A Network-Based Intrusion Prevention Approach for Cloud Systems Using XGBoost and LSTM Models

by

Siddharth Gianchandani

A Thesis Presented in Partial Fulfillment
of the Requirement for the Degree
Master of Science

Approved November 2023 by the
Graduate Supervisory Committee:

Stephen Yau, Chair
Ming Zhao
Kookjin Lee

ARIZONA STATE UNIVERSITY

December 2023

ABSTRACT

The advancement of cloud technology has impacted society positively in a number of ways, but it has also led to an increase in threats that target private information available on cloud systems. Intrusion prevention systems play a crucial role in protecting cloud systems from such threats. In this thesis, an intrusion prevention approach to detect and prevent such threats in real-time is proposed. This approach is designed for network-based intrusion prevention systems and leverages the power of supervised machine learning with Extreme Gradient Boosting (XGBoost) and Long Short-Term Memory (LSTM) algorithms, to analyze the flow of each packet that is sent to a cloud system through the network. The innovations of this thesis include developing a custom LSTM architecture, using this architecture to train a LSTM model to identify attacks and using TCP reset functionality to prevent attacks for cloud systems. The aim of this thesis is to provide a framework for an Intrusion Prevention System. Based on simulations and experimental results with the NF-UQ-NIDS-v2 dataset, the proposed system is accurate, fast, scalable and has a low rate of false positives, making it suitable for real world applications.

Dedication

This thesis is dedicated to my mother, who has been instrumental in supporting me throughout my education.

ACKNOWLEDGMENTS

I would like to acknowledge the very valuable guidance provided by my committee chair, Professor Stephen S. Yau, whose advice has helped me during my thesis journey at Arizona State University. Professor Yau's vision, wealth of life and academic experience have enhanced my research capabilities along with my academic and personal growths. Furthermore, I extend my gratitude to my committee members, Professors Ming Zhao and Kookjin Lee, for their time and valuable guidance, which were essential in the successful completion of this thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

Over the last few decades, an increased use of cloud computing for communications related to all aspects of life has been observed. These communications include private communications like sending essential bank details for online money transfers. Thus, there is a need to ensure that such communications remain private and protected. Unfortunately, the advancements in computer systems and communications have led to an increase in the number of threats to the privacy and security of our society. Attackers utilize the increased computation power of computers towards performing cyberattacks on cloud systems to get confidential data and use it for their own financial gain. This has led to an ever-increasing need for measures that need to be taken to protect the cloud systems from these intrusions and ensure that privacy of confidential data is maintained.

Currently, many novel attacks are being generated and this fact poses a challenge for network security to accurately detect intrusions [1]. Several methods have been developed to safeguard cloud systems like attack prediction, intrusion detection systems (IDS) and intrusion prevention systems (IPS). Approaches for attack prediction are time-based and generate probabilistic predictions of system-wide and sub-system security breaches with the respective time windows, in which breaches are most likely to occur [2]. Network-based IPS and IDS intercept packets sent to cloud systems and filter out suspicious packets. These systems are used as part of a layered approach to secure cloud systems against cyberattacks like hacking attempts, malware and data breaches. They work in conjunction with firewall and antivirus software to ensure the security of cloud systems. The difference between an IPS and an IDS is that

an IPS can work without human interaction. Intrusion prevention systems are designed to automatically prevent malicious attempts by attackers, making them more powerful than intrusion detection systems, which notify trained personnel when they detect these malicious attempts. Preventing a malicious attempt refers to detecting, blocking, or mitigating potentially harmful activities like cyberattacks, before they can cause damage or compromise the security of a cloud system. Due to the added power of IPS, they have an added responsibility of ensuring that a non-attacker is not classified as an attacker. Unfortunately, commercial intrusion detection systems and intrusion prevention systems typically suffer from low detection rates and high false positives which require substantial optimization and network specific fine tuning [3]. Most of the current intrusion detection systems and intrusion prevention systems rely on signature detection methods. Such methods are effective when dealing with known attacks, but it does not allow them to adapt to new kinds of attacks. It is also a challenge to expand such methods to cover more attacks. Though there have been efforts to introduce intelligence to these systems through machine learning, both supervised and unsupervised, there has been little real-world adaptation of these solutions due to their own inherent challenges and bottlenecks [3]. The intrinsic issues with supervised models include lack of training data for new attacks, and need of resources to train a good model. The biggest advantage associated with supervised models is that they are universal, and one model can be used to safeguard multiple cloud systems. Supervised models are scalable, effective for real-time detection of known attacks and applicable for new cloud systems and cloud systems expecting traffic change. The inherent issue of using unsupervised models is that they are not universal. An unsupervised model trained for one cloud system has a high chance of being ineffective for another cloud system without appropriate training for the new cloud system. Another disadvantage of using unsupervised learning can be observed

when the network traffic experiences a substantial change. One such example is when Animoto integrated with Facebook in 2008, it attracted 750,000 new users in 3 days [4]. The normal traffic for Animoto before this consisted of a few hundred users a day. If this company was using an unsupervised model for safeguarding their cloud systems, they would have either had many false positives or they would have had to use some other way to safeguard their cloud systems for a certain period of time that allowed their unsupervised model to get used to this change.

The aim of this thesis is to overcome issues associated with using traditional supervised machine learning models in intrusion prevention systems. The proposed intrusion prevention approach uses a hybrid supervised machine learning technique utilizing a combination of XGBoost and LSTM models. TCP reset is an abrupt closure of the session [5]. This hybrid technique, when used along with TCP reset functionality of TCP protocol in an intrusion prevention system, shows significant promise to address the limitations of traditional methods. The innovations of this thesis include developing a custom LSTM architecture, using this architecture to train a LSTM model to identify attacks and using TCP reset functionality to prevent attacks for cloud systems. This method is applicable to new cloud systems or cloud systems that anticipate network traffic change in addition to normal cloud systems due to use of supervised models. This thesis is organized into 10 chapters. The first chapter explains the motivation of this study. Background details of technologies used, recent research, approach, innovation and illustrative example are discussed in the second, third, fourth, fifth and sixth chapters respectively. The seventh chapter gives details of simulation. The eighth chapter will show evaluation of the models. The ninth chapter will discuss the limitations of this approach. The tenth chapter will discuss the conclusion and future work.

Chapter 2

BACKGROUND

2.1 Machine Learning Models

XGBoost is an efficient and powerful algorithm which supports easy learning. The XGBoost model runs more than ten times faster than popular solutions on a single machine and scales to billions of examples in distributed or memory-limited settings [6]. This makes it a very desirable solution for a real-time Intrusion Prevention Systems. This model builds decision trees sequentially, where each tree is trained to correct errors of previous trees. XGBoost also has a great inbuilt functionality of ranking the features it is trained on. This functionality gives a feature importance score for all the features it is trained on, which allows the selection of the most useful features from a dataset to train machine learning models. This process results in training higher performing models with reduced size of datasets. This helps in saving resources that would be spent on complex computations and extensive training to get the same result.

LSTM is a kind of Recurrent Neural Network (RNN). LSTM is one of the most powerful and dynamic classifiers that is known to the public [7]. One of the main difficulties encountered when training conventional RNNs is the issue of gradients diminishing, which hinders the network's capacity to comprehend extended patterns within sequential data. LSTMs were expressly devised to tackle this challenge. LSTMs undergo training via a method known as backpropagation through time (BPTT), akin to traditional RNNs. However, owing to their adeptness at capturing prolonged dependencies, LSTMs frequently exhibit superior performance in the acquisition of

knowledge from sequential data. LSTMs find extensive utility across a spectrum of applications involving sequential data and are used in diverse fields like natural language processing (NLP), time series analysis and recognizing handwriting patterns. Considering the ability of LSTMs to comprehend extended patterns within sequential data, and the fact that Denial of Service (DoS) and Distributed Denial of Service (DDoS) attacks are carried out over a period of time, it can be inferred that a LSTM model will be able to analyze the sequential data associated with packets coming in from suspicious IP addresses and detect these attacks accurately. This is the main reason why I have proposed using the LSTM model solely to detect DoS and DDOS attacks. The XGBoost model will be used to detect DDoS, DoS, Brute Force, Infiltration, and Bot attacks.

2.2 Cloud Systems

Cloud systems leverage the power of distributed computing, allowing users to access and utilize a wide range of computing resources over the internet. These resources include virtualized servers, storage, databases, and networking components which are hosted and managed by third-party cloud service providers. Transmission Control Protocol (TCP) is a communications standard that enables application programs and computing devices like cloud systems to exchange messages over a network. TCP provides a reliable, connection-oriented transmission service, ensuring all packets arrive at their destination in order without loss or duplication [8]. The TCP reset mechanism is a standard part of the TCP protocol. A TCP reset is identified by the RESET flag in the TCP header set to 1 [5]. TCP reset is used to abruptly close connections and prevents attackers from establishing and maintaining connections [9]. Tt causes the resources allocated to the connection to be immediately released and

all other information about the connection is erased. These resources can be used immediately by the cloud system.

2.3 NF-UQ-NIDS-v2 Dataset

Most of the benchmark datasets like NSL-KDD and UNSW-NB15 are old and limited. There are only 4 attacks in KDD99 and NSL-KDD datasets with 4.8 million total records and 311 thousand records respectively [10, 11]. UNSW-NB15 dataset only has 1,623,118 records with 1,550,712 (95.54 %) benign records 72,406 (4.46 %) attack records for 9 types of attacks [12]. The newer dataset, CIC-IDS2017, is built on a limited number of 25 users and thus, does not capture the real life scenario of a typical network traffic [13]. CSE-CIC-IDS2018 dataset also has a highly skewed distribution with only 12.14 % of the dataset being attack samples [12]. Due to these issues with datasets, I have selected the NF-UQ-NIDS-v2 dataset for this study. This dataset developed in [12] is the latest version of the NF-UQ-NIDS dataset. This dataset combines four popular real world datasets, namely UNSW-NB15, NF-ToN-IoT, NF-BoT-IoT and CSE-CIC-IDS2018. This dataset has 43 input features (45 with labels) and 75,987,976 records consisting of various kinds of attacks. Since this thesis is only for cloud systems, I have considered 21,191,222 records from the total to train the machine learning models, after filtering out the records corresponding to the IoT datasets from the NF-UQ-NIDS-v2 dataset and certain attacks. I have considered 5 attack classes from this dataset, namely DDoS, DoS, Infiltration, Brute Force, and Bot attacks, based on the number of records available for those classes, their popularity and techniques that can be used to mitigate the attacks. For example, Fuzzers can be mitigated with input validation and using secure coding practices. A breakdown of the 21,191,222 records is given below in the Table 2.1. The original

features of the NF-UQ-NIDS-v2 developed in [12] are shown in Table 2.2.

Label	**Records**
Benign	18,930,789
DDoS	1,390,270
DoS	489,793
Infiltration	116,361
Brute Force	120,912
Bot	143,097
Total	21,191,222

Table 2.1: Breakdown of Records in NF-UQ-NIDS-v2 Dataset

Feature	**Description**
IPV4_SRC_ADDR	IPv4 source address
IPV4_DST_ADDR	IPv4 destination address
L4_SRC_PORT	IPv4 source port number
L4_DST_PORT	IPv4 destination port number
PROTOCOL	IP protocol identifier byte
L7_PROTO	Layer 7 protocol (numeric)
IN_BYTES	Incoming number of bytes
OUT_BYTES	Outgoing number of bytes
IN_PKTS	Incoming number of packets
OUT_PKTS	Outgoing number of packets

Feature	**Description**
FLOW_DURATION_MILLISECONDS	Flow duration in milliseconds
TCP_FLAGS	Cumulative of all TCP flags
CLIENT_TCP_FLAGS	Cumulative of all client TCP flags
SERVER_TCP_FLAGS	Cumulative of all server TCP flags
DURATION_IN	Client to Server stream duration (msec)
DURATION_OUT	Client to Server stream duration (msec)
MIN_TTL	Min flow TTL
MAX_TTL	Max flow TTL
LONGEST_FLOW_PKT	Longest packet (bytes) of the flow
SHORTEST_FLOW_PKT	Shortest packet (bytes) of the flow
MIN_IP_PKT_LEN	Len of the smallest flow IP packet observed
MAX_IP_PKT_LEN	Len of the largest flow IP packet observed
SRC_TO_DST_SECOND_BYTES	Src to dst Bytes/sec
DST_TO_SRC_SECOND_BYTES	Dst to src Bytes/sec
RETRANSMITTED_IN_BYTES	Number of retransmitted TCP flow bytes (src->dst)
RETRANSMITTED_IN_PKTS	Number of retransmitted TCP flow packets (src->dst)
RETRANSMITTED_OUT_BYTES	Number of retransmitted TCP flow bytes (dst->src)

Feature	**Description**
RETRANSMITTED_OUT_PKTS	Number of retransmitted TCP flow packets (dst->src)
SRC_TO_DST_AVG_THROUGHPUT	Src to dst average thpt (bps)
DST_TO_SRC_AVG_THROUGHPUT	Dst to src average thpt (bps)
NUM_PKTS_UP_TO_128_BYTES	Packets whose IP size $\leq$ 128
NUM_PKTS_128_TO_256_BYTES	Packets whose IP size $>$ 128 and $\leq$ 256
NUM_PKTS_256_TO_512_BYTES	Packets whose IP size $>$ 256 and $\leq$ 512
NUM_PKTS_512_TO_1024_BYTES	Packets whose IP size $>$ 512 and $\leq$ 1024
NUM_PKTS_1024_TO_1514_BYTES	Packets whose IP size $>$ 1024 and $\leq$ 1514
TCP_WIN_MAX_IN	Max TCP Window (src->dst)
TCP_WIN_MAX_OUT	Max TCP Window (dst->src)
ICMP_TYPE	ICMP Type * 256 + ICMP code
ICMP_IPV4_TYPE	ICMP Type
DNS_QUERY_ID	DNS query transaction Id
DNS_QUERY_TYPE	DNS query type (e.g., 1=A, 2=NS...)
DNS_TTL_ANSWER	TTL of the first A record (if any)
FTP_COMMAND_RET_CODE	FTP client command return code

Table 2.2: Original Input Features of NF-UQ-NIDS-v2 Dataset

Chapter 3

CURRENT STATE-OF-THE-ART

Cybersecurity is an immensely broad topic, with different measures designed to counter different attack vectors [14, 15]. The use of machine learning and honeypots in intrusion prevention systems for such an endeavour has been researched for several years by many researchers. Yudai et al. [16] discussed the Log4Shell vulnerability in Apache Log4j and how this vulnerability allowed HTTP protocol attacks. They used low-interaction and high-interaction honeypots to collect data for different HTTP protocol attacks and their variants. This collected data was used to train machine learning models based on URL strings and rules were generated based on the results of the models for their Intrusion Prevention System. They worked with Naive Bayes, Generalized Liner Model, Logistic Regression, Fast Large Margin, Deep Learning, Decision Tree, Random Forest, Gradient Boosted Trees, and Support Vector Machine models. The attacks related to Log4Shell have not been considered in this study as this vulnerability was patched in later versions of Apache Log4j within two weeks of being publicly announced on December 9, 2021 [17].

Wooseok et al. [18] have proposed a two-level classifier system designed to enhance both real-time performance and detection accuracy in intrusion prevention systems. This system employs level 1 and 2 classifiers internally. The level 1 classifier initially performs real-time detection with moderate accuracy for incoming data traffic. If the data cannot be classified with high probability by the classifier, the classification is delayed until the traffic flow terminates. The level 2 classifier then collects the statistical features of the traffic flow for performing precise classification. They have trained their models separately on UNSW-NB15 and CICIDS2017 datasets with promising

results. They have used a subset of these datasets to avoid class imbalance. They have considered DoS, DDoS, Exploits, Reconnaissance, Generic, Fuzzers, Shell code, Bot, PortScan, SSH-Patator and FTP-Patator attacks out of all the attacks present in the two datasets. Exploit attacks take advantage of vulnerabilities or weaknesses in software, hardware, or system configurations to gain unauthorized access to a computer system or network. Thus, upgrading software to more secure versions that patch known vulnerabilities and using secure libraries for coding purposes can reduce the possibilities of exploit attacks. This solution is also applicable to Shell code attacks. Reconnaissance and Portscan attacks can be mitigated by using access control and network segmentation. The Generic attacks in the UNSW-NB15 dataset focus on targeting cryptography and causing a collision with each block-cipher [12]. The challenge with executing such attacks is that finding a collision in a resilient block cipher with a strong key is extremely difficult. Thus, attacks of this type can be easily mitigated by using a resilient block cipher with a strong key. Fuzzers can be avoided with input validation and using secure coding practices. SSH-Patator and FTP-Patator attacks can be mitigated by enforcing strong password policies and implementing account lockout mechanisms after a certain number of failed login attempts to prevent attackers from repeatedly guessing passwords. Due to these reasons, these attacks are not considered in this study.

Akhil et al. [19] proposed the use of Multi-Layer Perceptron (MLP) in an intrusion detection and prevention System with specific scripts designed for cyberattacks. They have used the KDD99 dataset to train this model. The model was trained on approximately 4.8 million records and achieved a reasonable accuracy of 91.4%. The attacks considered were DOS, probe, U2R and R2L. Probe, U2R and R2L attacks are not considered in this study because records for these attacks are not available in the NF-UQ-NIDS-v2 dataset. However, since the NF-UQ-NIDS-v2 dataset focuses on

more popular attacks like DDoS, Bot, Infiltration and Brute Force, it is more desirable than the KDD99 dataset. Another advantage of using the NF-UQ-NIDS-v2 dataset is that there are more records available in this dataset, the number of records for Benign class alone greatly exceed the total number of records in the KDD99 dataset.

Constantinides [3] proposed a novel network intrusion prevention system that utilises a Self Organizing Incremental Neural Network along with a Support Vector Machine. This model was trained on the NSL KDD dataset and achieved an accuracy of 89.67%. As NSL KDD is the updated version of the KDD99 dataset, both the datasets consider the same attacks, namely DoS, Probe, U2R and R2L attacks.

Research related to intrusion prevention systems have utilized techniques other than machine learning for the purpose of detecting network intrusions. Vibha et al. [20] discussed the use of signature-based pattern matching algorithms in an intrusion detection and prevention System. They experimented with Brute-force, RabinKarp, Boyer-Moore and Knuth-Morris-Pratt (KMP) algorithms for this purpose. Their results show that KMP and Boyer-Moore algorithms performed the best. Yadav et al. [21] have proposed an intrusion detection and prevention system for integrated internet environments with wireless sensor networks (WSN). They have defined rules for each level of communication and different attack. They have focused mainly on Denial of Service (DoS) attacks. Using sensing devices and gateways, this system effectively prevents threats to wireless communication, particularly when integrated with CoAP.

This study also discusses some related advancements in the fields of intrusion detection systems and intrusion detection since these fields are very closely related to intrusion prevention systems. Sarumi et al. [22] compared two systems for detecting intrusions: Apriori and Support Vector Machine(SVM). The NSL-KDD dataset and UNSW-NB15 datasets were utilized to assess the effectiveness of the two systems.

They were able to achieve an accuracy of 77% on the NSL-KDD dataset and 90.41% accuracy on the UNSW-NB15 dataset with the SVM model.

A Deep Belief Network (DBN) attack detection approach was proposed by Reddy and Shyam [23] in which the activation function and weights are refined using the Median Fitness focused Sea Lion Optimization method (MFSLnO). When DBN identifies a malicious point passed control to a bait technique that is lightweight and consistently moderates the most prevalent malicious nodes while maintaining normal relations.

A model based on a hierarchy of deep learning in the identification of anomaly and summarized the study articles on the subject were presented by Gamage and Samarabandu [24]. They trained 4 models, Autoencoder, Deep Belief Network, Feed-forward Neural Network, and LSTM network on KDD 99, NSL-KDD, CIC-IDS2017 and CIC-IDS2018 datasets. They considered Probe, DoS, U2R, R2L, Bot, DDoS, FTP-Patator, PortScan, SSH-Patator, Brute Force, XSS, FTP-BruteForce, Infilteration, SQL Injection, SSH - Bruteforce attacks. Similar to SSH-Patator and FTP-Patator attacks, SSH - BruteFore and FTP-BruteForce attacks can be mitigated by enforcing strong password policies and implementing account lockout mechanisms after a certain number of failed login attempts to prevent attackers from repeatedly guessing passwords. XSS attacks can be easily mitigated by using input validation techniques and secure cookies. SQL injection attacks can be easily mitigated by using stored procesures, input validation techniques and error handling in code. Hence, these attacks are not considered in this study.

The UNSW-NB15 intrusion detection dataset was used by Kasongo and Sun [25] for training and testing Support Vector Machine, Artificial Neural Network, K-Nearest-Neighbor, Decision Tree, and Logistic Regression models. They used the XGBoost algorithm to provide a filter-based feature reduction strategy. Their results

demonstrated that the XGBoost-based feature selection method allows for methods such as Decision Tree to increase its test accuracy from 88.13 to 90.85 % for a binary classification scheme.

Kasango [26] used the NSL-KDD and UNSW-NB15 datasets along with XGBoost feature selection to train RNN, GRU and LSTM models. They focused on both binary classification and multiclass classification tasks for both datasets without addressing the class imbalance problems. This study improves their approach by training XGBoost and LSTM models on the latest dataset for network intrusion, NF-UQ-NIDS-v2 dataset. This study addresses the class imbalance problems in the dataset and the models are trained for multiclass classification. The XGBoost and LSTM models are used together in this study to prevent attacks in real-time in an Intrusion Prevention System. The LSTM model trained in this study outperforms their LSTM model for UNSW-NB15 dataset in terms of accuracy, precision, recall, and F1 score, and it has less number of layers, making it less complicated and more desirable for real world scenarios. Only their LSTM model for UNSW-NB15 dataset is compared because NF-UQ-NIDS-v2 dataset incorporates records from the UNSW-NB15 dataset. Additionally, this study proposes the use of TCP reset functionality to make the proposed framework more suited to real world issues.

Farhat et al. [27] presented CADS-ML/DL, an efficient cloud-based multi-attack detection system. They evaluated eight intrusion detection systems based on machine learning and deep learning algorithms, including Decision Tree, Random Forest, Extreme Gradient Boosting, Gated Recurrent Units, Long Short-Term Memory, Stacked LSTM, and Bidirectional LSTM models. Their experimental results demonstrated that the CADS-ML/DL system, specifically the XGBoost model, outperformed the other models, exhibiting an accuracy of 97.70 % and a false error rate of 0.0230. They tested the XGBoost model on the CSE-CICIDS2018 dataset, attaining an accuracy

score of 99.99 and a false error rate of 0.0001.

Yau et al. [2] presented a time-based approach to quantitatively predicting imminent security breaches on critical cloud infrastructures. It used Bayesian network and Markhov Decision process to predict system-wide security breaches based on the probabilistic inputs on subsystem level security breaches and generated probabilistic predictions of system-wide and sub-system security breaches with respective time windows in which breaches are most likely to occur. Yau et al. [28] used the NSL-KDD and UNSW-NB15 datasets to train an Artificial Neural Network (ANN) model with genetic feature selection. They achieved an accuracy of 91.98% on the NSL-KDD dataset and an accuracy of 95.46% on the UNSW-NB15 dataset.

Poornima et al. [29] used the NSL-KDD dataset and XGBoost feature selection to train a LSTM model for 2 different classes, 'benign' or 'attack', for network attack classification. Their LSTM model had a learning rate of 0.01 and gave scores of 98.99% accuracy, 0.91 precision, 0.87 recall, and 0.85 F1 score. This study improves their approach with the use of an XGBoost and a LSTM model together to prevent attacks in real-time in an Intrusion Prevention System. While the LSTM model trained in this study gives a comparable accuracy score to their model, it outperforms their model in terms of precision, recall, and F1 score. The models in this study are trained on the latest dataset for network intrusion, NF-UQ-NIDS-v2 dataset. The models in this study are also trained to identify particular attacks, giving results more suited to real world scenarios, as compared to just identifying whether a network data is an attack or not an attack. Additionally, this study proposes the use of TCP reset functionality to make the proposed framework more suited to real world issues.

Patrik [9] used TCP reset cookies to mitigate TCP SYN flood attack in TCP connections. Their study proves how TCP reset cookies can be used to block SYN flooding attacks from spoofed IP addresses. They mention how their method needs

logic to switch between different SYN flood mitigation strategies. This study improves their approach by applying the use of TCP reset with two supervised machine learning models for 5 kinds of attacks in real time to protect cloud systems.

The XGBoost–DNN model was introduced by Devan and Khare [30], which uses the XGBoost algorithm for feature selection and classifying the deep neural network (DNN) network attacks. They used an Adam optimizer to train their model on the NSL-KDD dataset. They performed binary classification task and compared their results with logistic regression, SVM and Naive Bayes models to showcase its performance.

A new NIDS architecture based on a deep convolutional neural network was proposed by Khan et al. [31] which utilizes the network spectrogram images created with the help of the short-time Fourier transform. They used the CIC-IDS2017 dataset to assess the efficiency of the suggested solution. Attia et al. [32] have used the XGBoost model for feature extraction and then trained ANN model on Kitsune dataset. This dataset has 115 features and 21 million records. This dataset was divided into a 70-30 split for training and testing data. Hasan et al. [33] have developed a learning model for fast learning network (FLN) based on particle swarm optimization (PSO) that is named PSO-FLN. This model is trained on the KDD99 dataset. Pawlicki et al. [15] used the NSL-KDD and CICIDS2017 datasets to train a variety of ANN models. They have also done extensive work on hyperparameter tuning and showed changes of more than 9% in accuracy due to changes in hyperparameters. Imrana et al. [34] has used the NSL-KDD dataset to train a bidirectional Long-Short-Term-Memory (BiDLSTM) model. The BiDLSTM model achieved a higher detection accuracy for U2R and R2L attacks than conventional LSTM on this dataset.

Sarhan et al. [12] used their own dataset, NF-UQ-NIDS-vs dataset and its components to train an ensemble tree classifier known as Extra Trees to test their dataset

against its benchmark components. This was done for both binary and multiclass classification tasks. For binary classification, the combined dataset outperformed the original feature sets in terms of attack detection performance. For multiclass classification, the dataset substantially outperformed the original feature sets, with more than a 6% increase in accuracy between CSE-CIC-IDS2018 and NF-CSE-CIC-IDS2018-v2 datasets for the same model. The Extra Trees model achieved an accuracy of 96.93% on the overall NF-UQ-NIDS-v2 dataset.

Chapter 4

PROPOSED APPROACH

In this chapter, section 4.1 describes initial configuration for Intrusion Prevention System. Section 4.2 describes the details on preprocessing the dataset. Section 4.3 discusses feature selection process and model training in detail. The proposed IPS framework uses two supervised machine learning models: XGBoost and LSTM, along with an automated control unit and a list of malicious IP addresses. Section 4.4 discusses the overall approach of the proposed framework for IPS. Section 4.5 discusses the use of XGBoost in the approach. Section 4.6 discusses the use of the LSTM in the approach. Section 4.7 discusses the use of control unit and TCP reset request in the approach.

Section 4.1 Initial Configuration for The Proposed Intrusion Prevention System

The following description describes the steps for the initial configuration for Intrusion Prevention System. Figure 4.1 presents the following description in a block diagram.

The initial configuration for the proposed Intrusion Prevention System starts with preprocessing the NF-UQ-NIDS-v2 dataset. This dataset is not suitable for training the XGBoost and LSTM models directly due to presence of string values and class imbalance. So this dataset was preprocessed and made usable for training the models. Section 4.2 discusses this process in detail.

After preprocessing the dataset, XGBoost model was trained and fine-tuned. The best XGBoost model was used for feature selection. A training dataset was prepared

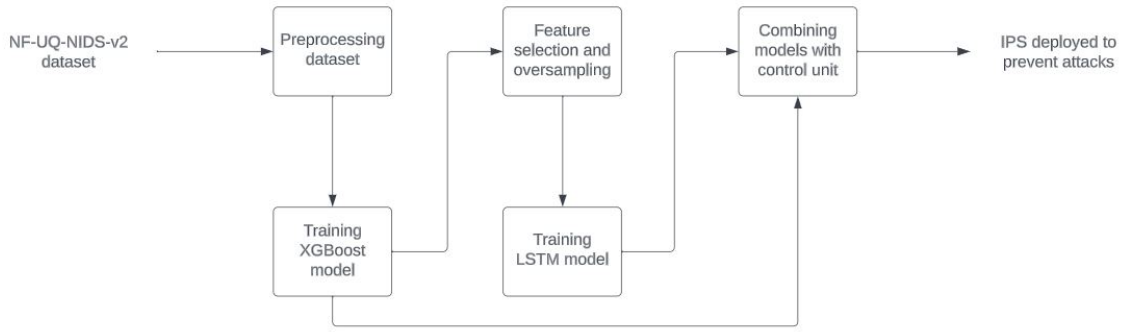

Figure 4.1: Initial Configuration for Intrusion Prevention System

for the LSTM model using these selected features with oversampling. Section 4.3 describes specific details of this step. The LSTM model was trained on the oversampled training dataset and evaluated. Several different versions of the LSTM model were trained and tested. Details on the architecture of the best LSTM model and its hyperparameters have been described in section 4.6. Intermediate results for training and fine-tuning both the models are shown in section 8.5.

After selecting the best XGBoost and LSTM models, they are combined with a control unit and used in the Intrusion Prevention System. A simulation was done using them, showing the reports generated by these models. The simulation has been described in detail in chapter 7. The details on evaluation of both models are included in chapter 8.

Section 4.2 Preprocessing the Dataset

As shown in Table 1, there is a huge class imbalance between the classes, with 18.9 million records belonging to 'Benign' class and 2.2 million records distributed between 5 classes. To reduce this imbalance, we have downsampled the records in

'Benign' class by 50%, using 9,465,395 records. I have used all the records for the other labels, which brings our total number of records to 11,725,828. I will also convert the string labels into numerical categorical values. 0 will stand for 'Benign' class, 1 for 'DDoS' class, 2 for 'DoS' class, 3 for 'Brute Force' class, 4 for 'Infiltration' class and 5 for 'Bot' class. To create separate training and testing datasets for our models, I employed an 80-20 split strategy. I separated each class into a separate file, then randomly selects 80% of the data from each class and aggregated them into one file for training purposes. Similar steps were taken to create a file for testing purposes, to evaluate the effectives of the trained model. This approach maintains a balanced representation of each distribution type in both training and testing files. It also guarantees that the model learns effectively from diverse examples, thus enabling it to make accurate predictions when confronted with unseen data. In addition, it has the benefit of assuring us that we avoided cases of where one class or more could be absent from either the training file or the testing file, no matter how small the possibilty could be. In summary, this preprocessing phase is pivotal in constructing a reliable and precise machine learning model for classification tasks, given the dataset's total size of 11,725,828 data points spanning all 6 classes.

In addition to the above steps, I also normalized some of the 43 attributes available in the dataset and dropped two of them ('`IPV4_SRC_ADDR`' and '`IPV4_DST_ADDR`') when combining the training and testing files for the XGBoost model. The attributes selected for normalization are shown in figure 4.2. I have used the min-max scaling approach to normalize these features and put them in range [-1,1]. This normalized value is calculated based on equation 4.1.

$$x_{\text{scaled}} = \frac{x - x_{\text{mean}}}{x_{\text{max}} - x_{\text{min}}} \tag{4.1}$$

Features Normalized
OUT_BYTES
IN_BYTES
OUT_PKTS
IN_PKTS
FLOW_DURATION_MILLISECONDS
DURATION_IN
DURATION_OUT
LONGEST_FLOW_PKT
SHORTEST_FLOW_PKT
MAX_IP_PKT_LEN
MIN_IP_PKT_LEN
DST_TO_SRC_AVG_THROUGHPUT
SRC_TO_DST_AVG_THROUGHPUT
NUM_PKTS_128_TO_256_BYTES
NUM_PKTS_1024_TO_1514_BYTES
NUM_PKTS_UP_TO_128_BYTES
NUM_PKTS_256_TO_512_BYTES
NUM_PKTS_512_TO_1024_BYTES
RETRANSMITTED_IN_BYTES
RETRANSMITTED_OUT_PKTS
RETRANSMITTED_OUT_BYTES
RETRANSMITTED_IN_PKTS
DST_TO_SRC_SECOND_BYTES
SRC_TO_DST_SECOND_BYTES

Figure 4.2: Features Normalized

For each column, the maximum value in that column is denoted by x_{max}. The minimum value in that column is represented by x_{min}. x_{mean} represents the mean value of that column and x is the value that is currently being scaled.

The reason for dropping '`IPV4_SRC_ADDR`' attribute was that this attribute is the IPv4 source address, which is categorical value and it will differ for each entity sending packets to the cloud system. The reason for dropping '`IPV4_DST_ADDR`' attribute was that this attribute is the IPv4 destination address, which is a constant categorical value for the address of a particular cloud system. Thus, they will make no positive

contribution to the analysis done by the XGBoost model. Instead, there is a possibility to create unnecessary bias against some IPv4 source addresses, which will lead to an increase in the false positive results of the model, leading to a lower performance of this model.

Section 4.3 Feature Selection and Training Models

XGBoost is a powerful machine learning model that possesses a feature importance mechanism. This intrinsic capability enables XGBoost to discern the most influential features within a dataset. This capability allows use to train machine learning models on the important features in a dataset, which improves their performance. Instead of relying solely on traditional statistical techniques, XGBoost employs an ensemble approach that iteratively refines its understanding of feature importance during the training process. By observing how features are utilized across multiple decision trees, it assigns importance scores to each attribute, revealing their impact on model predictions. These scores are then utilized to uncover valuable insights, facilitating data-driven decision-making, model refinement, and feature selection for enhanced predictive accuracy. In essence, XGBoost's feature importance functionality serves as a potent tool to uncover the hidden gems within a dataset. The XGBoost model consists of a set of decision trees, as shown in [6] by Chen et al. This is represented mathematically in equation 4.2.

$$y_i = \sum_{n=1}^{n} f(x_i) \tag{4.2}$$

Here n represents the number of decision trees, f corresponds to an independent tree structure, and x_i is the ith data point feature vector. For the purposes of training

a good model, it is required to minize the loss as much as possible. The objective function is represented mathematically by equation 4.3.

$$L = \sum_{i} l(y_i, y_t) + \sum_{n} \Omega(f) \tag{4.3}$$

Here y_i is the predicted value, y_t is the target value and l is a differentiable convex loss function that measures the difference between these two. $\Omega(f)$ is mathematically represented in Equation 4.4.

$$\Omega(f) = r^T + \frac{1}{2} \sum_{j=1}^{T} w_j^2 \tag{4.4}$$

Here T is the number of leaves and w_j^2 is the score of the jth leaf. $\Omega(f)$ is a regularization term that penalizes the complexity of the model and helps to smooth final learnt weights to avoid over-fitting. Intuitively, the regularized objective will tend to select a model employing simple and predictive functions. Due to an inbuilt functionality in the 'xgboost' library in python, I can get the feature importance scores of all input features directly, which helps conserve resources that would otherwise have been utilized towards performing complex mathematical calculations.

I have trained the XGBoost model on all 41 attributes in the training file and then used it to get a feature importance score for each of the 41 attributes. This resulted in a total of 22 features getting a feature importance score above 0.001. Similar to the approach shown by Kasango in [26], I trained the LSTM model on only these 22 features. A bar graph depicting this result for all 41 attributes is shown in figure 4.3. A table showing the top 22 attributes with feature importance score of more than

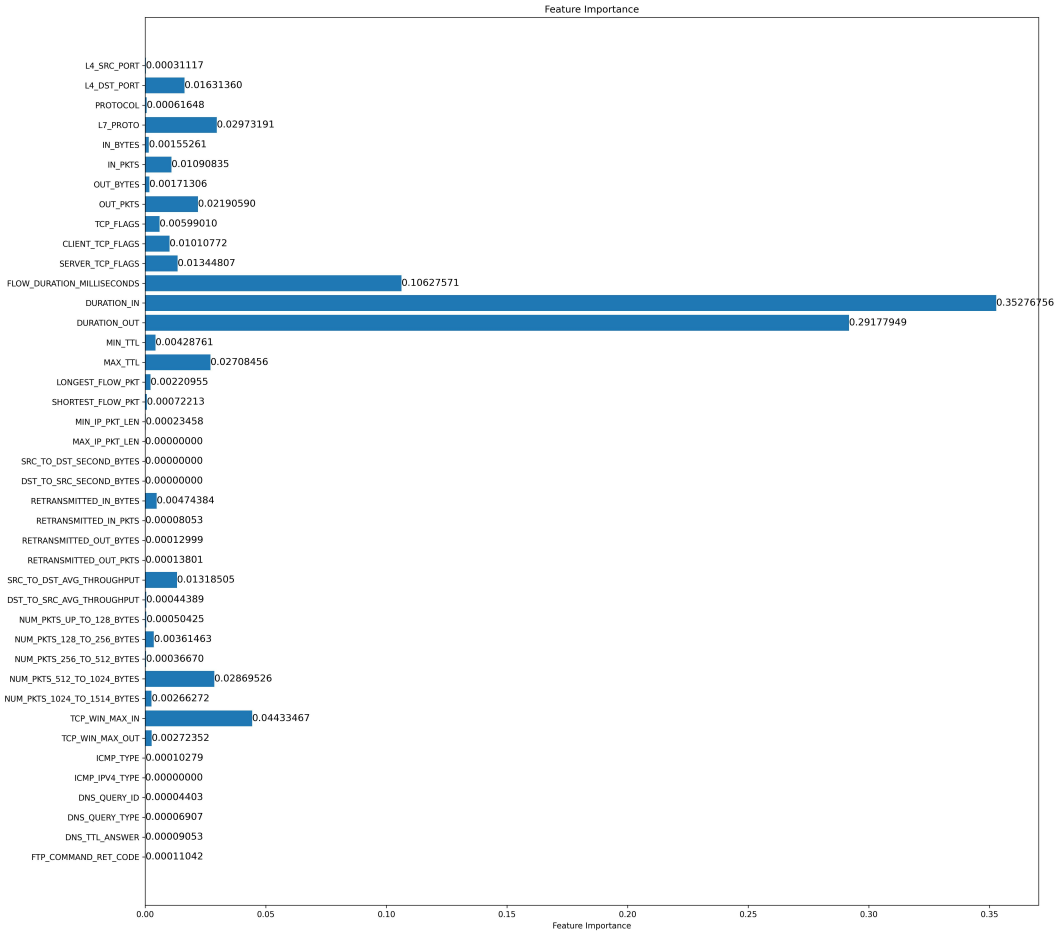

Figure 4.3: Feature Importance Scores of All 41 Attributes

Feature	Feature Importance Score
DURATION_IN	0.352768
DURATION_OUT	0.291779
FLOW_DURATION_MILLISECONDS	0.106276
TCP_WIN_MAX_IN	0.0443347
L7_PROTO	0.0297319
NUM_PKTS_512_TO_1024_BYTES	0.0286953
MAX_TTL	0.0270846
OUT_PKTS	0.0219059
L4_DST_PORT	0.0163136
SERVER_TCP_FLAGS	0.0134481
SRC_TO_DST_AVG_THROUGHPUT	0.013185
IN_PKTS	0.0109083
CLIENT_TCP_FLAGS	0.0101077
TCP_FLAGS	0.0059901
RETRANSMITTED_IN_BYTES	0.00474384
MIN_TTL	0.00428761
NUM_PKTS_128_TO_256_BYTES	0.00361463
TCP_WIN_MAX_OUT	0.00272352
NUM_PKTS_1024_TO_1514_BYTES	0.00266272
LONGEST_FLOW_PKT	0.00220955
OUT_BYTES	0.00171306
IN_BYTES	0.00155261

Figure 4.4: Feature Importance Scores of Top 22 Attributes

0.001 is shown in figure 4.4.

Regarding the hyperparameters used to train the XGBoost model on the NF-UQ-NIDS-v2 dataset, I set the objective to be 'multi:softprob' because I am dealing with a multiclass classification problem. The number of classes is 6 as there are 6 classes in the dataset for XGBoost model. The gradient descent tree type booster 'gbtree' was used due to its ability to extrapolate non-linearity between input variables and target label. Learning rate was set to 0.3 and maximum tree depth was set to 6. A fast histogram optimized approximate greedy algorithm 'hist' was used as the tree method with maximum number of bins set to 100. A separate training dataset was created for LSTM after performing feature normalization and feature selection. This consisted of records for 3 labels, namely 'Benign', 'DDoS' and 'DoS'. Labels for 'DDoS' and 'DoS' were oversampled, increasing the number of records to 19,604,716.

Section 4.4 The Overall Approach for Intrusion Prevention System

Figure 4.5 describes the overall approach for the proposed framework. The following description describes the overall steps for the proposed IPS framework:

Step 1: Incoming packet is intercepted by the IPS server. (1.1) If IP address is inside the list of known malicious IP addresses stored in the server, this packet is dropped. (1.2) If IP address is not in that list, packet is stored in IPS server and packet information is sent to XGBoost model for analysis.

Step 2: XGBoost model analyses the packet information (discussed in detail later in section 4.5). (2.1) First possible output is low probability of attack: Intercepted packet is sent to the cloud system. (2.2) Second possible output is high probability of attack . In this case, the result of the analysis by XGBoost model is sent to the control unit. (2.3) The third possible output is high probability of DDoS and DoS

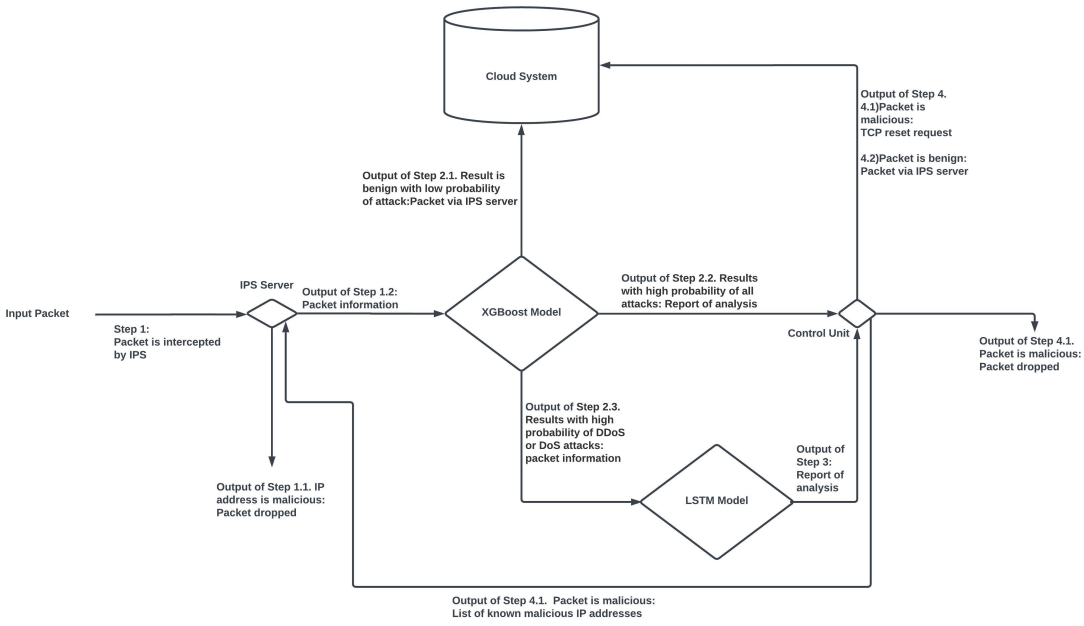

Figure 4.5: The Proposed Overall Approach

attacks, the information of intercepted packet is shared with the LSTM model.

Step 3: The result of analysis by LSTM model (discussed in detail later in section 4.6) is shared with control unit.

Step 4: Control unit decides if intercepted packet is benign or malicious (discussed in detail later in section 4.7). (4.1) If this packet is malicious, TCP reset request (discussed in detail later in section 4.7) is sent to cloud, intercepted packet is dropped, and updated list of known malicious IP addresses is shared with IPS server. (4.2) If this packet is benign, it is sent to cloud system.

Section 4.5 Use of XGBoost in the Approach

In step 2 of the approach intercepted packet's NetFlow data is subjected to analysis by the XGBoost model after normalization and storing the original values in a separate

variable. The actions taken in this step can be divided into the following three steps:

Step 2.1. If the result is benign with low probability of attacks: This result means that the XGBoost model deems the packet to be benign, with probabilities of attack labels being less than 30%. In this case, the XGBoost model sends a signal to the IPS server, informing it of this result and as a consequence, this packet is redirected to the cloud system.

Step 2.2. If the result has high probability of attacks: The result of the analysis by XGBoost model is sent to the control unit. This includes details on original NetFlow data of the packet, probabilities of labels and the opinion of the XGBoost model, as shown later in chapter 7. After this, go to step 4.

Step 2.3. If the result has high probability of DoS and DDoS attacks: The normalized NetFlow data is shared with the LSTM model for an analysis.

Section 4.6 Use of LSTM in the Approach

One significant innovation of this thesis is the introduction of a deep LSTM architecture for identifying DoS and DDoS attacks to protect cloud systems. This architecture includes one LSTM layer, one activation layer and two fully connected layers. Figure 4.6 depicts the architecture for the LSTM. For the NF-UQ-NIDS-v2 dataset, the LSTM layer takes in the 22 attributes and gives 64 output variables. The first fully connected layer takes in 64 input variables and gives them to 15 variables. The second fully connected layer reduces these 15 variables to 3 variables, corresponding to the number of classes. The LSTM model will generate probabilities for each label. The numbers mentioned are specific to Benign, DDoS and DoS attack records for network intrusion, and can be different depending on the attacks considered. The LSTM model has a learning rate of 0.0001 with a cross entropy loss function and an

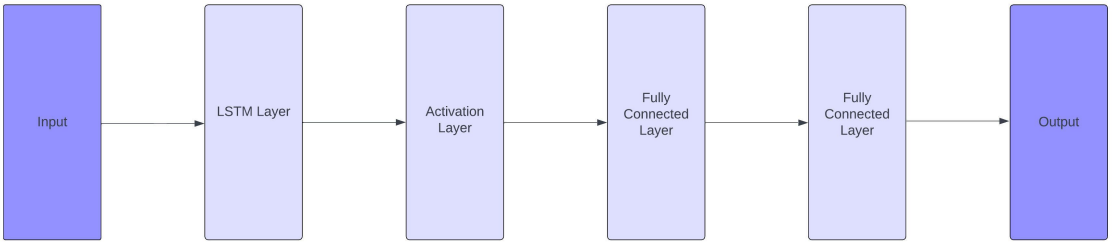

Figure 4.6: Architecture of LSTM model

Adam optimizer. PyTorch framework was used for constructing the custom LSTM model.

An LSTM layer is an RNN layer that learns long-term dependencies between time steps in time-series and sequence data [35]. As a sequential processing unit, the LSTM layer manages the temporal dependencies among the 22 input attributes that make up the LSTM model as a whole. It is appropriate for activities involving sequential data because of its capacity to selectively update information over time, which is very helpful when detecting DoS and DDoS type of attacks. The LSTM layer generates 64 output variables that provide a thorough depiction of the temporal elements taken out of the input. The model gains the ability to learn non-linear features by adding an activation layer between the LSTM layer and the first fully connected layer. The purpose of an activation function is to add non-linearity to the neural network [36]. The activation layer improves the model's overall ability to recognize and learn from the intricate patterns found in the data by allowing it to collect and utilize complicated, non-linear relations present in the data. In situations when the correlations between the input qualities are not strictly linear, the use of activation layer is very helpful. I have increased the model's ability to abstract and change features by employing two fully connected layers. Fully connected Layer give neural networks the ability to approximate any function — known as the Universal Approximation Theorem [37].

With 64 input variables and 15 output variables, the first fully connected layer serves as a bottleneck, reducing the amount of data gathered from the LSTM layer into a more succinct and abstract representation. This intermediary layer lowers the dimensionality of the data while highlighting important aspects. This also facilitates the information flow to the last layer. The second fully connected layer further refines this representation by reducing 15 input variables to 3 output variables, corresponding to the number of classes. By introducing a hierarchical feature selection procedure, this design decision highlights key traits in the data for categorization, allowing the LSTM model to learn intricate patterns, adjust to changing temporal dynamics, capture both short-term and long-term dependencies including non-linear dependencies without overfitting on the dataset, and generalize well to various dataset sequences. Results proving that the effectiveness of this architecture and showing that it does not overfit on the dataset are shown in chapter 5. The model is further improved with a training process using the Adam optimizer and cross-entropy loss, which guarantee convergence towards an ideal solution. These elements have been thoughtfully integrated into the overall architecture, which enhances the model's performance and robustness for identifying DDoS and DoS attacks.

The following steps were taken to develop the architecture of the LSTM model:

Step A. Defining the task: In this step, the nature of the task at hand was decided. Since the application is to identify DoS and DDoS attacks with the LSTM model, the task is to perform a multiclass classification and the architecture of the LSTM model was designed to reflect a classification model, not a regression model.

Step B. Understanding the data: In this step, I analyzed the characteristics of the input data. Factors such as dimensionality, range of values, and relationships within the data were considered. The results of the feature selection process specified in 4.3

were also considered in this step.

Step C. Identifying the input and output features: The input layer of the model was designed to have neurons corresponding to the features, and the output layer was designed to match the desired output format.

Step D. Selecting activation function: In this step, I chose the ReLU activation for hidden layers as the LSTM model is going to perform a multiclass classification task.

Step E. Deciding the number of hidden layers: This step consists of deciding the number of hidden layer for a model. Since the task is to classify network attacks, a deeper architecture was needed to achieve this task. I started from 3 layers: 1 LSTM layer followed by an activation layer and a fully connected layer. I have proved with results that an LSTM model with 4 layers (with 2 fully connected layers) work best for this task.

Step F. Determining the number of neurons in each hidden layer: This is a critical hyperparameter that can impact the capacity and performance of a model. It often requires heavy experimentation and tuning. I selected an initial number of 10 neurons in a fully connected layer.

Step G. Choosing the loss function: In this step, an appropriate loss function is chosen for the model. I have selected the categorical cross entropy loss function as the task is multiclass classification.

Step H. Selecting an optimizer: In this step, I selected the Adam optimizer for this model as the task is multiclass classification.

Step I. Defining Metrics for Evaluation: I have considered accuracy, precision, recall, and F1 score for considering the performance of the LSTM model.

Step J. Building an initial version of the model: In this step, I built an initial version of the LSTM model and trained it on a the training data.

Step K. Testing on test data: Finally, the model built was evaluated on a separate test set that it has never seen. This provides an unbiased assessment of the model's performance.

Step L. Adjusting the model: After evaluating the model's performance on the test data, I adjusted the model by fine-tuning hyperparameters and oversampling the data.

In step 3 of the approach, the LSTM model will perform analysis on selected features of this data. The result of this analysis by the LSTM model will be sent to the control unit. This includes probabilities of labels and opinion of the LSTM model, as shown later in chapter 7.

Section 4.7 Use of Control Unit and TCP Reset Request in the Approach

A significant innovation of this thesis is the proposed use of a special TCP packet, named TCP reset request. Step 4 of the overall approach is taken after the reports from XGBoost model and LSTM model (in case DDoS and DoS attacks are suspected) are received by the control unit for a packet from one IP address. The control unit will look at these reports and make the final decision. The final decision is only based on opinion of XGBoost model when the intercepted packet shows high probability of Brute Force, Infiltration and Bot attacks. The final decision is based on opinion of both models and their probabilities for DDoS and DoS attacks. If the models have different opinions, the opinion of the model showing higher probability for a certain class is taken as the final decision. For example, if XGBoost model shows high probability of intercepted packet being a DDoS attack but LSTM model shows higher probability of the packet being benign, then the control unit will decide that

the packet is benign. If the two models have the same opinion, this opinion is taken as the final decision. Actions taken in this step can be divided into two steps:

Step 4.1. Suspected packet is malicious: In this case, a TCP reset request is sent to the cloud system to terminate the connection and mitigate potential damage, if any, caused by the malicious activity. Details on generation of a TCP reset request are shared later in this section and details on its innovative use in this study is shared in chapter 5. The list of malicious IP addresses is updated to include IP address of the intercepted packet. The packet in question is dropped. The packets from the attacker in transit are automatically dropped as the connection is closed.

Step 4.2. Suspected packet is benign: In this case, control unit will signal IPS server to redirect packet to the cloud system. This packet is encapsulated with TCP protocol before it is redirected to the cloud system. The IP address will not be included in the list of IP addresses and TCP reset request will not be sent to the cloud system.

Steps taken to send a TCP reset request are:

Step a. Create a TCP Packet: In this step, a TCP packet is created specifically to send the cloud system information about the attacker.

Step b. Identify the attacker information: In this step, the information from the report by XGBoost model is used to fill information about the attacker in the TCP packet.

Step c: Send the TCP reset request: Once the packet is constructed, the TCP packet is sent to the cloud system. The cloud system will identify the attacker based on information received from the IPS and will create a TCP reset for the attacker. This TCP reset will have the reset (RST) flag sent to one, and it will include the acknowledgement number and sequence number for the attacker along with informa-

tion like port number, IP address. This step shows the innovative use of TCP reset request as the cloud system will now immediately terminate the connection with the attacker, dropping the packets in-transit to the cloud system and thereby prevent the attack.

Chapter 5

INNOVATION

In this chapter I present the innovations in my thesis. I have two innovations: (1) The custom architecture of the LSTM model, and (2) The use of TCP reset request for preventing 5 kinds of attacks in cloud systems.

One significant accomplishment of this thesis is the incorporation of an activation layer and two fully connected layers in addition to the LSTM layer for the development of the final LSTM model. These layers contribute to enhancing the model and make it capable of learning sequential temporal dependencies and non-linear dependencies in the data, as mentioned in section 4.6.

The results of experiments like oversampling and tuning are presented in section 8.5 of this thesis. The architecture of the final LSTM model does not overfit on the model. This can be shown by results of training different LSTM models with different numbers of fully connected layers in table 5.1. Best number of hidden units were found for each model with different number of layers after extensive testing and their results were compared. All these models had a learning rate of 0.0001, Adam optimizer and cross entropy loss. This proves that my proposed architecture greatly outperforms the other two LSTM models. The reason for the weak performance of 1 fully connected layer of LSTM model is that this model does not have sufficient capacity to understand the intricacies of DDoS and DoS attacks. This is shown by its F1 score of 0.6682. The LSTM model with 3 fully connected layers has sufficient capacity to understand the intricacies of DDoS and DoS attacks, as evident by comparing its performance with the previous model, but this model is too complicated and tends to overfit on the training dataset, which is why the F1 score for this is only 0.8743. The final

LSTM model with 2 fully connected layers has sufficient capacity for understanding the intricacies of DDoS and DoS attacks. This model is not as complicated as the LSTM model with 3 fully connected layers, so this model does not overfit on the training data, as evident by its F1 score of 0.9476.

No. of Fully Connected Layers	**Accuracy**	**Precision**	**Recall**	**F1 Score**
1	92.83%	0.7599	0.6305	0.6682
2	98.05%	0.9191	0.9803	0.9476
3	92.03%	0.8233	0.9568	0.8743

Table 5.1: LSTM with Different Layers

Another innovation of this study is the proposed use of TCP reset functionality to prevent attacks in real time for cloud systems that use TCP protocol in connections, as mentioned in section 4.7. Since popular application layer protocols like HTTP (Hypertext Transfer Protocol), HTTPS (HTTP Secure), FTP (File Transfer Protocol), SMTP (Simple Mail Transfer Protocol), DHCP (Dynamic Host Configuration Protocol), POP3 (Post Office Protocol version 3), Telnet, SSH (Secure Shell), IRC (Internet Relay Chat) and SFTP (SSH File Transfer Protocol) use TCP [38, 39], the use of TCP reset functionality is applicable to many cloud systems. TCP reset allows for an immediate response to identified security threats. As mentioned in section 4.7, when an attack is detected by the Intrusion Prevention System, a TCP reset request is sent to the cloud system. The TCP reset request is a packet that contains information about the attacker. The cloud system will then immediately send a TCP reset to the attacker, interrupting the connection between the attacker and the cloud system. This approach represents a proactive defense mechanism, and aims to disrupt

attacks as early as possible, instead of allowing malicious traffic to communicate with cloud systems. Due to this disruption, the window of opportunity for exploitation is reduced, preventing attacks and mitigating potential damage to cloud systems. Finally, the use of TCP reset allows for a more streamlined and responsive security architecture, and helps conserve resources as resources dedicated to communication with the attacker can be used immediately by the cloud system after being freed.

Chapter 6

ILLUSTRATIVE EXAMPLE

In this section, I illustrate the actions taken by the proposed IPS to protect a cloud system. For the purposes of presenting an illustration that covers more intricate details, I have made two assumptions:(1) The proposed IPS has been deployed to protect a cloud system, and (2) A new packet with attack type 'DoS' is sent to the cloud system from a previously unseen IP address.

Step 1: The packet is intercepted by the IPS. The IP address of this packet is checked to see if it is in the list of malicious IP addresses. Since this is a previously unseen IP address, it will not be in those lists, so the packet is temporarily stored in the IPS server, and NetFlow data of this packet is sent to the XGBoost model for analysis.

Step 2: The packet's NetFlow data is subjected to analysis by the XGBoost model after normalization. The result of this classification would be high probability of 'DoS' attack. Thus, the intercepted packet will not be sent to the cloud system.

Step 3: The result of the analysis by XGBoost model is sent to the control unit. As this result has high probability of 'DoS' attack, normalized NetFlow data of this packet is shared with the LSTM model for an analysis. In this case, the result of analysis by LSTM will show high indication of 'DoS' attack. This report will also be shared with the control unit.

Step 4: This step is taken as soon as the reports from XGBoost model and LSTM

model are received by the control unit. The control unit will generate a TCP reset request for this IP address as all reports indicate that the intercepted packet is not benign. The TCP reset request will be sent to the cloud system. The cloud system will then terminate the connection between the cloud system and this IP address. This IP address is included in the list of malicious IP addresses and shared with the IPS server so that all future incoming packets from this IP address are dropped.

Chapter 7

SIMULATION

In this chapter, I present a simulation showing the reports generated by XGBoost and LSTM models. For the purposes of this simulation, I make two assumptions:(1) The proposed IPS has been deployed to protect a cloud system, and (2) A new packet with attack type 'DoS' is sent to the cloud system from a previously unseen IP address. This packet will be intercepted by the IPS. The IP address of this packet will not be found in list of malicious IP addresses. NetFlow data of this packet will be sent to the XGBoost model. It will analyze this data and generate probabilities for all classes. This result will show a high probability of 'DoS' attack, a report about this packet is sent to the control unit. This report contains the original NetFlow data of the intercepted packet, results of the analysis by XGBoost model and the opinion of XGBoost model, as shown in figure 7.1. Since the result of analysis by XGBoost model shows high probability of 'DoS' attack, the normalized NetFlow data is sent to the LSTM model. LSTM model will look at the top 22 attributes identified in Section 4.3, and give result of its analysis. A report for the same will be generated and sent to control unit. This report will contain the probabilities for all classes and the opinion of the LSTM model, as shown in figure 7.2. As both models have predicted the same attack type correctly with strong confidence, this simulation shows reliability of the proposed framework. The total time taken by both the XGBoost and LSTM models to analyse this packet and generate reports came to approximately 0.004 seconds. The control unit will send a TCP reset request to the cloud for the IP address of this packet and this packet is dropped. The control unit will update the list of malicious IP address to include IP address of this packet and share it with IPS server. The

```
Report from XGBoost model:

IPV4_SRC_ADDR                18.219.193.20
L4_SRC_PORT                          40064
L4_DST_PORT                             80
PROTOCOL                                 6
L7_PROTO                                 7
IN_BYTES                              2474
IN_PKTS                                 21
OUT_BYTES                             4812
OUT_PKTS                                20
TCP_FLAGS                               27
CLIENT_TCP_FLAGS                        27
SERVER_TCP_FLAGS                        27
FLOW_DURATION_MILLISECONDS         4294780
DURATION_IN                           187
DURATION_OUT                          187
MIN_TTL                                 63
MAX_TTL                                 63
LONGEST_FLOW_PKT                      987
SHORTEST_FLOW_PKT                       52
MIN_IP_PKT_LEN                          52
MAX_IP_PKT_LEN                        987
SRC_TO_DST_SECOND_BYTES              2472
DST_TO_SRC_SECOND_BYTES              4812
RETRANSMITTED_IN_BYTES                  0
RETRANSMITTED_IN_PKTS                   0
RETRANSMITTED_OUT_BYTES               987
RETRANSMITTED_OUT_PKTS                  1
SRC_TO_DST_AVG_THROUGHPUT          104000
DST_TO_SRC_AVG_THROUGHPUT          200000
NUM_PKTS_UP_TO_128_BYTES                33
NUM_PKTS_128_TO_256_BYTES                0
NUM_PKTS_256_TO_512_BYTES                4
NUM_PKTS_512_TO_1024_BYTES               4
NUM_PKTS_1024_TO_1514_BYTES              0
TCP_WIN_MAX_IN                       26883
TCP_WIN_MAX_OUT                      26847
ICMP_TYPE                                0
ICMP_IPV4_TYPE                           0
DNS_QUERY_ID                             0
DNS_QUERY_TYPE                           0
DNS_TTL_ANSWER                           0
FTP_COMMAND_RET_CODE                     0

XGBoost results:
[[2.9607463e-07 3.1727807e-09 9.9999952e-01 3.2676176e-10 1.4345963e-07
  7.8791355e-11]]
XGBoost predicts DoS
```

Figure 7.1: Report by XGBoost model

```
Report from LSTM model:
LSTM results:
tensor([[1.5204e-02, 5.6288e-15, 9.8480e-01]], grad_fn=<SoftmaxBackward0>)
LSTM predicts DoS
```

Figure 7.2: Report by LSTM model

```
Cloud: Listening on 127.0.0.1:8888
Cloud: Accepted connection from ('127.0.0.1', 57505)
IPS: TCP reset request sent to Cloud Server to block Attacker 192.168.1.100
Cloud: Received data: TCP Reset Request
Cloud: Blocking attacker 192.168.1.100
```

Figure 7.3: Screenshot Demonstrating Use of TCP Reset Request

cloud system will receive this TCP reset request and block the attacker as shown in figure 7.3. In this figure, address '127.0.0.1' with port '57505' represents the IPS and the packet identified to be malicious belongs to IP address '192.168.1.100'.

Chapter 8

EVALUATION

In this chapter, section 8.1 describes details on evaluation metrics for classification models. Section 8.2 discusses results for the XGBoost model. Section 8.3 discusses results for the LSTM model. Section 8.4 shows comparison of XGBoost and LSTM models with other machine learning models. Section 8.5 discusses intermediate results obtained during training and fine-tuning the models.

Section 8.1 Details on Metrics

One of the most valuable tools for evaluating the performance of a classification model is the confusion matrix [40]. It provides a summary of the model's predictions as compared to the actual ground truth. As shown in figure 8.1, it is a table with 4 different combinations of predicted and actual values [40].

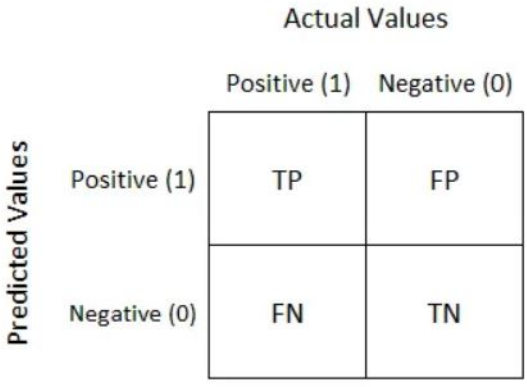

Figure 8.1: Structure of Confusion Matrix

Consider a case where a model predicts positive or negative cases for a certain input to help understand this matrix. TP stands for True Positive and it indicates that the model predicted a positive result accurately. TN stands for True Negative. This indicates that the model accurately predicted a negative result. FP stands for False Positive. This indicates that the model predicted a positive case when the result should have been negative. FN stands for False Negative. In this case, the model inaccurately predicted a negative case. From the confusion matrix, the following performance metrics can be calculated [40]:

1. **Accuracy**: Accuracy measures the overall correctness of the model's predictions. It is calculated as:

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

2. **Precision**: Precision quantifies the model's ability to correctly predict positive instances. It is calculated as:

$$Precision = \frac{TP}{TP + FP}$$

3. **Recall**: Recall is also known as Sensitivity or True Positive Rate. It measures the model's ability to capture all positive instances. It is calculated as:

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

4. **F1 Score**: F1 score is a widely used metric for evaluating the performance of a classification model. It combines precision, and recall into a single value that balances the trade-off between them.

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

These metrics provide insights into different aspects of model performance, which helps to evaluate how useful a model is. Thanks to the 'sklearn' library in Python, I can directly get this metrics from the code, without having to calculate each value in the confusion matrix. Thus, I will showcase the outputs directly with screenshots. Note that the precision, recall and F1 scores shown are weighted average values calculated for all classes. This was done by setting the average type to 'macro' in the code.

Section 8.2 Results for XGBoost Model

The number of training records, number of testing records, accuracy, precision, recall and F1 score values for the XGBoost model are shown in table 8.1. Figure 8.2 shows a code snippet for generating the evaluation metrics. These values were calculated for 'Benign', 'DDoS', 'DoS', 'Infiltration', 'Brute Force' and 'Bot' classes, with 41 attributes as input data and one label as output. This figure also shows how efficient this model is as it took about 2 seconds for it to finish processing over 2.3 million records. The resultant accuracy of 98.05% with high precision, recall and F1 score values proves the reliability of our model.

Figure 8.3 shows the result of simulation done for an individual record. It also shows that the XGBoost model can analyse a single flow in a little over 0.001 seconds with a 16GB CPU. Professional environments use processors several times better than a 16GB CPU. This fact proves the capability of XGBoost model to take action in real time. If the result of the classification is high probability of attack, the XGBoost

```
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average="macro")
recall = recall_score(y_test, y_pred, average="macro")
f1 = f1_score(y_test, y_pred, average='macro')

print(f"XGBoost Results:")
print(f"Accuracy: {accuracy:.8f}")
print(f"Precision: {precision:.8f}")
print(f"Recall: {recall:.8f}")
print(f"F1 Score: {f1:.8f}")
```

```
Testing time for full test set: 2.0740 seconds
XGBoost Results:
Accuracy: 0.99329515
Precision: 0.99651012
Recall: 0.88821764
F1 Score: 0.91498735
```

Figure 8.2: Screenshot of Results of XGBoost Model

```
start_time_single = time.time()
result=model.predict_proba(temp)
end_time_single = time.time()
elapsed_time_single = end_time_single - start_time_single
print(f"Time taken to classify a single record by XGBoost: {elapsed_time_single:.8f} seconds")

print(result)
print(y_test[0])
```

```
Time taken to classify a single record by XGBoost: 0.00100279 seconds
[[9.9999714e-01 6.1391479e-09 2.6850062e-08 1.0630298e-09 2.8103950e-06
  1.3686279e-09]]
0
```

Figure 8.3: Screenshot of Single Analysis by XGBoost Model

model allows our IPS to immediately redirect the suspicious packet to the honeypot and begin the subsequent steps for dealing with it. Conversely, if the packet is deemed to be benign with low probability of attack (less than 30%), the packet is redirected to the cloud system in a matter of milliseconds on a 16 GB CPU, allowing the sender to enjoy an almost seamless communication with the cloud system.

Category	**XGBoost**	**LSTM**
Training Records	9,380,660	19,604,716
Testing Records	2,345,168	2,269,092
Accuracy	99.33%	98.05%
Precision	0.9965	0.9191
Recall	0.8882	0.9803
F1 Score	0.9150	0.9476

Table 8.1: Training and Testing Information of XGBoost and LSTM Models

Section 8.3 Results for LSTM Model

The number of training records, number of testing records, accuracy, precision, recall and F1 score values for the LSTM model are shown in Table 8.1. Figure 8.4 shows a code snippet for generating the evaluation metrics. These values were calculated for 'Benign', 'DDoS' and 'DoS' classes, with 22 attributes as input data and 3 attributes as output. The resultant accuracy of 98.05% with high precision, recall and f1 score values proves the reliability of our model.

Figure 8.5 shows the result of simulation done for an individual record. It also shows that the LSTM model can analyse a single flow in a little over 0.001 seconds. The three values shown in 8.5 are sent to the control unit. The predicted result shown in this figure will also be added in this report.

An interesting phenomenon was observed when training this LSTM model. When

```
accuracy = accuracy_score(all_labels, all_predictions)

precision = precision_score(all_labels, all_predictions, average='macro')

recall = recall_score(all_labels, all_predictions, average='macro')

f1 = f1_score(all_labels, all_predictions, average='macro')

print("Results For LSTM model:")
print(f"Accuracy: {accuracy:.8f}")
print(f"Precision: {precision:.8f}")
print(f"Recall: {recall:.8f}")
print(f"F1 Score: {f1:.8f}")
```

```
Results For LSTM model:
Accuracy: 0.98046884
Precision: 0.91913643
Recall: 0.98032429
F1 Score: 0.94764707
```

Figure 8.4: Screenshot of Results of LSTM Model

```
input_sequence = torch.tensor(x_test_numpy[10000], dtype=torch.float32).unsqueeze(0)
start_time_single = time.time()

out = model(input_sequence)
end_time_single = time.time()

probabilities = torch.softmax(out, dim=1)
predicted_class = torch.argmax(probabilities, dim=1)

print("Values for each class:", probabilities)
print("Predicted class:", predicted_class)
elapsed_time_single = end_time_single - start_time_single
print(f"Time taken to classify a single record by LSTM: {elapsed_time_single:.8f} seconds")
print(y_test_numpy[10000])
```

```
Values for each class: tensor([[9.9237e-01, 5.5843e-04, 7.0671e-03]], grad_fn=<SoftmaxBackward0>)
Predicted class: tensor([0])
Time taken to classify a single record by LSTM: 0.00100207 seconds
0
```

Figure 8.5: Screenshot of Single Analysis by LSTM Model

there was only one fully connected layer that mapped 64 input variables to 3 output variables, an accuracy of 92.83% was observed with 0.7599 precision, 0.6305 recall and 0.6683 F1 score for the same hyperparameters. Thus, I was able to achieve substantially better results without overfitting the model just by adding one fully connected layer.

Section 8.4 Comparison with Other Models

To compare the performance of our models with other models, I trained K Nearest Neighbor (KNN), Support Vector Machine(SVM), Multi-Layer Perceptron (MLP), Extra Trees, Decision Tree and Random Forest classifiers on the NF-UQ-NIDS dataset for multi-class classification. Table 8.2 shows the evaluation metrics of these models and our models, demonstrating how our models are superior when considering all 4 evaluation metrics. Comparing these results with the multiclass Extra Trees model trained by Sarhan at al. in [12], it was observed that my models outperform the multiclass Extra Trees model (96.93%) in terms of accuracy.

Model	**Accuracy**	**Precision**	**Recall**	**F1 Score**
KNN	46.67%	0.4195	0.4765	0.3711
SVM	54.38%	0.4455	0.5188	0.4167
Decision Tree	97.24%	0.7442	0.7436	0.7434
Random Forest	97.76%	0.8154	0.7478	0.7757
MLP	99.19 %	0.9902	0.8615	0.8905
Extra Trees	99.30 %	0.9689	0.8965	0.9192
LSTM	98.05%	0.9191	0.9803	0.9476
XGBoost	99.33%	0.9965	0.8882	0.9150

Table 8.2: Comparison of Models

Section 8.5 Intermediate Results of Training and Fine-Tuning

In this section, I have shared results obtained during the preprocessing, feature selection, oversampling steps and hyperparameter tuning for the XGBoost and LSTM models. Table 8.3 shows intermediate results for splitting the dataset into 3 different ratios for training and testing data respectively, tested on the XGBoost model. Table 8.4 shows the results of selecting different number of features based on their score by the XGBoost model. Table 8.5 shows the results of a LSTM model trained without oversampling and the results of the final model.

Table 8.9 shows the results for different learning rates for the LSTM model. Results for XGBoost model with different learning rates are shown in table 8.6. Results for XGBoost model with different maximum bins are shown in table 8.7. Table 8.8

shows results of training the XGBoost model with different maximum depths. Table 8.10 shows the results of hyperparameter tuning for a LSTM model with 2 fully connected layers.

Figure 8.6 shows the results for training different versions of LSTM, Random Forest and SVM models. In this figure, fc layer stands for fully connected layer. Only the hyperparameters mentioned in this figure are changed, other hyperparameters were kept the same for different versions of the models trained. This figure shows the importance of fine-tuning complexity and hyperparameters of machine learning models when dealing with imbalanced datasets like the NF-UQ-NIDS-v2 dataset. This figure also shows the superiority of the final LSTM model (with 2 fully connected layers) over the other models shown in this figure.

Split	**Accuracy**	**Precision**	**Recall**	**F1 Score**
60-40	96.75%	0.4815	0.5000	0.4904
70-30	95.20	0.9545	0.8216	0.8778
80-20	99.33%	0.9965	0.8882	0.9150

Table 8.3: Results for Splitting Training and Testing Data

Model	**Features**	**Accuracy**	**Precision**	**Recall**	**F1 Score**
LSTM	21	98.32%	0.9148	0.9815	0.9461
LSTM	22	98.05%	0.9191	0.9803	0.9476
LSTM	23	97.93%	0.9161	0.9789	0.9456

Table 8.4: Results for Feature Selection

Model	Oversampled	Accuracy	Precision	Recall	F1 Score
LSTM	No	90.72%	0.7433	0.6488	0.6807
LSTM	Yes	98.05%	0.9191	0.9803	0.9476

Table 8.5: Results for Oversampling

Learning Rate	Accuracy	Precision	Recall	F1 Score
0.1	99.31%	0.9960	0.8859	0.9123
0.2	99.32%	0.9965	0.8872	0.9138
0.3	99.33%	0.9965	0.8882	0.9150
0.4	99.33%	0.9947	0.8892	0.9159

Table 8.6: XGBoost with Different Learning Rates

Maximum Bins	Accuracy	Precision	Recall	F1 Score
90	99.33%	0.9957	0.8879	0.9146
100	99.33%	0.9965	0.8882	0.9150
110	99.33%	0.9961	0.8879	0.9145

Table 8.7: XGBoost with Different Maximum Bins

Maximum Depth	Accuracy	Precision	Recall	F1 Score
5	99.32%	0.9966	0.8873	0.9140
6	99.33%	0.9965	0.8882	0.9150
7	99.32%	0.9964	0.8865	0.9131

Table 8.8: XGBoost with Different Maximum Depth

Learning Rate	Accuracy	Precision	Recall	F1 Score
0.001	82.49%	0.7080	0.9289	0.7686
0.0001	98.05%	0.9191	0.9803	0.9476
0.0002	85.65%	0.6719	0.8768	0.7383

Table 8.9: LSTM with Different Learning Rates

No. of Hidden Units	Accuracy	Precision	Recall	F1 Score
60	95.50%	0.8470	0.9720	0.9005
70	92.85%	0.8264	0.9015	0.8605
78	98.01%	0.9068	0.9810	0.9399
79	98.05%	0.9191	0.9803	0.9476
80	97.99%	0.9171	0.9806	0.9471
90	97.75%	0.9112	0.9780	0.9421
100	92.95%	0.8067	0.9645	0.8698
110	91.84%	0.7513	0.9490	0.8187

Table 8.10: LSTM with Different Hidden Units

Model	Parameter	Accuracy	Precision	Recall	F1 Score
LSTM	1 fc layer	92.83%	0.7599	0.6305	0.6682
	2 fc layers	98.05%	0.9191	0.9803	0.9476
	3 fc layers	92.03%	0.8233	0.9568	0.8743
Random Forest	max depth=2	95.51%	0.4857	0.4647	0.4742
	max depth=3	97.76%	0.8154	0.7478	0.7757
	max depth=4	97.52%	0.7481	0.7935	0.7581
SVM	50 epochs	54.38%	0.4455	0.5188	0.4167
	80 epochs	46.19%	0.4523	0.5015	0.3967

Figure 8.6: Observation in Training Models

Chapter 9

LIMITATIONS

One drawback associated with using NF-UQ-NIDS-v2 dataset is the lack of timestamps, which affects the LSTM model. The reason is that LSTM models are specifically designed to handle sequential data with time-dependent patterns. Without timestamps, the model may lose critical information about the order and timing of events. There is also a possibility that the model's capacity to capture dependencies might be underutilized. Finally, attacks like DoS and DDoS exhibit recurring patterns. Having timestamps in the data would have allowed the LSTM model to better capture such patterns. Fortunately, since all the records in the dataset are individual flows captured over time from the network behaviour, I was able to sufficiently train the LSTM model and obtain reasonable results. While this method works, I believe that having a proper sequential dataset with timestamps will yield better results, when used to train a time-series model. This approach exhibits limitations inherent to use of supervised machine learning, it is a probabilistic approach. I ran a simulation to check how many known attacks are being sent to cloud system. Out of 452089 known attacks, 15336 were sent to cloud system based on results of XGBoost model. This shows that my approach prevents 96.61% known attacks with XGBoost model. LSTM model failed to detect 727 out of 376013 known attacks, approximately 0.19%, which is a remarkable result of preventing 99.81% known attacks, but it still allows some attacks to go to the cloud system. Due to the use of supervised machine learning algorithms, this approach does not guarantee protection from zero day attacks or new types of attacks, and expanding this approach to cover new attacks will require retraining the models on data for new attacks.

Chapter 10

CONCLUSIONS AND FUTURE WORK

This thesis presented an intrusion prevention approach that analyses NetFlow data of incoming packets and takes action in real time to safeguard cloud systems. Results demonstrate that XGBoost and LSTM models show results comparable to the current state-of-art models in terms of accuracy, precision, recall and F1 scores. These models are very fast in terms of time taken for prediction of a single flow and reliable. With the use of TCP reset functionality, the proposed approach shows promise for working in real time. The proposed approach has considered the possibilities of false positives and has incorporated effective measures to minimize the risk of false positives. This work also emphasizes the importance of using specialized models for specific purposes to ensure faster and more reliable results. This approach is applicable for cloud systems using the TCP protocol. Choosing TCP in cloud systems is a good idea for security because it makes sure data is kept intact, delivered reliably, and in the right order. Using TCP aligns with the high standards of cloud systems, emphasizing the need for a secure and trustworthy way of exchanging data.

In the future, I plan to work on collecting network intrusion data. This would be achieved by simulating honeypots and cloud systems in a private server and inviting people to attack them. Data collected will be analyzed by domain experts and compiled into a dataset. My intention here is to cover popular attack vectors for each packet received by honeypot and/or cloud server in sufficient number. This will allow me to have a dataset with timestamps and reasonably balanced number of records across classes, making this dataset ideal for training good time series models. This dataset will be used to train powerful time series models like Transformers, N-HiTS

and Prophet and using them in proposed framework, replacing LSTM model for better results. Future work also includes using unsupervised machine learning in this framework in combination with time series models.

Additionally, I plan to showcase use of a high-interaction honeypot along with machine learning models for real-time prevention of network attacks. High-interaction honeypots are configured to mirror production systems, and designed to give an attacker full reign of an operating system in the event that they are lured into compromising it [41]. A low-interaction honeypot can easily be noticed by a professional hacker, whilst a high-level interactive one cannot be so easily noticed [42]. A medium-interaction honeypot is also detected more easily than a high-interaction honeypot. Adam Doupe et al. showed in [43] using user studies that none of the attackers realized they were inside a honeypot until informed explicitly about the honeypot flag. Thus, when high-interaction honeypot is utilized, the chances of an attacker realizing they are being monitored are very low. This offers a very big tactical advantage as now a network administrator can watch attacker exploit vulnerabilities of the system. This will allow the network administrator to keep track of weaknesses that need to be improved to prevent such attacks in the future. Adding a honeypot to this approach will aid the use of more powerful machine learning models for real time applications. It will also allow use of multiple lists to keep track of IP addresses who are suspicious but not known to be malicious, allowing incoming packets from these IP addresses to be stored in the honeypot and observing their behaviour without harming the cloud system.

REFERENCES

[1] Zeeshan Ahmad, Adnan Shahid Khan, Cheah Shiang, and Farhan Ahmad. Network intrusion detection system: A systematic study of machine learning and deep learning approaches. *Transactions on Emerging Telecommunications Technologies*, 32, 01 2021.

[2] Stephen S. Yau, Arun Balaji Buduru, and Vinjith Nagaraja. Protecting critical cloud infrastructures with predictive capability. In *2015 IEEE 8th International Conference on Cloud Computing*, pages 1119–1124, 2015.

[3] C. Constantinides, S. Shiaeles, B. Ghita, and N. Kolokotronis. A novel online incremental learning intrusion prevention system. In *2019 10th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, pages 1–6, Canary Islands, Spain, 2019.

[4] Amazon Web Services. Aws case study: Animoto, 2014. https://aws.amazon.com/solutions/case-studies/animoto/.

[5] Microsoft. Troubleshoot tcp/ip connectivity, 2023. https://learn.microsoft.com/en-us/troubleshoot/windows-client/networking/tcp-ip-connectivity-issues-troubleshooting.

[6] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 785–794, New York, NY, USA, 2016. Association for Computing Machinery.

[7] Ralf C. Staudemeyer and Eric Rothstein Morris. Understanding lstm - a tutorial into long short-term memory recurrent neural networks. *ArXiv abs/1909.09586*, 2019.

[8] Tutorials Point. Tcp/ip model: The key to seamless internet connectivity and security, 11 Sept. 2023. https://www.tutorialspoint.com/tcp-ip-model-the-key-to-seamless-internet-connectivity-and-security.

[9] Patrik Goldschmidt. Tcp reset cookies–a heuristic method for tcp syn flood mitigation. *Excel@ FIT 2019*, 2019.

[10] M. Tavallaee, E. Bagheri, W. Lu, and A. Ghorbani. A detailed analysis of the kdd cup 99 data set. *Second IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA)*, 2009.

[11] University of New Brunswick Network Security Laboratory. Nsl-kdd dataset. https://www.unb.ca/cic/datasets/nsl.html.

[12] Mohanad Sarhan, Siamak Layeghy, and Marius Portmann. Towards a standard feature set for network intrusion detection system datasets. *Mobile Networks and Applications*, 103:108379, 2022.

[13] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *4th International Conference on Information Systems Security and Privacy (ICISSP)*, Portugal, January 2018.

[14] M. Choras and R. Kozik. Machine learning techniques applied to detect cyber attacks on web applications. *Logic J. IGPL*, 23(1):45–56, 2015.

[15] Michał Choraś and Marek Pawlicki. Intrusion detection approach based on optimised artificial neural network. *Neurocomputing*, 452:705–715, 2021.

[16] Yudai Yamamoto and Shingo Yamaguchi. A method to prevent known attacks and their variants by combining honeypots and ips. In *2022 IEEE 11th Global Conference on Consumer Electronics (GCCE)*, pages 302–305, 2022.

[17] Lily Hay Newman. A year later, that brutal log4j vulnerability is still lurking, 2022. https://www.wired.com/story/log4j-log4shell-one-year-later/.

[18] Wooseok Seo and Wooguil Pak. Real-time network intrusion prevention system based on hybrid machine learning. *IEEE Access*, 9:46386–46397, 2021.

[19] Akhil Krishna, Ashik Lal M.A., Athul Joe Mathewkutty, Dhanya Sarah Jacob, and M. Hari. Intrusion detection and prevention system using deep learning. In *2020 International Conference on Electronics and Sustainable Communication Systems (ICESC)*, pages 273–278, 2020.

[20] Vibha Gupta, Maninder Singh, and Vinod K. Bhalla. Pattern matching algorithms for intrusion detection and prevention system: A comparative analysis. In *2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 50–54, 2014.

[21] Anurag Yadav, Himanshu Gupta, and Sunil Kumar Khatri. A security model for intrusion detection and prevention over wireless network. In *2019 4th International Conference on Information Systems and Computer Networks (ISCON)*, pages 12–16, 2019.

[22] O.A. Sarumi, A.O. Adetunmbi, and F.A. Adetoye. Discovering computer networks intrusion using data analytics and machine intelligence. *Scientific African*, 9, 2020.

[23] S. Reddy and G.K. Shyam. A machine learning based attack detection and mitigation using a secure saas framework. *Journal of King Saud University - Computer and Information Sciences*, 2020.

[24] S. Gamage and J. Samarabandu. Deep learning methods in network intrusion detection: A survey and an objective comparison. *Journal of Network and Computer Applications*, page 102767, 2020.

[25] S.M. Kasongo and Y. Sun. Performance analysis of intrusion detection systems using a feature selection method on the unsw-nb15 dataset. *Journal of Big Data*, 7, 2020.

[26] Sydney Mambwe Kasongo. A deep learning technique for intrusion detection system using a recurrent neural networks based framework. *Computer Communications*, 199:113–125, 2023.

[27] S. Farhat, M. Abdelkader, and et al. A. Meddeb-Makhlouf. Cads-ml/dl: efficient cloud-based multi-attack detection system. *Int. J. Inf. Secur.*, 22:1989–2013, 2023.

[28] Sayantan Guha, Stephen S. Yau, and Arun Balaji Buduru. Attack detection in cloud infrastructures using artificial neural network with genetic feature selection. In *2016 IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing, 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*, pages 414–419, 2016.

[29] R. Poornima, Mohanraj Elangovan, and G. Nagarajan. Network attack classification using lstm with xgboost feature selection. *J. Intell. Fuzzy Syst.*, Jan 2022.

[30] P. Devan and N. Khare. An efficient xgboost–dnn-based classification model for network intrusion detection system. *Neural Computing and Applications*, 32:12499–12514, 2020.

[31] A.S. Khan, Z. Ahmad, J. Abdullah, and F. Ahmad. A spectrogram image-based network anomaly detection system using deep convolutional neural network. *IEEE Access*, pages 87079–87093, 2021.

[32] Amr Attia, Miad Faezipour, and Abdelshakour Abuzneid. Network intrusion detection with xgboost and deep learning algorithms: An evaluation study. In *2020 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 138–143, 2020.

[33] Mohammed Hasan Ali, Bahaa Abbas Dawood Al Mohammed, Alyani Ismail, and Mohamad Fadli Zolkipli. A new intrusion detection system based on fast learning network and particle swarm optimization. *IEEE Access*, 6:20255–20261, 2018.

[34] Yakubu Imrana, Yanping Xiang, Liaqat Ali, and Zaharawu Abdul-Rauf. A bidirectional lstm deep learning approach for intrusion detection. *Expert Systems with Applications*, 185:115524, 2021.

[35] MathWorks. lstmlayer long short-term memory (lstm) layer for recurrent neural network (rnn). www.mathworks.com/help/deeplearning/ref/nnet.cnn.layer.lstmlayer.html.

[36] Pragati Baheti. Activation functions in neural networks, Nov 15, 2020. https://www.v7labs.com/blog/neural-networks-activation-functions.

[37] Adam Green. A guide to deep learning layers, Nov 15, 2020. https://towardsdatascience.com/a-guide-to-four-deep-learning-layers-225c93646e61.

[38] geeksforgeeks. Protocols in application layer. https://www.geeksforgeeks.org/protocols-application-layer/.

[39] geeksforgeeks. Application layer protocols in tcp/ip, 11 Sept. 2023. https://www.geeksforgeeks.org/application-layer-protocols-in-tcp-ip/.

[40] Sarang Narkhede. Understanding confusion matrix, 2018. https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62.

[41] Chris Sanders and Jason Smith. Chapter 12 - using canary honeypots for detection. In Chris Sanders and Jason Smith, editors, *Applied Network Security Monitoring*, pages 317–338. Syngress, Boston, 2014.

[42] S. Ravji and M. Ali. Integrated intrusion detection and prevention system with honeypot in cloud computing. In *2018 International Conference on Computing, Electronics & Communications Engineering (iCCECE)*, pages 95–100, Southend, UK, 2018.

[43] Siddhant Bhambri, Purv Chauhan, Frederico Araujo, Adam Doupé, and Subbarao Kambhampati. Using deception in markov game to understand adversarial behaviors through a capture-the-flag environment, Nov 09 2022.