Network Intrusion Detection Using Machine Learning Processes

Akshatha Pramod
Electrical Engineering
San Jose State University in California
San Jose, CA, 95195, U.S.A
email: pramodakshatha@gmail.com

Meghna Shankar
Electrical Engineering
San Jose State University in California
San Jose, CA, 95195, U.S.A
email: megan.shanx@gmail.com

Nader F. Mir
Electrical Engineering
San Jose State University in California
San Jose, CA, 95195, U.S.A
email: nader.mir@sjsu.edu

Abstract—Network security methods must work towards preventing data breaches as we have entered the growing era of technology. This paper provides a work-in-progress computer network security study in which the network's security capability has been enhanced by using Machine Learning (ML)-assisted intrusion detection system. The goal of the paper is to present a study on improving network intrusion detection systems by incorporating machine learning techniques. We specifically focus on the accuracy and overall performance of two ML algorithms, Random Forest and XGBoost. We introduce this study when ML is incorporated in an existing network infrastructure specifically within the network monitoring and analysis components. The network performance is centered around the two algorithms. These algorithms are deployed on an available dataset named the ToN_IoT and compared based on accuracy and overall performance thus providing an analysis of the effectiveness of these models in detecting network intrusions.

Keywords—Network Security; Network Intrusion Detection System; Machine Learning; XGBoost; Random Forest; Recurrent Neural Networks; ToN_IoT Dataset.

I. INTRODUCTION

A computer network is a backbone that involves preventing unauthorized access and protecting the integrity of the information and allocation of resources. With the growing dependence on technology, the need for protection of networks becomes essential against data breaches, phishing of information and various malware [1]. The traditional methods of network intrusion detection are not real-time based thus there is a need for the integration of artificial learning processes, such as Machine Leaning (ML). This not only makes the detection system real-time but also makes the system adaptable to learn from previous actions in detecting network breaches and in analyzing the symptoms of the occurrence of an attack. ML integration into security systems makes organizations stay safe and secure with a preventive defense ready [2].

A Network Intrusion Detection System (NIDS) provides continuous network monitoring across its network territory including the cloud infrastructure to detect malicious activity like policy violations, or data exfiltration (a cybercriminal stealing data from personal or corporate devices). There can be two types of architectural models of Artificial Intelligence (AI) based NIDS: *Standalone* and *hybrid*. Standalone models are designed based on historical data of network attack attacks. Hybrid NIDS combines both anomaly-based and signature-based which implies it works on both trained and untrained datasets.

NIDS plays a vital role in protecting computers against malicious attacks and can be classified based on its detection techniques as *signature-based* detection or *anomaly-based* detection. The signature-based detection technique is significant in use if the type of attack is previously known, as it is based on comparing the attack to a list of known attacks. However, this method is not effective in determining previously unknown attacks. Anomaly-based detection techniques can determine previously unknown attacks. It works based on classifying normal and abnormal behavior of the system. New generation NIDS comes with the combination of both detection techniques; this is often known as hybrid [4].

There are certain challenges, such as *false alarms*, *response time*, *unbalanced dataset* and *low detection rates*, a NIDS may face. False alarms are often triggered based on how the system is configured and range anywhere from 2% to 90%. The response is coupled with the traffic volume and its complexity, alert prioritization, network latency and limited infrastructure and resources [4]. While unbalanced datasets can be a challenge for some detection techniques, it does not typically affect anomaly-based detection, where the model is trained on one-class data corresponding to normal network behavior. The low detection rates challenge affects the overall security as it causes a system to fail to detect threats due to insufficient signature coverage, improper signature tuning, limitations in detecting anomalies and insufficient resources.

A confusion matrix helps in evaluating the performance of each of the security models. The use of Artificial Intelligence (AI) in intrusion detection systems has evolved from rule-based systems to more sophisticated ML and deep learning systems [5][6][9]. ML systems are based on decision trees, Support Vector Machines (SVM) and random forest [7]. Deep learning involves using Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN).

The rest of this paper is organized as follows. In Section II, we present a detailed methodology on how ToN_IoT datasets are acquired and pre-processed. In Section III, we present detailed results of our analysis, and in Section IV, a conclusive statement is presented.

II. PROPOSED METHODOLOGY

The proposed methodology for improving NIDS involves a multi-step process that starts with robust dataset creation and preparation. This is followed by selecting and tuning machine learning models, and finally, evaluating their performance.

Each stage is crucial to ensure the accuracy and reliability of the NIDS.

A. Dataset Creation

The process is depicted in Figure 1 and begins with the collection of raw network packets from various sources, including sensors, Internet of Things (IoT) devices, network traffic samples, and Operating System (OS) logs. Such packets are captured using a sniffing tool and are stored in Packet Capture (PCAP) format. The collected packets are then converted into a standardized format and stored in the Telemetry over Networks Internet of Things (ToN_IoT) database, which is publicly available and serves as a benchmark for evaluating NIDS. In the figure, this process starts with the sniffed packets being fed into a Format Converter, which standardizes them before they are stored in the ToN_IoT dataset. The pre-processed dataset is then used in various ML tasks, leading to the selection and deployment of an optimal ML model. The compilation of this dataset was recorded by a team at University of New South Wales (UNSW) Canberra [8].

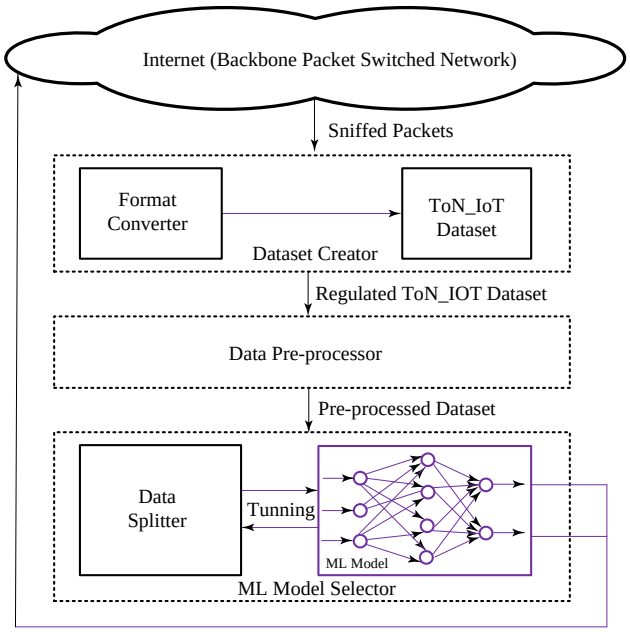

Figure 1. Process of data collection leading to the selection and deployment of an ML model.

B. Data Pre-processor

After a comprehensive data collection leading to the ToN_IoT dataset, data must be loaded to a file system (Google Drive) for processing. This also involves combining multiple *Comma-Separated Values* (CSV) files into a single file. A CSV file uses a comma to separate values and store tabular data (numbers and text) in plain text. A separate CSV file is used as a test dataset. It is important to note that the test dataset, which is kept separate to ensure the integrity of the model evaluation, is not combined with the training data. Instead, a distinct CSV file is used exclusively for testing the model after it has been trained.

As a first step, *data cleaning* is asserted by using *removal techniques* to eliminate data anomalies. Removal of various columns that are trivial IP address ports, HTTP-specific features and Uniform Resource Identifier (URI) versions. Columns that contained null values for the majority of the population, columns that contained singular values and columns that proved to be a non-contributing attribute for training the model were removed as part of the data cleanup.

In the second step, *data transformation* is applied using a *label encoder* to convert label categories to numerical values and formats. Examples of this process include transforming High, Medium, and Low into 0, 1, and 2 respectively. This allows us to represent values as binary vectors and create a new column for each category accordingly. Similarly, normal, Denial of Service (DoS), Distributed Denial of Service (DDoS) and backdoor types of attacks are also converted to numerical values.

Data normalization phase is asserted in the third step. To standardize features, we utilize the *standard scalar* technique which involves the elimination of the mean and scaling it to unit variance [8]. The implementation of a standard scalar ensures feature consistency across the dataset. One of the actions at this step is the conversion of varied data formats to a standardized format, such as Comma-Separated Values (CSV) (a text file format that uses commas to separate values, format for uniform processing). The ultimate purpose of this step is to ensure that no single feature dominates others due to scale differences.

Finally, in the fourth step, *feature selection* to choose the top K features based on the scores from a statistical test is applied. Utilization of *select KBest* identifies the most impactful features for modelling and focuses on the most relevant features for modelling. The combination of the above four steps results in the preprocessed dataset which is a clean, normalized, and optimized dataset ready for analysis or model training. It also improves model accuracy and efficiency by focusing on relevant data.

C. ML Model Selector

The bottom block of Figure 1 shows the process of ML model selection for training and learning purposes involving the following four steps:

Step 1 – **Data Splitter:** involves splitting the dataset into a "training set" for learning, and a "testing set" for evaluation. The training set is usually used to understand the patterns and relationships between the different data points obtained to make certain predictions. A testing set is generally used to evaluate the trained model's performance.

Step 2 – **ML Algorithm Parameter Tuning:** considers efficient ML algorithms such as Random Forest [7], XGBoost [9], or Recurrent Neural Networks (RNN)[9]. The Random Forest algorithm handles nonlinear data well with multiple decision trees to prevent overfitting. It is good for large datasets and integrating diverse data features, and it is suitable for understanding complex network behaviors. XGBoost

algorithm excels in sequential training and is efficient with large datasets. It enhances feature recognition, suitable for detailed ToN_IoT analysis. RNN are best for sequential data, capturing time-dependent patterns without extensive manual feature engineering. It integrates well with other models like Convolution Neural Networks (CNN), enhancing its capability for analyzing complex network traffic and anomaly detection.

Step 3 – **Performance Analysis**: optimizes the ML algorithm parameters to improve model performance. It assesses models using metrics such as accuracy and precision on the testing set. It also carries out iteration for optimization if the performance is unsatisfactory and adjusts parameters.

Step 4 – **Best Model Detected and Deployed**: select the model that best meets performance criteria for deployment.

III. ANALYS AND RESULTS

In this section, we present results and analysis conducted using the developed NIDS based on various ML models implemented through Python-based programming, TensorFlow, PyTorch, and other supporting tools. The analysis focuses on evaluating the intrusion detection capabilities in finding and mitigating potential security threats in a simulated network environment.

A. Data Preparation

The ToN_IoT dataset was used for training and testing NIDS models. The data was first preprocessed to ensure its compatibility with the machine learning algorithms, also involving normalization, feature extraction, and the handling of all the missing values to optimize the performance of the model. The main segment of the script for this task is in Phyton and shown in Figure 2. Next, we will explain some important segments of this script.

The script imports required Python libraries, such as pandas for data orchestration, OS which is used for interacting with the operating system, sklearn for machine learning tasks, XGBoost for model training, and matplotlib and seaborn for visualization. The training_data_folder describes the path to the folder on Google Drive where the training data is stored. This setup automatically assumes that one is using Google Colab and has mounted your Google Drive. The script also creates a detailed list of each one of the CSV files in the specified directory. It uses os. listdir() to get a list of files in the directory and filter this list to include only files that end with .csv. An empty list of train_data_frames is initialized to store each DataFrame.The script loops over each CSV file path in train_csv_files reads the CSV file into data frames using the pd.read command, and appends each data frame to the list train_data_frames. Finally, with Concatenate DataFrames, all the DataFrames stored in train_data_frames are concatenated into a single DataFrame called all_data using pd. concat(). This results in a single data frame containing all the data from the various files, making it easier to perform further data preprocessing or analysis.

```
import pandas as pd
import os
from sklearn.model_selection import train_test_split
import xgboost as xgb
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Path to the folder containing your training data
training_data_folder = '/content/drive/My Drive/training_data'

# List all CSV files in the training data folder
train_csv_files = [os.path.join(training_data_folder, file)
                   for file in os.listdir(training_data_folder) if file.endswith('.csv')]

# Initialize an empty list to store each training data DataFrame
train_data_frames = []

# Loop through the list of CSV files, read each one and append to the list of DataFrames
for file in train_csv_files:
    df = pd.read_csv(file)
    train_data_frames.append(df)

# Concatenate all data frames into a single DataFrame
all_data = pd.concat(train_data_frames, ignore_index=True)

# Concatenate all training DataFrames in the list into one DataFrame
train_data = pd.concat(train_data_frames, ignore_index=True)

# Read the test data into a DataFrame
test_data_path =
    '/content/drive/MyDrive/final_proj_testdataset/train_test_network (1).csv'
test_data = pd.read_csv(test_data_path)

# Drop unnecessary columns from both training and testing datasets
columns_to_drop = [
    'src_ip', 'dst_ip', 'src_port', 'dst_port',  # IP addresses and ports
    'service',  # Service
    'http_method', 'http_uri', 'http_request_body_len',
    'http_response_body_len', 'http_status_code',
    'http_user_agent',
    'http_orig_mime_types', 'http_resp_mime_types',
    'weird_name', 'weird_addl', 'weird_notice'
]

train_data.drop(columns=columns_to_drop, errors='ignore', inplace=True)
test_data.drop(columns=columns_to_drop, errors='ignore', inplace=True)

# Separate features and target variables
x_train = train_data.drop(['label', 'type', 'uid'], axis=1, errors='ignore')
y_train = train_data['type']  # Assuming 'type' is the target variable
x_test = test_data.drop(['label', 'type'], axis=1, errors='ignore')
# Assuming 'type' is the target variable in the test set as well
y_test = test_data['type']
```

Figure 2. Script for preprocessing of data.

B. Model Selection and Training

Three ML models, XGBoost, Random Forest, and Recurrent Neural Network (RNN), were selected based on their capability to manage complex and unbalanced data as follows. XGBoost boosts effectively to tackle one as well as the other binary and multi-class classification challenges. It is known for its efficiency in feature handling and its accuracy in data classification. The Random Forest model was chosen because it is known for utilizing a combination of decision trees to enhance accuracy and reliability while also minimizing the risk of overfitting. The RNN model was selected for its potential to process sequences and is used to identify temporal patterns in network traffic. Despite its capabilities, the RNN algorithm was found to be less effective most likely because of the specific characteristics of the data and the sensitivity of the model's settings and training methodology.

The three models have been applied to a dataset for detecting various types of cybersecurity threats. The results provide a clear comparison of overall accuracy, performance,

and issues encountered for each model, helping to understand their effectiveness and look at the areas that may require further development or reconsideration.

C. Comparison of Models and Performance Analysis

The three machine learning models, XGBoost, Random Forest, and Recurrent Neural Networks () have been applied to a dataset under the process of detecting various types of cybersecurity threats. The result of this process helps us understand the effectiveness of each model and look at the areas that may require further development or reconsideration. The two most effective models, Random Forest and XGBoost models are the focus of our analysis. Table 1 shows a statistical comparison of the distribution of attacks for both Random Forest and XGBoost models. On Normal Network Behavior, both Random Forest and XGBoost demonstrated high accuracy in detecting normal behavior. The Random Forest model showed accuracy in classifying Normal Network Behavior (49,891 instances) and Password attacks (19,880 instances). It, however, struggled more with distinguishing between DDoS and DoS attacks, as well as between Injection and Cross-Site Scripting (XSS) attacks.

TABLE 1 THE DISTRIBUTION OF THE ATTACKS DETECTED BY RANDOM FOREST AND XGBOOST

Category	Random Forest	XGBoost	Description
Normal Network Behavior	24.00%	25%	A significant portion of network traffic is benign.
Scanning	19.70%	12.10%	High incidence of network scanning activity.
Password	11.00%	11%	A noticeable amount of password-related security incidents.
DDoS	10.30%	10.70	Considerable number of Distributed Denial of Service events.
DoS	9.60%	10.7%	Frequent detection of Denial-of-Service attacks.
Injection	9.10%	9.4%	Attacks involving malicious data sent to the system.
XSS	9.10%	8.9%	Cross-site scripting vulnerabilities are prevalent.
MitM	0.40%	0.3	Man-in-the-middle attacks are less commonly identified.
Backdoor	0.20%	0.30	Least frequently predicted type of attacks.
Ransomware	6.60%	7.7%	Significant identification of ransomware, though less common than other threats like scanning or password attacks.

The XGBoost model displayed an improvement in accuracy and notably excelled in categorizing Backdoor, DDoS, and DoS attacks. On DDoS and Dos attacks, Random Forest identified DDoS 19,677 times and DoS 19,864 times, while XGBoost correctly identified DDoS 19,479 times and DoS 19,221 times. Random Forest had an edge in distinguishing between DDoS and DoS. XGBoost on the other hand demonstrates better accuracy in distinguishing Ransomware and makes fewer errors when categorizing Normal Network Behavior and attack scenarios. The decision between the two models might rely on the needs of the security system and which types of attacks are crucial to detect with precision.

On the trend of accuracy improvement, there is a consistent improvement in accuracy for both Random Forest and XGBoost models as the depth of trees increases as seen in Figure 3.

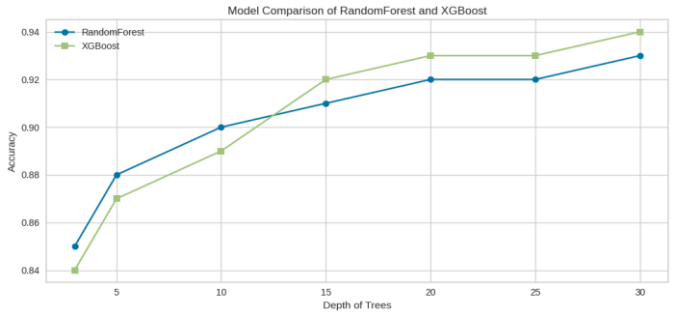

Figure 3. Model comparison of Random Forest and XGBoost based on depth of tree and accuracy.

At lower depths, the accuracy of the two models converges. However, as tree depth increases, a divergence emerges with XGBoost marginally outperforming the Random Forest model, suggesting a better handling of model complexity. Both models exhibit accuracy improvement beyond certain depths—around 15 for Random Forest and 20 for XGBoost. This implies that there is an optimal tree depth for each model beyond which the incremental gains in accuracy become marginal.

XGBoost achieved an overall accuracy of 91%, outperforming Random Forest, which achieved an accuracy of 86%. The RNN model showed an overall accuracy of 9.43%, which is significantly lower than both Random Forest and XGBoost, indicating that it struggled more with detecting the various types of attacks.

IV. CONCLUSIONS

This paper presented an analysis of a NIDS system and the advantage it provides in detecting network attacks using a pre-existing dataset. The results showed the performance of different models in terms of accuracy, precision and recall and how well it ats on detecting different attack patterns. Focused models like XGBoost and Random Forest showed good accuracy and are useful in detecting attack patterns. These models, in the meantime, faced some challenges in detecting attack types such as DoS and DDoS. The results proved the effectiveness of integrating ML-driven models into real-time cybersecurity frameworks.

REFERENCES

[1] T. Ahmed, B. Orehounig, and R. W. Brennan, “Artificial intelligence for network intrusion detection: Review and future trends,” Expert Systems with Applications, vol. 92, pp. 417-430, 2017.

[2] S. Garcia, M. Grill, J. Stiborek, and A. Zunino, “An empirical comparison of botnet detection methods,” Computers & Security, vol. 45, pp. 100-123, 2014.

[3] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, “A detailed analysis of the KDD CUP 99 data set,” in Proceedings

of the Second IEEE Symposium on Computational Intelligence for Security and Defense Applications, pp. 1-6, 2009.

[4] Sarker, I.H. Machine Learning: Algorithms, Real-World Applications and Research Directions.SNCOMPUT.SCI. 2, 160 (2021). https://doi.org/10.1007/s42979-021-00592-x

[5] C. Cortes and V. Vapnik, “Support-vector networks,” Machine Learning, vol. 20, no. 3, pp. 273-297, 1995.

[6] M. A. Hearst, S. T. Dumais, E. Osman, J. Platt, and B. Scholkopf, “Support vector machines,” IEEE Intelligent Systems and their Applications, vol. 13, no. 4, pp. 18-28, 1998.

[7] L. Breiman, “Random forests,” Machine Learning, vol. 45, no. 1, pp. 5-32, 2001.

[8] J. Xie, R. P. Gopalan, and A. A. Ghorbani, “A novel anomaly detection algorithm for sensor networks,” IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), vol. 40, no. 5, pp. 553-566, 2010.

[9] N. Bhattacharyya and J. Kalita, “DANNIDS: A dynamically adaptive neural network-based intrusion detection system,” Expert Systems with Applications, vol. 56, pp. 42-56, 2016.