**SDN Intrusion Detection**

Team number: 11

Team members: Ruthwik Bommenahalli Gowda

Rutuja Manohar Kute

**INTRODUCTION:**

Software Defined Networking (SDN) has become an important technology in the field of networking due to its flexibility, scalability, and easy management. However, with the increase in the complexity and scale of network infrastructure, the number and types of network attacks have also increased, making it essential to have a robust intrusion detection system in place to identify and prevent these attacks. Intrusion Detection Systems (IDS) and Prevention Systems (IPS) are critical defense tools that aid network users in protecting themselves against online threats. As technology advances and the demand for network usage grows, the implementation of IoT, Cloud and SDN has made it easier for users and organizations to access services and data as per their requirements. However, the increasing use of these networks has also led to an increase in cyber threats. Cybercriminals use various methods to inject malicious traffic into SDN and steal sensitive information. One way to detect network attacks in SDN is through traffic monitoring. This involves capturing real-time traffic data daily and analyzing it for any suspicious or malicious activity. In this project, we aim to develop an intrusion detection system using machine learning algorithms to identify various types of network attacks and classify them into different categories.

**Motivation and Goals:**

The primary motivation behind developing an accurate and efficient network intrusion detection system for SDNs is to ensure the protection of sensitive information against various online threats. As the use of SDNs continues to grow, the risk of cyber-attacks also increases, and it becomes imperative to have a robust security system in place to detect and prevent such attacks. The goal of this project is to explore and evaluate the performance of various machine learning algorithms in detecting network intrusion in an SDN. Machine learning algorithms can learn from patterns in the data and can be trained to accurately identify several types of network traffic, including normal traffic and traffic that indicates malicious activity. By applying these algorithms to the data collected from the SDN, we aim to achieve high accuracy, precision, and recall while minimizing false positives and false negatives.

**Dataset Description:**

For this project, we will be using the Network Intrusion Detection System (NIDS) 2017 dataset, which is a publicly available dataset frequently used in research. The dataset was collected in a lab setting using various attack tools such as Metasploit, Nmap, and Hping3, and it includes a tabular record of everyday real-time traffic. The dataset contains 79 features, where one feature denotes a qualitative attribute, and the remaining 78 features denote quantitative attributes. The dataset was originally a Packet Capture file (PCAP).

The dataset includes information about different types of attacks such as DDoS, web attack brute force, web attack XSS, web attack SQL injection, and benign traffic. There are a total of 1,188,333 rows of observations of network incursion and whitelisted traffic. The dataset contains 798,322 instances of benign traffic, 383,439 instances of DDoS attacks, 4,550 instances of web attack brute force, 1,962 instances of web attack XSS, and 60 instances of web attack SQL injection.

In this project, we will be using this dataset to train machine learning models to classify network traffic into different categories and identify potential attacks. The dataset will be pre-processed, and feature selection techniques will be used to select the most important features for the classification task

A picture containing graphical user interface

Description automatically generated

A page of a book

Description automatically generated with low confidence

**Methodologies:**

**Data Processing:**

The data processing part of the project begins by importing necessary libraries such as numpy, pandas, seaborn, and sklearn. The data is loaded using pandas' read\_csv function and stored in a DataFrame for ease of manipulation.The next step involves exploring the data using various methods. The shape of the DataFrame is determined using the shape function, which returns the number of rows and columns present in the dataset. In this case, the dataset has 1,188,333 rows and 80 columns. The columns containing null values are then determined using the isnull() function. However, since dropping null values would lead to a significant loss of data, the missing rows are dropped using the dropna() function. The 'Unnamed: 0' column is also dropped as it contains redundant information. After removing the missing values, the data is stored in a new DataFrame called 'df\_1'. The next step involves encoding the categorical variables using LabelEncoder and scaling the data using MinMaxScaler to normalize the range of values. This ensures that all features are equally weighted during the modeling process.

Finally, the preprocessed data is split into training and testing sets using the train\_test\_split function from sklearn's model\_selection module. The processed data is now ready for modeling using various classification algorithms such as DecisionTreeClassifier, GaussianNB, RandomForestClassifier, LogisticRegression, and XGBClassifier. The performance of each model is evaluated using accuracy\_score, confusion\_matrix, and learning\_curve functions.

The data set has the following Columns:

A close-up of a document

Description automatically generated

Sample Data:

Table

Description automatically generated

Graphical user interface, application

Description automatically generated with medium confidence

**Feature Selection:**

In the given code, feature selection is performed on the input dataset. The steps followed are as follows:

Encoding the Class column: The target variable 'Class' is encoded using the LabelEncoder function to convert the categorical variable to numerical values.

Examining Correlation of Features: The correlation matrix of the numerical features is computed and the correlation coefficients between the target variable 'Class' and the input features are plotted as a bar graph. This graph shows the correlation between each feature and the target variable.

Chart

Description automatically generated

Chart, funnel chart

Description automatically generated

Chart, scatter chart

Description automatically generated

Dropping Columns which are highly correlated: The features which are highly correlated with each other (correlation coefficient greater than 0.8) are identified and removed from the input dataset using the drop() function.

Overall, feature selection is an important step in machine learning to identify the most relevant input features for building a predictive model. The correlation matrix is a useful tool to visualize the correlation between each input feature and the target variable. By removing the highly correlated features, we can reduce the dimensionality of the input dataset and improve the performance of the machine learning model.

**Visualization:**

The first step is to compute the correlation matrix using the corr() function from pandas, which returns a square matrix containing the Pearson correlation coefficients between each pair of features.

Next, the seaborn library is used to create a heatmap of the correlation matrix. The sns.set() function is used to set the font size to 1.10 for the first plot and 4 for the second plot. The plt.figure() function is used to set the size of the figure to 30x30 for the first plot and 60x60 for the second plot.

The sns.heatmap() function is then used to create the heatmap. The arguments vmax, linewidths, square, annot, cmap, and linecolor are used to customize the appearance of the heatmap. vmax specifies the maximum value for the color scale, linewidths sets the width of the lines separating each cell in the heatmap, square makes the cells square-shaped, annot displays the correlation coefficients in each cell, cmap sets the color scheme to "viridis", and linecolor sets the color of the lines separating each cell to white.

Chart

Description automatically generated

Finally, some columns are dropped from the original dataset using the drop() function from pandas, and the correlation matrix is recomputed and visualized using the same code as before.

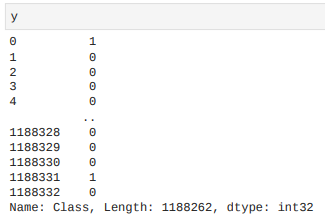
Graphical user interface, chart

Description automatically generated

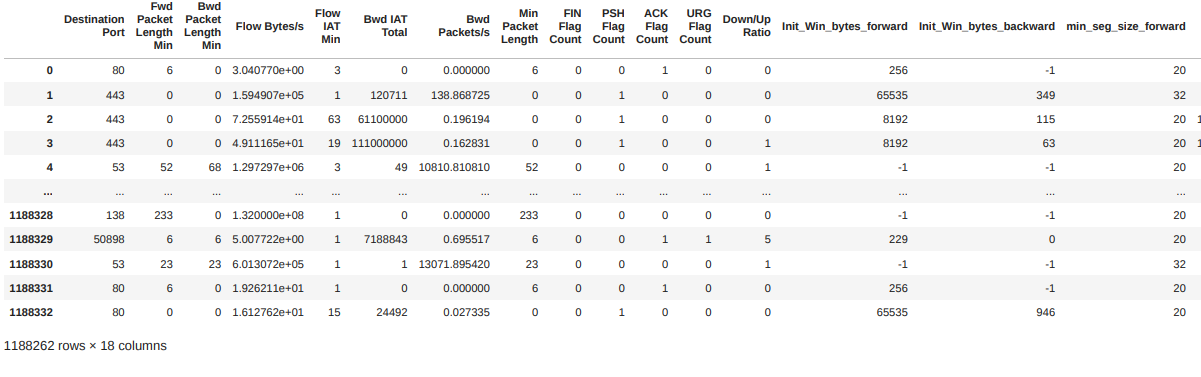
In summary, two heatmaps of the correlation matrix for a dataset are created, with the second heatmap showing the effects of dropping some columns from the original dataset. This visualization can help identify which features are strongly correlated with each other, and which features have little to no correlation with the target variable.

**Train Test Split and Scaling:**

The data is split into the target variable y, which is the 'Class' column, and the input variables X, which are all the remaining columns of the dataframe 'df', except for the 'Class' column.



Next, the input variables X are scaled using a method that is not shown in the code snippet provided. Scaling is a preprocessing step that is often used to normalize the range of input variables. This is done to ensure that all input variables are on the same scale, which can help improve the performance of machine learning models.



Finally, the data is split into training and testing sets using the train\_test\_split function from the scikit-learn library. The training set is used to train the machine learning model, while the testing set is used to evaluate its performance.

**Normalization of Data:**

The data is first clipped to a threshold value of 1000 using the clip method of pandas DataFrame. This is followed by normalization of the data using the MinMaxScaler method from the sklearn.preprocessing module.

The MinMaxScaler scales all the features to a range between 0 and 1. This is done to ensure that all features are on a similar scale, which can help in the training of the machine learning model.

Finally, the data is split into training and testing sets using the train\_test\_split method from the sklearn.model\_selection module. The training set contains 80% of the data, while the testing set contains the remaining 20%. The random\_state parameter is set to 42, which ensures that the same split is obtained every time the code is run.

**Classes of Model:**

**Decision Tree:**

The supervised machine learning algorithm called Decision Tree uses a set of principles to make judgments, much like how people do. This machine learning classification algorithm can be thought of as being created to make decisions. The model is typically said to forecast the class of the previously unseen input, but in reality, the algorithm must choose the class to be assigned. Decision trees are otherwise called CART since they could perform both classification and regression. The basic idea behind it is to keep building yes/no questions and branch out, building a tree until all the datapoints and conditions are pin-pointed and decided. A question inevitably spilts the dataset in terms of features and creates new branches or nodes. The image below explains the dataset of a family budgets and is eventually split according to the question posed

Diagram, table

Description automatically generated

Decision tree classifier is the first model used to predict classes based on a dataset. The maximum depth of the tree is defined as 3 and the model is trained using the training data. After training, predictions on the test data are made with an accuracy of 96.96%. A confusion matrix is created to understand the classification of the predicted classes. The confusion matrix showed that the model performed well for most classes, except for one.

Table

Description automatically generated with low confidence

Calendar

Description automatically generated

**Learning Curve:**

Machine learning requires lots of monitoring, data preparation, and experimentation, especially if it’s a new project. In all that process, learning curves play a fundamental role.

The learning curve typically shows how the model's performance, often measured as the error rate or accuracy, changes as the amount of training data increases. In general, a learning curve will show that a model's performance improves as it is trained on more data, but the rate of improvement may slow down as the model approaches its optimal performance.

Learning curves are useful for identifying whether a model is underfitting or overfitting the data. An underfit model will have a high error rate on both the training and test sets, indicating that it is not capturing the underlying patterns in the data. An overfit model, on the other hand, will have a low error rate on the training set but a high error rate on the test set, indicating that it is fitting the noise in the training data and not generalizing well to new data.

Chart, surface chart

Description automatically generated

**Hyperparameter Tuning:**

To improve the performance of the model, RandomizedSearchCV is used to find the optimal hyperparameters. The hyperparameters are tuned, including the maximum depth of the tree, minimum samples split, and minimum samples leaf. After tuning the hyperparameters, the model is trained again, and predictions are made on the test data. An accuracy of 99.70% is found, which was an improvement over the previous model's accuracy.

Cross Validation:

Cross-validation is used to evaluate the performance of the model. A 5-fold cross-validation is used to evaluate the model's performance. The cross-validation scores showed that the model had a high accuracy across all folds, and the average cross-validation score was 99.69%.

The model performed well, and we achieved an accuracy of 99.70%. The learning curve and cross-validation scores showed that the model is not overfitting or underfitting the data.

Graphical user interface, text, application

Description automatically generated

**Random Forest Classifier:**

The Random Forest Classifier model was built using the Scikit-learn library in Python. The model was trained on the provided training set with 100 decision trees and a maximum depth of 2. The model was then evaluated on the testing set, and it achieved an accuracy of 94.53%. The accuracy score indicates that the model performed relatively well in predicting the target variable.

Graphical user interface, text

Description automatically generated

**Naive Bayes Classifier:**

Bayes Theorem is a way to measure the probability of a subset of a data belonging to a particular class considering the possessed previous knowledge. Here

Bayes Theorem : P(class|data) = (P(data|class) \* P(class)) / P(data)

Here P(class|data) is the probability described before. An approach for categorizing issues with binary (two classes) and multiple classes is called Naive Bayes. Because the probabilities for each class are calculated in a simple manner to make them tractable, it is known as Naive Bayes or idiot Bayes. The probability of each attribute value are presumed to be conditionally independent given the class value rather than trying to compute them. The idea that the qualities do not interact is a very strong assumption that is highly improbable given actual data. However, on data where this assumption is false, the method works surprisingly well.

The Naive Bayes Classifier model was built using the GaussianNB algorithm provided by Scikit-learn. The model was trained on the provided training set and made predictions on the testing set. The performance of the model was then evaluated using the accuracy\_score function from the Scikit-learn library. The model achieved an accuracy score of 0.9557884815255856, indicating that it performed well in predicting the target variable.

Graphical user interface, text, application, chat or text message

Description automatically generated

**Logistic Regression Classifier:**

The Logistic Regression Classifier model was built using the Scikit-learn library. The model was trained on the provided training set using the OvR (One-vs-Rest) approach for multi-class classification. The maximum number of iterations was set to 500. The model was then evaluated on the testing set and achieved an accuracy of 97.78%. This accuracy score indicates that the model performed well in predicting the target variable.

Graphical user interface, text, application

Description automatically generated

**XGBoost Classifier:**

A gradient boosting architecture serves as the foundation for XGBoost. Machine learning techniques like gradient boosting are used to solve classification, regression, and clustering issues. In order to make predictions, the model is optimized. With this method, various models are combined to do a single task. Weak learners refer to the foundation models. They operate under the premise that a weak learner provides subpar predictions when acting alone but superior predictions when acting as a team. Based on weak learners, XGBoost develops strong learners.

The XGBoost Classifier model was built using the XGBoost library in Python. The model was trained on the provided training set with a maximum depth of 2. The model was then evaluated on the testing set, and it achieved an accuracy of 99.74%. This accuracy score indicates that the model performed extremely well in predicting the target variable.

Graphical user interface, text, application, chat or text message

Description automatically generated

**Conclusion:**

It is clear that the XGBoost Classifier outperformed the other models in terms of accuracy with an impressive score of 99.74%. However, it was observed that the XGBoost model overfit the data, as it performed significantly better on the training set than on the testing set. The Logistic Regression Classifier also performed well with an accuracy score of 97.78%. The Random Forest Classifier and Naive Bayes Classifier performed slightly lower, but still achieved respectable accuracy scores of 94.53% and 95.58% respectively.

Overall, the project demonstrates the effectiveness of machine learning algorithms in predicting the target variable of your dataset. The high accuracy scores achieved by the models indicate that they have successfully learned the underlying patterns in the data and can make accurate predictions on new, unseen data.

In conclusion, the above project provides valuable insights into the use of machine learning for classification tasks and highlights the importance of selecting appropriate models and hyperparameters for achieving the best results.