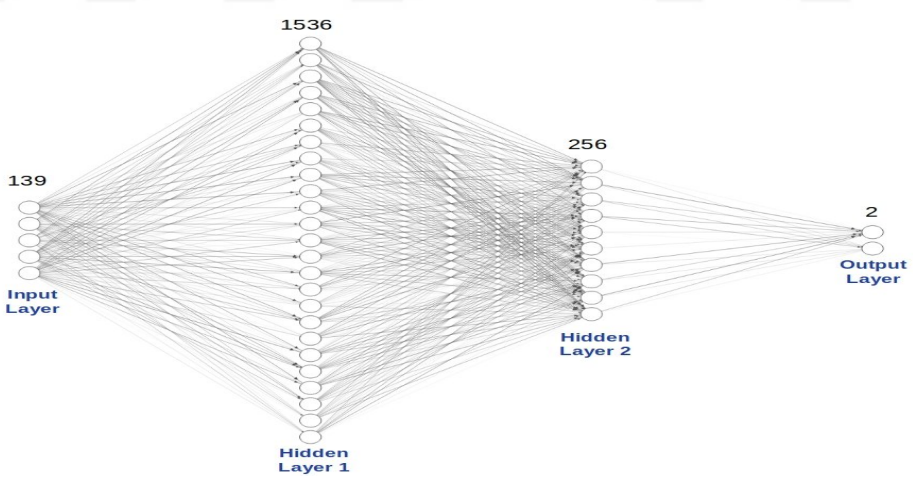

Figure 25 Proposed MLP model to compare with the CNN model of this study.

Results from the MLP model is illustrated in Table 21 and performance comparison with the CNN model is given in Table 22. There are two points to infer from these results. First, MLP model demonstrated a satisfactory performance in terms of the evaluation criteria of this study. Its performance is even higher than the ensemble learning model despite the MLP model also suffers from the similar drawbacks with the CNN model. Second, compared with the CNN model, it performs worse. The difference between two models is not significant

(0.5% in F1 score, 0.15% in ROC AUC score). However, these results are from a single attack type of probing and for a more robust inference on MLP model's performance on anomaly-based IDS field, further research is needed.

Table 21 Institutional data set results with MLP model.

Results for Multi Layer Perceptron (MLP) Model		
Confusion Matrix		
	0	1
0	10272	78
1	95	9555
F1 Score		
0.9910283669553493		
ROC AUC Score		
0.9983608895897474		

Table 22 Results from the MLP and CNN model with institutional data set.

Model	F1 Score	ROC AUC Score
Multi Layer Perceptron	0.9910	0.9983
Convolutional Neural Network	0.9963	0.9998

UNSW-NB15 data set constitutes a modern benchmarking alternative to longstanding DARPA and KDD data sets. Therefore, there have been numerous studies conducted on this data set. To make a comparison and better evaluate the results from this study, Table 20 demonstrates the results from recent studies based on UNSW-NB15 data set. There are two significant points to underline. First, all studies given below trained one multi-class classifier to predict all 10 classes of intrusion attempts and normal traffic. Presented scores belong to reconnaissance type of intrusions for making comparisons. Second, each study utilized different ML models and compared them to get the best predictive model. Presented results are the best ones from the related studies. They suggest that this study's detection performance is better than the best predictive model with 4% and worst predictive model with 15% difference. However, as previously stated, these results are from multi-class classifiers and previous studies demonstrate that it is not possible to obtain a binary classifier's performance with a multi-class classifier in each class' prediction (Sánchez-Maroño, Alonso-Betanzos, García-González, & Bolón-Canedo, 2010; Sokolova & Lapalme, 2009).

Table 23 Results from recent studies based on UNSW-NB15 data set.

No	Study	F1 Score	ROC AUC Score
1	(Khammassi & Krichen, 2017)	0.8287	-
2	(Souhail et. al., 2019)	0.8077	-
3	(Divekar et al., 2018)	0.86	-
4	(Sarıkaya, 2018)	0.82	-
5	(Janarthanan & Zargari, 2017)	0.825	0.977
6	(Gharaee & Hosseinvand, 2017)	0.9312	0.9346
7	(Serkani, Gharaee Garakani, & Mohammadzadeh, 2019)	0.9459	0.9516

Research question of this study in the first chapter of introduction asked whether an anomaly-based ML model for IDS is beneficial with regard to advantages of rule-based (misuse based) intrusion detection methods. The results demonstrated that, in the randomly selected sample of 100,000 flows, 51,338 sessions were permitted by firewall rules and 48,662 were dropped. However, out of those 51,338 permitted sessions, 1,994 were actually probing attacks identified based on labeling methodology presented in the Chapter 3. On the contrary, anomaly-based model classified a vast majority of those sessions. Table 24 and 25 present individual performance of both methods. As Table 26 illustrated, anomaly-based method predicted attack instances better such that its recall score is significantly higher than misuse-based method (96.16% and 98.52% respectively). Similarly, F1 score of anomaly based method is also higher indicating a better overall performance (98.04% and 98.59% respectively). Apart from statistical evaluation, even one detection might mean much in such a critical system. In this setting, nearly five hundred more accurate intrusion detections in 100,000 flows could affect the organization's information security dramatically. Therefore, it could be argued that anomaly-based ML model for IDS is more effective than conventional misuse-based model.

Table 24 Misuse-based method confusion matrix on the institutional data set.

		True Condition	
	Total Population	**Condition Positive**	**Condition Negative**
Predicted Condition	**Predicted Condition Positive**	True Positive 48662	False Positive 0
	Predicted Condition Negative	False Negative 1944	True Negative 49344

Table 25 Anomaly-based method (ensemble model with KNN base classifier) confusion matrix on the institutional data set.

		True Condition	
	Total Population	**Condition Positive**	**Condition Negative**
Predicted Condition	**Predicted Condition Positive**	True Positive 10133	False Positive 137
	Predicted Condition Negative	False Negative 152	True Negative 9578

Table 26 Anomaly-based misuse-based performance comparison for institutional data set.

Status	Precision	Recall	F1 Score
Misuse-Based (Rule-Based)	1.0000	0.9616	0.9804
Anomaly Based ML Model	0.9867	0.9852	0.9859

CHAPTER 6

CONCLUSION

This study had the goal of building an anomaly-based ML model for IDS. An institutional data set was utilized to reveal whether a model could be designed that achieve two things which were proposed in the first chapter of introduction. First, the designed model needed to be valid. To ensure its validity, two ML models were created and after implementing on institutional data set, UNSW-NB15 benchmarking data set was used to compare results. The evaluation of two result sets demonstrated that the methodology and both models are valid. And second, anomaly-based model needed to have significantly better classification rate to leverage on misuse-based models. The results suggested that the proposed model provides both better recall and F1 scores with approximately 500 more accurate classifications in 100,000 sessions only in the chosen attack type of probing.

Two ML models were trained to compare their performance and the CNN model did slightly better than the ensemble models on the institutional data set. However, ensemble model became the model of choice because it did not suffer from two critical issues. First, the ensemble model is more “explainable” and security is a sensitive field in which transparency and accountability is extensively needed. Second, the ensemble model required significantly less time to be trained than the CNN model. Although the proposed model was designed offline, it is more convenient to convert ensemble model to online as a future work.

To conclude, in the never ending struggle between intruders and information security professionals, the outcome of this study might contribute the provision of information security’s famous triangle of confidentiality, integrity and availability (CIA).

6.1. Future Work

As to enumerate the things to be done, at least three fields need improvement. First, the proposed pipeline could be designed to be working online. This is because of vast rate of the incoming traffic and the need to process and classify the sessions as normal or not in real time. Evolving the model to an online one means more optimization in the pipeline since the training duration must decrease dramatically. However, even in the current version, training could be done in certain periods asynchronously and the trained model could be placed on the security hardware as a new layer in defense-in-depth concept.

Second, since this study proposed a binary classifier to distinguish flows as probing attack or not, a solution to the other attack types are needed to be developed. This could be a multi-class classifier to label different categories or multiple one-vs-all binary classifiers that run in parallel to identify each class of intrusion attempt separately. The first option needs a sophisticated classifier with the capability to distinguish among myriad types of modern

attacks but requires moderate computational power as there would be only one model for traffic to go through. On the other hand, second option promises higher detection rates with simpler models but needs more computational power to flow traffic through multiple models in real time without causing latency.

Third, to overcome one significant drawback of the CNN model, its level of explainability could be increased. The field of “explainable AI” is emerging and there are proposed methods to discover inner mechanisms of black-box models. After achieving a degree of interpretability for the CNN model, its higher detection rate could be used instead of the ensemble model.

6.2. Limitations

There are some limitations which needed to be acknowledged. First, institutional data set was gathered from a production environment. Therefore, it was limited to the attack types related to the services provided by the organization. Another real-world data set might present significantly different patterns. Second, there were three tools for feature and session extraction from raw pcap files. Since tcptrace could only extract TCP sessions’ features but other two utilize other protocols such as UDP or ICMP, the amount of missing values were large for some features. Although the beforementioned missing value handling techniques were implemented, this could have introduced bias to the model to some extent. And third, supervised learning model was chosen and labeling was done using the three sources underlined in the methodology chapter. Regardless of the scrutiny on its implementation, labeling is prone to error as it was emphasized in previous studies as a common pitfall in supervised learning in IDS models.

REFERENCES

Adadi, A., & Berrada, M. (2018). Peeking inside the black-box: A survey on Explainable Artificial Intelligence (XAI). *IEEE Access*, *6*, 52138–52160.

Alkasassbeh, M. (2017). An empirical evaluation for the intrusion detection features based on machine learning and feature selection methods. *Journal of Theoretical and Applied Information Technology*, *95*(22), 5962–5976.

Ambusaidi, M. A., He, X., & Nanda, P. (2015). Unsupervised feature selection method for intrusion detection system. *2015 IEEE Trustcom/BigDataSE/ISPA*, *1*, 295–301. IEEE.

Ambusaidi, M. A., He, X., Nanda, P., & Tan, Z. (2016). Building an intrusion detection system using a filter-based feature selection algorithm. *IEEE Transactions on Computers*, *65*(10), 2986–2998.

Ambusaidi, M. A., He, X., Tan, Z., Nanda, P., Lu, L. F., & Nagar, U. T. (2015). A novel feature selection approach for intrusion detection data classification. *Proceedings - 2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2014*, 82–89.

Amin, A., Anwar, S., Adnan, A., Khan, M. A., & Iqbal, Z. (2015). Classification of cyber attacks based on rough set theory. *2015 1st International Conference on Anti-Cybercrime, ICACC 2015*, 1–6. IEEE.

Argus. (2020). openargus - Documentation. Retrieved May 24, 2020, from https://openargus.org/documentation

Arrieta, A. B. et al. (2020). Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion*, *58*, 82–115.

Axelsson, S. (2000). *Intrusion Detection Systems: A Survey and Taxonomy*.

Berger, J. (2013). *Statistical decision theory and Bayesian analysis.* Springer Science & Business Media.

Bergstra, J., Yamins, D., & Cox, D. D. (2013). Hyperopt: A Python Library for Optimizing the Hyperparameters of Machine Learning Algorithms. In *Proceedings of the 12th Python in science conference* (Vol. 12).

Bhattacharyya, D. K., & Kalita, J. K. (2013). *Network Anomaly Detection: A Machine Learning Perspective*. Crc Press.

Bolón-Canedo, V., Sánchez-Maroño, N., & Alonso-Betanzos, A. (2016). Feature selection for high-dimensional data. *Progress in Artificial Intelligence*, *5*(2), 65–75.

Borji, A. (2007). Combining heterogeneous classifiers for network intrusion detection. *Annual Asian Computing Science Conference*, *4846 LNCS*, 254–260. Springer.

Bowker, G. C. (2019). The Computerization Time Crunch. *Computer*, *52*(7), 67–71.

Brownlee, J. (2014). Classification Accuracy is Not Enough: More Performance Measures You Can Use. Retrieved June 16, 2020, from https://machinelearningmastery.com/classification-accuracy-is-not-enough-more-performance-measures-you-can-use/

Buczak, A. L., & Guven, E. (2016). A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection. *IEEE Communications Surveys and Tutorials*, *18*(2), 1153–1176.

Calzolari, M. (2019). manuel-calzolari/sklearn-genetic. Retrieved January 2, 2020, from https://github.com/manuel-calzolari/sklearn-genetic

Casas, P. (2019). Feature Selection using Genetic Algorithms in R. Retrieved January 2, 2020, from https://towardsdatascience.com/feature-selection-using-genetic-algorithms-in-r-3d9252f1aa66

Chitrakar, R., & Chuanhe, H. (2012). Anomaly detection using Support Vector Machine classification with k-Medoids clustering. *Asian Himalayas International Conference on Internet*, 1–5. IEEE.

Crichigno, J., Bou-Harb, E., & Ghani, N. (2019). A Comprehensive Tutorial on Science DMZ. *IEEE Communications Surveys and Tutorials*, *21*(2), 2041–2078.

Didaci, L., Giacinto, G., & Roli, F. (2002). Ensemble learning for intrusion detection in computer networks. *Workshop Machine Learning Methods Applications*.

Dietterich, T. (1995). Overfitting and Undercomputing in Machine Learning. *ACM Computing Surveys (CSUR)*, *27*(3), 326–327.

Divekar, A., Parekh, M., Savla, V., Mishra, R., & Shirole, M. (2018). Benchmarking datasets for Anomaly-based Network Intrusion Detection: KDD CUP 99 alternatives. *Proceedings on 2018 IEEE 3rd International Conference on Computing, Communication and Security, ICCCS 2018*, 1–8. IEEE.

Došilović, F. K., Mario, B., & Nikica, H. (2018). Explainable artificial intelligence: A survey. *International Convention on Information and Communication Technology, Electronics and Microelectronics*, 210–215.

Estévez-Tapiador, J. M., García-Teodoro, P., & Díaz-Verdejo, J. E. (2005). Detection of web-based attacks through Markovian protocol parsing. *Proceedings - IEEE Symposium on Computers and Communications*, 457–462.

Gaikwad, D. P., & Thool, R. C. (2015). Intrusion detection system using bagging ensemble method of machine learning. *International Conference on Computing Communication Control and Automation*, 291–295. IEEE.

Ganapathy, S., Kulothungan, K., Muthurajkumar, S., Vijayalakshmi, M., Yogesh, L., & Kannan, A. (2013). Intelligent feature selection and classification techniques for intrusion detection in networks: A survey. *Eurasip Journal on Wireless Communications and Networking*, *2013*(1), 1–16.

Garcia, S. (2020). Stratosphere IPS Project. Retrieved March 21, 2020, from https://www.stratosphereips.org/datasets-overview

Gharaee, H., & Hosseinvand, H. (2017). A new feature selection IDS based on genetic algorithm and SVM. *2016 8th International Symposium on Telecommunications, IST 2016*, 139–144. IEEE.

Gil, P. (2019). Cleaning Big Data: Most Time-Consuming, Least Enjoyable Data Science Task, Survey Says. Retrieved December 29, 2019, from https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/#5877a22d6f63

Goodrich, M., & Tamassia, R. (2011). *Introduction to Computer Security*. Pearson Education Limited.

Guidotti, R. et al. (2018). A survey of methods for explaining black box models. *ACM Computing Surveys (CSUR)*, *51*(5), 1–42.

Hacker, P., & et al. (2020). Explainable AI under contract and tort law: legal incentives and technical challenges. *Artificial Intelligence and Law*, 1–25.

Hamamoto, A. H., Carvalho, L. F., Sampaio, L. D. H., Abrão, T., & Proença, M. L. (2018). Network Anomaly Detection System using Genetic Algorithm and Fuzzy Logic. *Expert Systems with Applications*, *92*, 390–402.

Harbi, N., Singh, D., & Zahidur Rahman, M. (2010). Combining Naive Bayes and Decision Tree for Adaptive Intrusion Detection. *International Journal of Network Security & Its Applications*, *2*(2), 12–25.

Hase, P., & Mohit, B. (2020). *Evaluating Explainable AI: Which Algorithmic Explanations Help Users Predict Model Behavior?*

Heaton, J. (2008). The Number of Hidden Layers. In *Introduction to Neural Networks for Java* (2nd ed., pp. 157--158). Heaton Research, Inc.

Huilgol, P. (2019). Accuracy vs. F1-Score - Analytics Vidhya - Medium. Retrieved June 16, 2020, from https://medium.com/analytics-vidhya/accuracy-vs-f1-score-6258237beca2

Hurwitz, J. (2018). Diagnosing Computerization: The Effects of Technical Change on Professions/Fields. *Academy of Management Proceedings*, *2018*(1), 15005.

Iglesias, F., & Zseby, T. (2015). Analysis of network traffic features for anomaly detection. *Machine Learning*, *101*(1–3), 59–84.

Janarthanan, T., & Zargari, S. (2017). Feature selection in UNSW-NB15 and KDDCUP'99 datasets. *IEEE International Symposium on Industrial Electronics*, 1881–1886. IEEE.

Khammassi, C., & Krichen, S. (2017). A GA-LR wrapper approach for feature selection in network intrusion detection. *Computers and Security*, *70*, 255–277. https://doi.org/10.1016/j.cose.2017.06.005

Kravchik, M., & Shabtai, A. (2018). Detecting cyber attacks in industrial control systems using convolutional neural networks. *Proceedings of the ACM Conference on Computer and Communications Security*, 72–83.

Kwon, D., Natarajan, K., Suh, S. C., Kim, H., & Kim, J. (2018). An empirical study on network anomaly detection using convolutional neural networks. *Proceedings - International Conference on Distributed Computing Systems*, *2018-July*(October), 1595–1598.

Liao, H. J., Richard Lin, C. H., Lin, Y. C., & Tung, K. Y. (2013). Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, *36*(1), 16–24.

Lockheed Martin. (2020). Cyber kill chain. Retrieved May 29, 2020, from https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html

Mallawaarachchi, V. (2017). Introduction to Genetic Algorithms. Retrieved January 2, 2020, from https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3

Miller, T. (2019). Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence*, *267*, 1–39.

MIT Lincoln Laboratory, D. (1998). MIT Lincoln Laboratory, DARPA Intrusion Detection Evaluation Data Sets. Retrieved March 21, 2020, from https://www.ll.mit.edu/r-d/datasets/1998-darpa-intrusion-detection-evaluation-dataset

Mitchell, T. M. (1997). *Machine learning.* New Delhi: McGraw Hill Education.

Moh, K., Aung, M., & Oo, N. N. (2015). Association Rule Pattern Mining Approaches Network Anomaly Detection. *Proceedings of 2015 International Conference on Future Computational Technologies*, 164–170.

Mohammadi, S., Mirvaziri, H., Ghazizadeh-Ahsaee, M., & Karimipour, H. (2019). Cyber intrusion detection by combined feature selection algorithm. *Journal of Information Security and Applications*, *44*, 80–88.

Mohammadpour, L., Ling, T. C., Liew, C. S., & Chong, C. Y. (2018). A Convolutional Neural Network for Network Intrusion Detection System. *Proceedings of the Asia-Pacific Advanced Network*, (46), 50–55.

Morgan, S. (2019). 2019 Cybersecurity Almanac: 100 Facts, Figures, Predictions and Statistics. Retrieved January 6, 2020, from https://cybersecurityventures.com/cybersecurity-almanac-2019/

Moustafa, N., & Slay, J. (2015). UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). *2015 Military Communications and Information Systems Conference, MilCIS 2015 - Proceedings*. Institute of Electrical and Electronics Engineers Inc.

Moustafa, N., & Slay, J. (2016). The evaluation of Network Anomaly Detection Systems: Statistical analysis of the UNSW-NB15 data set and the comparison with the KDD99 data set. *Information Security Journal*, *25*(1–3), 18–31.

Naseer, S., & Saleem, Y. (2018). Enhanced network intrusion detection using deep convolutional neural networks. *KSII Transactions on Internet and Information Systems*, *12*(10), 5159–5178.

Nguyen, H. T., Franke, K., & Petrović, S. (2012). Feature extraction methods for intrusion detection systems. *Threats, Countermeasures, and Advances in Applied Information Security*, *3–3*, 23–52.

Ostermann, S. (2020). tcptrace - Official Homepage. Retrieved May 24, 2020, from http://www.tcptrace.org/manual.html

Panda, M., & Patra, M. R. (2009). Ensemble Voting System for Anomaly Based Network Intrusion Detection. *International Journal of Recent Trends in Engineering*, *2*(5), 1–8.

Peddabachigari, S., Abraham, A., Grosan, C., & Thomas, J. (2007). Modeling intrusion detection system using hybrid intelligent systems. *Journal of Network and Computer Applications*, *30*(1), 114–132.

Pham, N. T., Foo, E., Suriadi, S., Jeffrey, H., & Lahza, H. F. M. (2018). Improving performance of intrusion detection system using ensemble methods and feature selection. *ACM International Conference Proceeding Series*, 1–6. Association for Computing Machinery.

Powers, D. M. W. (2011). Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness and Correlation. *Journal of Machine Learning Technologies*, *2*(1), 37–63.

Reeker, L. (2001). Theoretical Constructs for Measurement Performance and Intelligence in Intelligent Systems. *NIST Special Publications*, 49–59.

Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). "Why should i trust you?" Explaining the predictions of any classifier. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1135–1144.

Robbins, C. (2019). *2018 annual evaluation by Cisco CEO*.

Ruizendaal, R. (2017). Deep Learning: More on CNNs & Handling Overfitting. Retrieved March 2, 2020, from https://towardsdatascience.com/deep-learning-3-more-on-cnns-handling-overfitting-2bd5d99abe5d

Russel, S., & Norvig, P. (2016). *Artificial Intelligence: A Modern Approach, Global Edition*. Pearson Education Limited.

Saito, T., & Rehmsmeier, M. (2015). The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets. *PLoS ONE*, *10*(3).

Samek, W., Thomas, W., & Klaus-Robert, M. (2017). *Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models.*

Samuel, A. L. (1959). Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development*, *44*(1.2), 210–229.

Sánchez-Maroño, N., Alonso-Betanzos, A., García-González, P., & Bolón-Canedo, V. (2010). Multiclass classifiers vs multiple binary classifiers using filters for feature selection. *Proceedings of the International Joint Conference on Neural Networks*, 1–8. IEEE.

Sarıkaya, A. (2018). *Anomaly-based Cyber Intrusion Detection System With Ensemble Classifier*. Middle East Technical University.

Schaffer, C. (1993). Overfitting avoidance as bias. *Machine Learning*, *10*(2), 153–178.

Serkani, E., Gharaee Garakani, H., & Mohammadzadeh, N. (2019). Anomaly Detection Using SVM as Classifier and Decision Tree for Optimizing Feature Vectors. *Iranian Society of Cryptology*, *11*(2), 159–171.

Sfakianakis, A., Douligeris, C., & Marinos, L. (2019). *ENISA Threat Landscape Report 2018: 15 Top Cyberthreats and Trends*.

Sharafaldin, I., Lashkari, A. H., & Ghorbani, A. A. (2019). Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. *ICISSP*, 108–116.

Shung, K. P. (2018). Accuracy, Precision, Recall or F1? - Towards Data Science. Retrieved June 16, 2020, from https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9

Singh, D., & Singh, V. P. (2012). Comparative Study of Various Distributed Intrusion Detection Systems for WLAN. *Global Journal of Researches in Engineering Electrical and Electronics Engineering*, *12*(6).

Smoltakov, V. (2017). Ensemble Learning to Improve Machine Learning Results. Retrieved January 3, 2020, from https://blog.statsbot.co/ensemble-learning-d1dcd548e936

Sokolova, M., & Lapalme, G. (2009). A systematic analysis of performance measures for classification tasks. *Information Processing and Management*, *45*(4), 427–437.

Souhail et. al., M. (2019). Network Based Intrusion Detection Using the UNSW-NB15 Dataset. *International Journal of Computing and Digital Systems*, *8*(5), 477–487.

Stallings, W., & Brown, L. (2017). *Computer security : principles and practice*. Upper Saddle River, NJ, USA: Pearson Education.

Stolfo, S., Fan, W., Lee, W., & Prodromidis, A. (1999). Kdd cup 1999 data. Retrieved March 21, 2020, from https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

Syarif, I., Prugel-Bennett, A., & Wills, G. (2012). Unsupervised Clustering Approach for Network Anomaly Detection. In *International conference on networked digital technologies* (pp. 135–145). Berlin: Springer International Publishing.

Tavallaee, M., Bagheri, E., Lu, W., & Ghorbani, A. A. (2009). A detailed analysis of the KDD CUP 99 data set. *IEEE Symposium on Computational Intelligence for Security and Defense Applications, CISDA 2009*, 1–6. IEEE.

Tsai, J. J. P., & Yu, P. S. (2009). Machine learning in cyber trust: Security, privacy, and reliability. In *Machine Learning in Cyber Trust: Security, Privacy, and Reliability*. Springer Science & Business Media.

Turek, M. (2017). *Explainable Artificial Intelligence*. Retrieved from https://www.darpa.mil/program/explainable-artificial-intelligence

Vanerio, J., & Casas, P. (2017). Ensemble-learning approaches for network security and anomaly detection. *Proceedings of the Workshop on Big Data Analytics and Machine Learning for Data Communication Networks*, 1–6.

Viganò, L., & Daniele, M. (2018). *Explainable security*.

Witten, I. H., Frank, E., Hall, M. A., & Pal, C. J. (2017). *Data Mining Practical Machine Learning Tools and Techniques Fourth Edition*.

Zhang, J., & Zulkernine, M. (2006). Anomaly based network intrusion detection with unsupervised outlier detection. *IEEE International Conference on Communications*, *5*, 2388–2393.

Zhu, M., Ye, K., Wang, Y., & Xu, C. Z. (2018). A deep learning approach for network anomaly detection based on AMF-LSTM. *IFIP International Conference on Network and Parallel Computing, 11276 LNCS*, 137–141. Springer.

APPENDICES

APPENDIX A

UNSW-NB15 DATA SET FEATURES

No.	Name	Description
1	srcip	Source IP address
2	sport	Source port number
3	dstip	Destination IP address
4	dsport	Destination port number
5	proto	Transaction protocol
6	state	Indicates to the state and its dependent protocol, e.g. ACC, CLO, CON, ECO, ECR, FIN, INT, MAS, PAR, REQ, RST, TST, TXD, URH, URN, and (-) (if not used state)
7	dur	Record total duration
8	sbytes	Source to destination transaction bytes
9	dbytes	Destination to source transaction bytes
10	sttl	Source to destination time to live value
11	dttl	Destination to source time to live value
12	sloss	Source packets retransmitted or dropped
13	dloss	Destination packets retransmitted or dropped
14	service	http, ftp, smtp, ssh, dns, ftp-data ,irc and (-) if not much used service
15	Sload	Source bits per second
16	Dload	Destination bits per second
17	Spkts	Source to destination packet count
18	Dpkt	Destination to source packet count
19	swin	Source TCP window advertisement value
20	dwin	Destination TCP window advertisement value
21	stcpb	Source TCP base sequence number
22	dtcpb	Destination TCP base sequence number
23	smeansz	Mean of the raw packet size transmitted by the src
24	dmeansz	Mean of the raw packet size transmitted by the dst
25	trans_depth	Represents the pipelined depth into the connection of http request/response transaction

26	res_bdy_len	Actual uncompressed content size of the data transferred from the server's http service.
27	Sjit	Source jitter (mSec)
28	Djit	Destination jitter (mSec)
29	Stime	record start time
30	Ltime	record last time
31	Sintpkt	Source interpacket arrival time (mSec)
32	Dintpkt	Destination interpacket arrival time (mSec)
33	tcprtt	TCP connection setup round-trip time, the sum of synack and ackdata.
34	Synack	TCP connection setup time, the time between the SYN and the SYN_ACK packets.
35	ackdat	TCP connection setup time, the time between the SYN_ACK and the ACK packets.
36	is_sm_ips_ports	If source (1) and destination (3)IP addresses equal and port numbers (2) (4) equal then, this variable takes value 1 else 0
37	ct_state_ttl	No. for each state (6) according to specific range of values for source/destination time to live (10) (11).
38	ct_flw_http_mthd	No. of flows that has methods such as Get and Post in http service.
39	is_ftp_login	If the ftp session is accessed by user and password then 1 else 0.
40	ct_ftp_cmd	No of flows that has a command in ftp session.
41	ct_srv_src	No. of connections that contain the same service (14) and source address (1) in 100 connections according to the last time (26).
42	ct_srv_dst	No. of connections that contain the same service (14) and destination address (3) in 100 connections according to the last time (26).
43	ct_dst_ltm	No. of connections of the same destination address (3) in 100 connections according to the last time (26).
44	ct_src_ltm	No. of connections of the same source address (1) in 100 connections according to the last time (26).
45	ct_src_dport_ltm	No of connections of the same source address (1) and the destination port (4) in 100 connections according to the last time (26).
46	ct_dst_sport_ltm	No of connections of the same destination address (3) and the source port (2) in 100 connections according to the last time (26).
47	ct_dst_src_ltm	No of connections of the same source (1) and the destination (3) address in in 100 connections according to the last time (26).
48	attack_cat	The name of each attack category. In this data set , nine categories e.g. Fuzzers, Analysis, Backdoors, DoS Exploits, Generic, Reconnaissance, Shellcode and Worms

APPENDIX B

PYTHON CODE FOR FEATURE SELECTION

```
# This script is exported from a jupyter notebook environment
# imports

from __future__ import print_function
import sys
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn.feature_selection as skfs
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import RandomForestClassifier
import sklearn.preprocessing as skpr
from sklearn.feature_selection import f_classif
from genetic_selection import GeneticSelectionCV

# display options

np.set_printoptions(threshold=sys.maxsize)
pd.set_option('display.max_rows', 50000)
pd.set_option('display.max_columns', 500)
pd.set_option('display.float_format', lambda x: '%.6f' % x)

# read csv files

X = pd.read_csv('/***/***/***/***/***/***.csv', index_col=0, low_memory=False)
y = pd.read_csv('/***/***/***/***/***/***.csv', index_col=0, low_memory=False)
df_whole = pd.read_csv('/***/***/***/***/***/***.csv', index_col=0, low_memory=False)
X = X.drop(["SrcAddr", "DstAddr"], axis=1)
df_whole = df_whole.drop(["SrcAddr", "DstAddr"], axis=1)

# Feature selection with filter method starts
# 1.Features with low varience (below 0.005) are eliminated

sel = skfs.VarianceThreshold(threshold=(.995 * (1 - .995)))
sel.fit(df_whole)
df_wo_low_variance = df_whole[df_whole.columns[sel.get_support(indices=True)]]
df_wo_low_variance.to_csv(path_or_buf="/***/***/***/***/***/***.csv")

# 2a. Filter method feature selection with correlation > 0.3

target = y["Class"]
df_wo_low_variance = df_wo_low_variance.astype(dtype="float64")
importances = df_wo_low_variance.apply(lambda x: x.corr(target))
corr_importances = importances[abs(importances) > 0.3]
indices = np.argsort(corr_importances)
print(corr_importances[:])
corr_importances.index
```

 # Highly correlated features illustrated

```
features_corr = corr_importances.index
plt.title("features_corr")
plt.barh(range(len(indices)), importances[indices], color="g", align="center")
plt.yticks(range(len(indices)), [features_corr[i] for i in indices])
plt.xlabel("Relative Importance")
plt.show()
```

 # 2b. Filter method feature selection with ensemble learning extra trees

```
model = ExtraTreesClassifier(n_estimators=10000, n_jobs=-1)
model.fit(df_wo_low_variance, y)
print(model.feature_importances_)

df_wo_low_variance_columns = df_wo_low_variance.columns
df_model_feature_importances = pd.DataFrame(model.feature_importances_, index =
df_wo_low_variance_columns, columns=["importance"])
df_decision_forest_selected_columns =
df_model_feature_importances[df_model_feature_importances["importance"].astype(float) > 0.005]
print(df_decision_forest_selected_columns.index)
```

 # 2c. Filter method feature selection with ANOVA f-value method

```
fvalue_selector = skfs.SelectKBest(score_func=f_classif, k=25)
fvalue_selector.fit(df_wo_low_variance, target)
print(fvalue_selector.scores_)

df_wo_low_variance_columns = df_wo_low_variance.columns
df_model_feature_importances = pd.DataFrame(fvalue_selector.scores_,
index=df_wo_low_variance_columns, columns=["score"])
df_f_value_selected_columns = df_model_feature_importances[df_model_feature_importances
["score"].astype(float) > 10000]
print(df_f_value_selected_columns.index)
```

 # 2d. Filter method feature selection with chi2 method

```
min_max_scaler = preprocessing.MinMaxScaler(feature_range=(0, 100))
np_wo_low_variance_minmax = min_max_scaler.fit_transform(df_wo_low_variance)
df_wo_low_variance_minmax = pd.DataFrame(np_wo_low_variance_minmax,
columns=df_wo_low_variance.columns)

chi2_selector = skfs.SelectKBest(skfs.chi2, k=25)
chi2_selector.fit(df_wo_low_variance_minmax, target)
print(chi2_selector.scores_)

df_wo_low_variance_columns = df_wo_low_variance.columns
df_model_feature_scores = pd.DataFrame(chi2_selector.scores_,
index=df_wo_low_variance_columns, columns=["score"])
df_chi2_selector_selected_columns =
df_model_feature_scores[df_model_feature_importances["score"].astype(float) > 10000]
print(df_chi2_selector_selected_columns.index)
```

```
# Results from four different filter methods

print(features_corr)
print(type(features_corr))
print("**********************************************************************")
# print(df_decision_forest_selected_columns)
df_decision_forest_selected_columns_list = df_decision_forest_selected_columns.index.values.tolist()
print(df_decision_forest_selected_columns_list)
print(type(df_decision_forest_selected_columns_list))
print("**********************************************************************")
df_f_value_selected_columns_list = df_f_value_selected_columns.index.values.tolist()
print(df_f_value_selected_columns_list)
print(type(df_f_value_selected_columns_list))
print("**********************************************************************")
df_chi2_selector_selected_columns_list = df_chi2_selector_selected_columns.index.values.tolist()
print(df_chi2_selector_selected_columns_list)
print(type(df_chi2_selector_selected_columns_list))

# Combination of four filter method feature selection result sets

combined_selected_features = set(features_corr + df_decision_forest_selected_columns_list +
df_f_value_selected_columns_list + df_chi2_selector_selected_columns_list)
print(combined_selected_features)

# 3. Elimination of highly inter-correlated (higher than 0.75) features
# Filter method final feature set is ready.

X = df_wo_low_variance[combined_selected_features]

def correlation(dataset, threshold):
    col_corr = set()
    corr_matrix = dataset.corr()
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            if abs(corr_matrix.iloc[i, j]) > threshold:
                colName = corr_matrix.columns[i]
                col_corr.add(colName)
    return col_corr
col = correlation(X, 0.75)

print('Correlated columns:',col)
print(len(col))
filter_method_selected_features = combined_selected_features - col
print("**********************************************************************")
print('Remaining Columns:', filter_method_selected_features)
print(len(filter_method_selected_features))

df_selected_features_filter = df_wo_low_variance[filter_method_selected_features]
df_selected_features_filter.to_csv(path_or_buf="/***/***/***/***/***/
df_selected_features_filter.csv", index=False)
df_selected_features_filter = df_selected_features_filter.to_numpy()
```

```
scaler = skpr.MinMaxScaler(feature_range=(0,1), copy=False)
scaler.fit(df_selected_features_filter)
scaler.transform(df_selected_features_filter)
df_selected_features_filter = pd.DataFrame(df_selected_features_filter,
columns=filter_method_selected_features)
```

 # 4. Final feature selection with the wrapper method of genetic algorithm

```
X = df_selected_features_filter.copy()

estimator = RandomForestClassifier(n_estimators=1000, n_jobs=1)
selector = GeneticSelectionCV(estimator,
                              cv=5,
                              verbose=1,
                              scoring="accuracy",
                              max_features=18,
                              n_population=300,
                              crossover_proba=0.5,
                              mutation_proba=0.2,
                              n_generations=50,
                              crossover_independent_proba=0.1,
                              mutation_independent_proba=0.05,
                              tournament_size=3,
                              n_gen_no_change=10,
                              caching=True,
                              n_jobs=-1)
selector = selector.fit(X, y.values.ravel())

print(selector.support_)
print(X.columns)

X.drop(X.columns[np.where(selector.support_ == False)[0]], axis=1, inplace=True)

dset = pd.DataFrame()
dset['attr'] = X.columns
dset['importance'] = selector.estimator_.feature_importances_
dset = dset.sort_values(by='importance', ascending=False)

plt.figure(figsize=(16, 14))
plt.barh(y=dset['attr'], width=dset['importance'], color='#1976D2')
plt.title('sklearn.GeneticSelectionCV - Feature Importances', fontsize=20, fontweight='bold', pad=20)
plt.xlabel('Importance', fontsize=14, labelpad=20)
plt.show()

sklearn_GeneticSelector_selected_columns = X.columns
print(sklearn_GeneticSelector_selected_columns, len(sklearn_GeneticSelector_selected_columns))
```

*** Note that * symbols were placed where required to mask personal and institutional information.**

APPENDIX C

PYTHON CODE FOR ENSEMBLE ML MODEL

```
# This script is exported from a jupyter notebook environment
# imports

from __future__ import print_function
import sys
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
import sklearn.metrics as metrics
from sklearn.metrics import precision_score, recall_score, roc_auc_score, roc_curve, SCORERS
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix, f1_score

# display options

np.set_printoptions(threshold=sys.maxsize)
pd.set_option('display.max_rows', 50000)
pd.set_option('display.max_columns', 500)
pd.set_option('display.float_format', lambda x: '%.6f' % x)
np.set_printoptions(suppress=True)

# numpy arrays are loaded

X = np.load("/***/***/***/***/***/***.npy")
y = np.load("/***/***/***/***/***/***.npy")

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25)

# Ensemble ML model with Bagging Classifier using four different base classifiers
# For all classifiers, hyperparameters are obtained beforehand using hyperopt hyper-parameter optimization library
# 1a. SVM base classifier

model_svc = BaggingClassifier(base_estimator=SVC(C=474.0, kernel='rbf',
gamma=1.6480790131462046, degree=4.250804641943586), bootstrap='True',
bootstrap_features='False', max_features=0.876210911922131, max_samples=0.751498038137023,
n_jobs=-1, warm_start='False')
```

```
# training the model and prediction

model_svc.fit(X_train, y_train.ravel())
rfc_predict = model_svc.predict(X_test)
# printing the performance metric results

print("=== Results for Support Vector Machine (SVM) Base Learner ===")
print("=== Confusion Matrix ===")
np = confusion_matrix(y_test, rfc_predict)
df_cm = pd.DataFrame(np, index=["0", "1"], columns=["0", "1"])
print(df_cm)
print('\n')
print("=== F1 Score ===")
print(f1_score(y_test, rfc_predict))
print('\n')
print("=== ROC AUC Score ===")
roc_auc = roc_auc_score(y_test, rfc_predict)
print(roc_auc)

# Illustration of ROC AUC score

probs = model_svc.predict_proba(X_test)
preds = probs[:,1]
fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()


# 1b. KNN base classifier

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25)

model_knn = BaggingClassifier(base_estimator=KNeighborsClassifier(algorithm='auto',
leaf_size=26.0, n_neighbors=3, p=1.0, weights='distance'), bootstrap='False',
bootstrap_features='False', n_jobs=-1, max_features=0.954309914114886,
max_samples=0.6054250829008887, warm_start='True')

# training the model and prediction

model_knn.fit(X_train, y_train.ravel())
rfc_predict = model_knn.predict(X_test)

# printing the performance metric results

print("=== Results for KNN Base Learner ===")
print("=== Confusion Matrix ===")
```

```
np = confusion_matrix(y_test, rfc_predict)
df_cm = pd.DataFrame(np, index=["0", "1"], columns=["0","1"])
print(df_cm)
print('\n')
print("=== F1 Score ===")
print(f1_score(y_test, rfc_predict))
print('\n')
print("=== ROC AUC Score ===")
roc_auc = roc_auc_score(y_test, rfc_predict)
print(roc_auc)
```

 # Illustration of ROC AUC score

```
probs = model_knn.predict_proba(X_test)
preds = probs[:,1]
fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

 # 1c. Naive Bayes base classifier

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25)

model_nb = BaggingClassifier(base_estimator=GaussianNB(var_smoothing =
3.1522149602830965e-05), bootstrap='False', bootstrap_features='True', warm_start='False',
max_features=0.795410586443666, max_samples=0.4281704573756816, n_jobs=-1)
```

 # training the model and prediction

```
model_nb.fit(X_train, y_train.ravel())
rfc_predict = model_nb.predict(X_test)
```

 # printing the performance metric results

```
print("=== Results for Naive Bayes Base Learner ===")
print("=== Confusion Matrix ===")
np = confusion_matrix(y_test, rfc_predict)
df_cm = pd.DataFrame(np, index=["0", "1"], columns=["0","1"])
print(df_cm)
print('\n')
print("=== F1 Score ===")
print(f1_score(y_test, rfc_predict))
print('\n')
print("=== ROC AUC Score ===")
roc_auc = roc_auc_score(y_test, rfc_predict)
print(roc_auc)
```

 # Illustration of ROC AUC score

```
probs = model_nb.predict_proba(X_test)
preds = probs[:,1]
fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

 # 1d. Logistic Regression Base Classifier

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25)

model_lr = BaggingClassifier(base_estimator=LogisticRegression(C=190.0, max_iter=200.0,
multi_class='auto', n_jobs=-1, penalty='none', solver='lbfgs', tol=0.06735330298902958,
warm_start='False'), bootstrap='False', bootstrap_features='False', warm_start='False',
max_features=0.8906458640258335, max_samples=0.06929383629914337, n_jobs=-1)
```

 # training the model and prediction

```
model_lr.fit(X_train, y_train.ravel())
rfc_predict = model_lr.predict(X_test)
```

 # printing the performance metric results

```
print("=== Results for Naive Bayes Base Learner ===")
print("=== Confusion Matrix ===")
np = confusion_matrix(y_test, rfc_predict)
df_cm = pd.DataFrame(np, index=["0", "1"], columns=["0","1"])
print(df_cm)
print('\n')
print("=== F1 Score ===")
print(f1_score(y_test, rfc_predict))
print('\n')
print("=== ROC AUC Score ===")
roc_auc = roc_auc_score(y_test, rfc_predict)
print(roc_auc)
```

 # Illustration of ROC AUC score

```
probs = model_lr.predict_proba(X_test)
preds = probs[:, 1]
fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
print(roc_auc)
plt.title('Receiver Operating Characteristic')
```

```
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

*** Note that * symbols were placed where required to mask personal and institutional information.**

APPENDIX D

PYTHON CODE FOR CNN MODEL PREPROCCESSING

```
# This script is exported from a jupyter notebook environment
# imports

from __future__ import print_function
import sys
import random
import pandas as pd
import numpy as np
from sklearn import preprocessing as skpr

# display options

np.set_printoptions(threshold=sys.maxsize)
pd.set_option('display.max_rows', 50000)
pd.set_option('display.max_columns', 500)
pd.set_option('display.float_format', lambda x: '%.6f' % x)
np.set_printoptions(suppress=True)

# load .csv files and type conversion

X = pd.read_csv('/***/***/***/***/***/***.csv', low_memory=False, index_col=0)
y = pd.read_csv('/***/***/***/***/***/***.csv', low_memory=False, index_col=0)

X_np = np.zeros([100000, 1024], dtype=float)
y_np = np.zeros([100000, 1], dtype=int)

X_list = X_np.tolist()
y_list = y_np.tolist()

print(type(X_np), type(y_np))
print(X_np.shape, y_np.shape)
X.drop(["SrcAddr", "DstAddr"], axis=1, inplace=True)

X = X.to_numpy(dtype=float)
y = y.to_numpy(dtype=float)

# Scaling between 0-1 with MinMaxScaler

scaler = skpr.MinMaxScaler(feature_range=(0,1), copy=False)
scaler.fit(X)
scaler.transform(X)

# Dataframe row vector with 1x139 size converted into 1x1024 vector
# by replicating features until 973rd element (7x139). The rest (until 1024) were given
# random numbers to avoid bias.

sample_size = len(X_list)
feature_size = len(X_list[0])
```

```
sample_count = 0
for sample in range(sample_size):
    feature_count = 0
    for feature in range(feature_size):
        if feature_count < 973:
            X_list[sample_count][feature_count] = X[sample_count, (feature_count%139)]
        else:
            feature = float(random.uniform(0, 1))
        feature_count = feature_count + 1
    sample_count = sample_count + 1
print(sample_count, feature_count)

X_np = np.asarray(X_list, dtype=float)

# Dataframe row vector with 1x1024 size converted into 32x32 matrix

X_np = np.reshape(X_np, (100000, 32, 32))

np.save('/***/***/***/***/***/***.npy', X_np)
np.save('/***/***/***/***/***/***.npy', y)
```

*** Note that * symbols were placed where required to mask personal and institutional information.**

APPENDIX E

PYTHON CODE FOR CNN MODEL

```
# This script was written in google colab jupyter notebook environment,
# some functions might be environment-specific.

# imports

import tensorflow as tf
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import io
import keras
from tensorflow.python.keras.layers import Dense, Conv2D, Flatten, Dropout, MaxPooling2D
from tensorflow.python.keras import Sequential
from google.colab import files
import matplotlib.pyplot as plt
from keras.wrappers.scikit_learn import KerasClassifier
from tensorflow.python.keras.utils import to_categorical
from sklearn.metrics import precision_score, recall_score, roc_auc_score, roc_curve, SCORERS, auc,
confusion_matrix, f1_score, classification_report, multilabel_confusion_matrix

# display options

np.set_printoptions(suppress=True)

# read numpy arrays

uploaded = files.upload()
X = np.load(io.BytesIO(uploaded['80000_32_32.npy']))

uploaded = files.upload()
y = np.load(io.BytesIO(uploaded['80000_1.npy']))

# Illustration of a session in a gray-scale picture format

%matplotlib inline
plt.imshow(X[567], cmap="gray")

# required formatting

y = keras.utils.to_categorical(y, 2)
X = X.reshape(80000,32,32,1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25)
```

```
# building the model with keras sequential function
# hyper-parameters are obtained beforehand using hyperas hyper-parameter optimization library for keras

model = Sequential()
model.add(Conv2D(64, (3, 3), activation="relu", padding="same", input_shape=(32,32,1)))
model.add(Conv2D(64, (3, 3), activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.5442420919810019))

model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.43404126142165134))

model.add(Flatten())
model.add(Dense(256, activation='sigmoid'))
model.add(Dropout(0.6975075528853121))
model.add(Dense(2, activation='softmax'))

model.summary()

# Model compilation and training

model.compile(loss='categorical_crossentropy', optimizer="adam", metrics=["accuracy"])
hist = model.fit(X_train, y_train, epochs=7, batch_size=256, shuffle=True, validation_data=(X_test, y_test))

# Illustration of loss per epoch

plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper right')
plt.show()

# Illustration of accuracy per epoch

plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='lower right')
plt.show()
```

 # conversion of categorical y value to scalar value

```
y_pred_keras = model.predict(X_test)
y_pred_keras_f1 = (y_pred_keras > 0.5)
y_pred_keras_f1 = np.argmax(y_pred_keras_f1, axis=1)
y_test_f1 = np.argmax(y_test, axis=1)
```

 # printing the performance metric results

```
print("=== Confusion Matrix ===")
np = confusion_matrix(y_test_f1, y_pred_keras_f1)
df_cm = pd.DataFrame(np, index=["0", "1"], columns=["0","1"])
print(df_cm)
print('\n')
print("=== F1 Score ===")
print(f1_score(y_test_f1, y_pred_keras_f1))
print('\n')
roc_auc = roc_auc_score(y_test, y_pred_keras)
print("=== RoC Result for CNN Classifier ===")
print(roc_auc)
```

 # Illustration of ROC AUC score

```
probs = model.predict_proba(X_test)
preds = probs[:,1]
fpr, tpr, threshold = roc_curve(np.argmax(y_test, axis=1), preds)
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

*** Note that * symbols were placed where required to mask personal and institutional information.**