Network Intrusion Detection on Encrypted Traffics using Machine Learning

Md Khairul Islam(mi3se)

May 11, 2022

Abstract

The rapid advance of internet has resulted in huge increase in network size and related data. As a result, new challenges are being posed by novel attacks that threaten the network security. It is important to monitor the traffic to detect any malicious or anomalous behavior. Machine Learning (ML) and Deep Neural Networks (DNNs) have been extensively used in current world to detect network intrusions. However, most of these network intrusion detection systems (NIDS) focus training on old dataset which are obsolete in current time. This limits their capabilities in detecting intrusions in modern day traffic. This work focused on the using Machine Learning techniques for intrusion detection on state-of-the-art HIKARI-2021 (Ferriyan et al., 2021) dataset. The dataset is recently published, represents modern world encrypted networks and overcomes several limitations of the prior works. We show why state-of-the-art model on this dataset does not correct evaluation setup. Then we follow standard machine learning procedures to address imbalance of this data, pre-process raw feature set, find the best classifier, tune model parameters, then finally show the test performance. We also mention importance of different features. Our results show significant improvement compared to the state-of-the-art model and can be used to hugely increase network intrusion detection rate.

1 Introduction

Web applications are getting increasingly popular as internet becomes an integral part of our daily life. As a consequence, network systems are being targeted more by attackers (Ashiq et al., 2019) with malicious intent (Jonas et al., 2019). To detect intruders in a network system, there are generally two approaches: signature-based and anomaly-based detection. Signature-based systems maintain a database of previously known attacks and raise alarms when any match is found with the analyzed data. However, they are vulnerable to zero-day attacks.

An anomaly in a network means a deviation of traffic data from its normal pattern. Thus, anomaly detection techniques have the advantage of detecting zero-day attacks. However, in a complex and large network system, it is not easy to define a set of valid requests or normal behavior of the endpoints. Hence, anomaly detection faces the disadvantage of having a high false-positive error rate.

According to Ahmed et al. (2016), the main attack types are DoS, Probe, User to Root (U2R), and Remote to User (R2U) attacks. Ahmed et al. (2016) mapped the point anomaly with the U2R and the R2U attacks, the DoS attack to the collective anomaly, and the Probe attack to the contextual anomaly.

Our work in this paper is to focus on detecting application layer attacks that employ HTTPS. The HIKARI-2021 Ferriyan et al. (2021) we use has not yet fully explored to create NIDS system using machine learning techniques. Given this dataset is very recent and represent modern day traffic better than the older dataset we decide to use this in our experiments. For the sake of comparison, we also mention the experimentation setup the current state-of-the-art follows and what are its limitations. The limitation presents in the applied techniques can present severely misleading results. Islam et al. (2020) found that using different experimental setups, results can be vastly different for the same dataset and model. They also found in many cases results presented the the prior works do not reflect real-world evaluation. We discuss what is the limitation, then present our modifications. Finally we show our result significantly outperforms state-of-the-art model and overcomes its limitation to be more usable in practice.

The rest of the paper has been organized as follows. Section2 describes the recent works related to this project. Our dataset is described in Section 3. Section 4 lists the evaluation metrics we have used during our experiments. The experimental setup is explained in Section 5. Section 6 presents the results. Section 7 discuss our findings in this work. Possible future works are described in 8. Finally, Section9 has the concluding remarks.

2 Related Works

For network intrusion detection KDDCUP99, NSL-KDD, DARPA, UNSW-NB15 are among the benchmark dataset. There has been numerous works on detecting intrusions based on these datasets.

Moustafa et al. Moustafa and Slay (2017) used central points of attribute values and Association Rule Mining for feature selection on a high level of abstraction from datasets UNSW-NB15 and NSL-KDD. They have partitioned the datasets into train and test sets following an equation. Then evaluated performance using Expectation-Maximisation clustering (EM), Logistic Regression (LR), and Naïve Bayes (NB). Moustafa et al. Moustafa et al. (2018) also proposed a beta mixture model-based anomaly detection system on the UNSW-NB15 dataset. They first selected eight features from the dataset, then randomly selected samples from it.

In another work, Moustafa et al. Moustafa et al. (2019) selected random samples from the UNSW-NB15 dataset and ran ten-fold cross-validation on it. Koroniotis et al.Koroniotis et al. (2017) selected the top ten features of the UNSW-NB15 combined (train+test) dataset using Information Gain Ranking Filter.

Suleiman et al. Suleiman and Issac (2018) explored the performance of machine learning classifiers on benchmark and new datasets (UNSW-NB15, NSL-KDD, and Phishing) using ten-fold cross-validation. They found the RandomForest classifier performs the best. Nawir et al. Nawir et al. (2019) applied ten-fold cross-validation on the binary classification of the combined (train+test) dataset by using the WEKA tool. They also compared centralized and distributed AODE algorithms based on accuracy against the number of nodes.

Meftah et al. Meftah et al. (2019) applied both binary and multiclass classification on the UNSW-NB15 dataset. They found that for binary classification SVM performs the best in ten-fold cross-validation and decision tree (C5.0) for multiclass classification. Hanif et al. Hanif et al. (2019) used ANN(Artificial Neural Network) on the same dataset. They compared their performance with prior works on the NSL-KDD dataset, instead of works on the same dataset.

Vinaykumar et al. Vinayakumar et al. (2019) used classical machine learning classifiers, and deep neural networks on several intrusion detection datasets. The classical models performed much better than the neural network models. Dahiya et al. Dahiya and Srivastava (2018) applied feature reduction techniques on both larger and smaller versions of the UNSW-NB15 dataset.

3 Dataset

The dataset selected for this work in HIKARI-2021 Ferriyan et al. (2021) which is the state-of-the-art network intrusion detection dataset. This dataset is created following several standard rules that the authors found crucial to generate quality intrusion detection data. Previous NIDS datasets only fulfil some of those criteria. Also this dataset is regularly updated and is publicly available here. The authors focused on application layer attacks that employ HTTPS with TLS version 1.2 to deliver the attacks. So working on this dataset helps detect attacks on encrypted networks. The dataset has 86 features, 80 of them are adopted from the CICIDS-2017 (Sharafaldin et al., 2018) dataset.

Figure 1 shows the traffic category count in the whole dataset. Benign and background are considered benign traffic, where the other found category are intrusion attacks. Around 93.21% traffic in the dataset is benign and the rest is intrusion attacks. The dataset is imbalanced and fully labeled. We have used this dataset to perform **supervised binary classification**.

4 Evaluation Metrics

The evaluation metrics used in this work are the following,

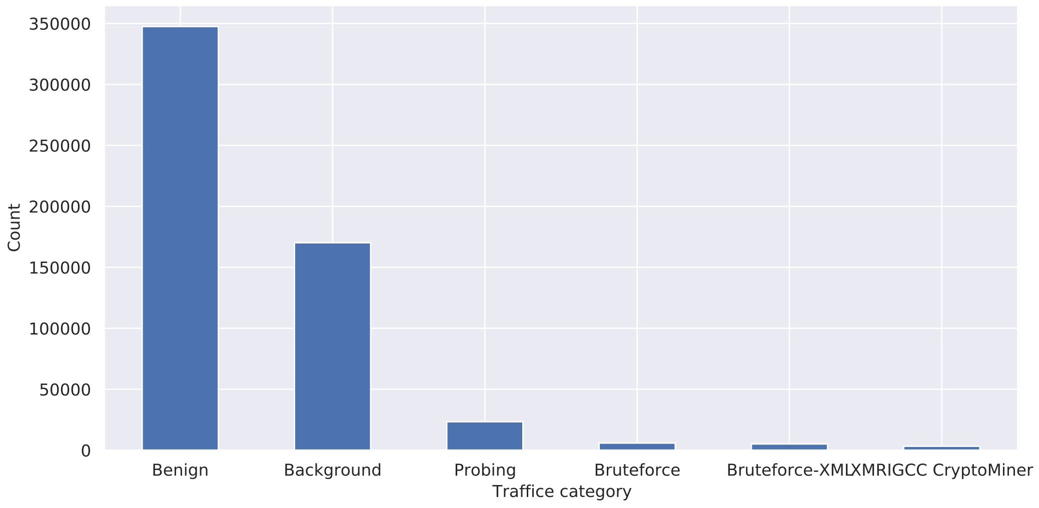

Figure 1: Traffic category count.

- **ROC-AUC**
- **Accuracy**
- **Precision**
- **Recall**
- **F1-score**

ROC-AUC and F1-score are important here because this is an imbalanced dataset. So using only accuracy metric does not reflect the actual performance of the model. We additionally mention precision and recall.

5 Experimental Setup

This section describes the experiments performed in our work. All of these experiments were implemented using python. The default seed was set to 2022 for all random methods to ensure reproducible results.

5.1 Models

For our experiment in this work, we use the following models,

1. LogisticRegression
2. DecisionTreeClassifier
3. RandomForestClassifier
4. GaussianNB
5. XGBClassifier
6. LGBMClassifier

The first four models are imported from scikit-learn python framework. The XGBClassifier and LGBMClassifier are imported from xgboost and lightgbm framework respectively.

5.2 Train-Test Split

First we implement a similar experimental setup as the state-of-the-work on this dataset (Ferriyan et al., 2021). There the authors presented train result as the model performance, which is around 99% is all metrics. But this does not represent the test performance of the model. So to prove why model performance in th the prior work is limited, we randomly split the dataset into train test set using 80:20 ratio. Then each of the models in Section 5.1 is trained of the train data, then tested on the separate 20% test data. The results are presented in Table 1.

5.3 Handling Dataset Imbalance

Following our observations from the Table 1 we use the RandomUnderSampler class from Imblearn library to randomly under-sample our training dataset. Then used the under-sampled dataset to train the same models on the same test dataset. We don't balance our test dataset to evaluate real-world performance of the model. As in real world, network intrusions ratio is much less than benign traffics. The results are mentions in Table 2.

Following the performance improvement after under-sampling the training data, **for the rest of the experiments we always under-sample our train dataset, but not the test or validation data**.

5.4 Selecting Best Classifier

In order to select the best classifier for our work, we use **ten-fold cross validation** on the 80% train dataset. At iteration of the cross-validation the train data is under-sampled, then evaluated on the unchanged validation dataset. We use the Imblearn pipeline to automate this process. Then mention the mean validation performance across all folds in the Table 3.

Then best classifier is then hyper-tuned varying maximum depth, number of estimators and learning rate. The best performing parameters are used to evaluated on the test dataset.

5.5 Evaluation on Test Data

The test dataset is kept separated and unseen during the whole experimentation to ensure getting real world alike performance. We use models trained in each fold of the cross-validation to predict on the test data. Then mention the average results across all of those models. This ensures more robust test results. We also saw, the results are quite similar for those models. This validate that the models trained in those folds are quite robust.

Environment: Model training and testing were done in Google Colab notebooks with CPU. The processor was dual core Intel(R) Xeon(R) CPU @ 2.20GHz with 12.68GB RAM memory.

6 Results

In this section we present the results of the experiments described in the previous section.

6.1 Train-test split

Table 1 shows the model performance when trained on 80% data, and tested on the rest. The results show a significant performance drop. Which indicates good training results not necessarily guarantee good test results. In their work Ferriyan et al. (2021) found RandomForestClassifier with default parameters to achieve near perfect scores. Here we can see that on a separate test set there is significant performance drop.

We can see the accuracy is still very high for some models. This is because there are a lot benign traffic in the dataset. Even if the model always predicts the traffic to be benign it will still get a very high accuracy, since 93%+ traffics in this dataset is benign.

Model	**auc**	**f1**	**precision**	**recall**	**accuracy**
LogisticRegression	0.336	0.006	0.215	0.003	0.931
DecisionTreeClassifier	0.536	0.137	0.121	0.159	0.863
RandomForestClassifier	0.89	0.106	0.112	0.1	0.884
GaussianNB	0.628	0.161	0.088	0.977	0.302
XGBClassifier	0.946	0.181	0.542	0.108	0.933
LGBMClassifier	0.947	0.324	0.481	0.244	0.93

Table 1: Test Performance for 80:20 train-test split .

6.2 Under-sampling Dataset

Observing the significant performance drop in the previous section we decide to address the imbalanceness of the dataset first. We address this by using RandomUnderSampler from Imblearn. This methods randomly samples the majority classes to be of same number as the minority classes. Table 2 shows that we can already see significant performance improvement compared to Table 1. This proves under-sampling the dataset is beneficial for this work. So throughout the rest of the experiments we have always under-sampled the training dataset.

Model	**auc**	**f1**	**precision**	**recall**	**accuracy**
LogisticRegression	0.888	0.362	0.222	0.977	0.765
DecisionTreeClassifier	0.865	0.472	0.325	0.862	0.868
RandomForestClassifier	0.929	0.484	0.325	0.945	0.862
GaussianNB	0.633	0.17	0.093	0.978	0.345
XGBClassifier	0.948	0.498	0.332	1.0	0.862
LGBMClassifier	0.946	0.5	0.334	0.999	0.863

Table 2: Test Performance for 80:20 train-test split after under-sampling train data.

We have also tried using StandardScaler and MinMaxScaler to see if the provide any performance improvement. However, scaling didn't have any positive impact of the models for this dataset. So finally we didn't use it.

6.3 Selecting the Best Classifier

Table 3 shows the ten-fold cross-validation score for each of the models on the 80% train data. Each model was trained with its default parameters. **We find LGBMClassifier to perform the best across those models in terms of F1-score and close to best in AUC score**. It also performs greatly in other metrics. So we chose this as our best classifier.

Model	**auc**	**f1**	**precision**	**recall**	**accuracy**
LogisticRegression	0.879	0.363	0.223	0.968	0.77
DecisionTreeClassifier	0.863	0.456	0.316	0.817	**0.868**
RandomForestClassifier	0.929	0.479	0.321	0.946	0.861
GaussianNB	0.609	0.166	0.091	0.979	0.335
XGBClassifier	**0.949**	0.492	0.326	0.999	0.86
LGBMClassifier	0.947	**0.496**	**0.33**	**0.999**	0.862

Table 3: Cross validation score on under-sampled 80% train data.

6.4 Hyper-parameter Tuning

We tune the max_depth, n_estimators and learning_rate parameters of the LGBMClassifier. Table 4 shows the tuning performance. We see the results are almost identical in most cases. **However considering both AUC and F1-score we chose max_depth=None, n_estimators=100 and learning_rate=0.05 as the best combination**. And use this to evaluate the 20% test data.

learning_rate	**n_estimator**	**depth**	**accuracy**	**auc**	**f1**	**precision**	**recall**
0.1	100	None	0.862	0.947	0.496	0.33	0.999
0.1	100	3	0.86	0.949	0.492	0.326	0.999
0.1	100	5	0.862	0.949	0.494	0.329	0.999
0.1	500	None	0.862	0.94	0.493	0.328	0.986
0.1	500	3	0.863	0.948	0.496	0.33	0.999
0.1	500	5	**0.863**	0.944	0.496	0.33	0.995
0.05	**100**	**None**	0.862	**0.949**	**0.496**	**0.33**	**0.999**
0.05	100	3	0.852	0.946	0.477	0.314	0.998
0.05	100	5	0.859	0.949	0.49	0.325	0.999
0.05	500	None	0.863	0.944	0.496	0.33	0.996
0.05	500	3	0.862	0.949	0.495	0.329	0.999
0.05	500	5	0.863	0.947	0.496	0.33	0.998

Table 4: Hyper-parameter tuning of LGBMClassifier using ten-fold cross validation score on under-sampled 80% train data.

6.5 Test results

Table 5 shows the test performance of our best model. This result is significantly better than what we saw in our first experiment in Table 1. This shows hugely improved AUC and F1-score. Though we can also see the precision score is still pretty low.

Prediction Threshold	**auc**	**f1**	**precision**	**recall**	**accuracy**
0.50	0.948	0.502	0.335	0.999	0.864
0.82	0.948	0.523	0.410	0.722	0.910

Table 5: Test result on 20% test data using models trained in the ten-fold cross validation on 80% train data

To investigate this we try varying our prediction threshold from 0.1 to 0.99 and plot the corresponding results in Figure 2. This curve shows that increasing threshold barely has any impact until 0.78. At threshold 0.82 (line drawn) we see a relatively good balance between precision and recall, also a good F1-score. Test performance at this point is also mentioned in Table 5. So users can choose best threshold point based on their preference.

Figure 3 shows the ROC-AUC curve of our predictions. We see our model covers a good amount of area in the curve.

Feature Importance

The feature importance are calculated using the feature_importances_ attribute of the LGBMClassifier classifier. For each model during each fold of the ten-fold cross-validation, the importance is extracted and finally the mean importance is converted into percentage and sorted. We only mention the top ten features here.

7 Discussion

In this section we discuss the overall observations from our experiments.

Our first observation shows that it is important to follow standard machine learning practices when applying ML models. Otherwise the results can be very misleading (as discussed in Section 6.1). We see real world network intrusion attack datasets generally imbalanced. Hence requires careful handling to address that imbalanceness when training the model.

Overall we found it difficult to get a good F1-score on this dataset. This shows the complexity of distinguishing intrusion attacks in the wild, specially for novel attacks this should be more difficult. But from our model we also see it has a near perfect recall but low precision. So the model predicts a lot of false positives. Which in the problem with these anomaly based systems, as they can often

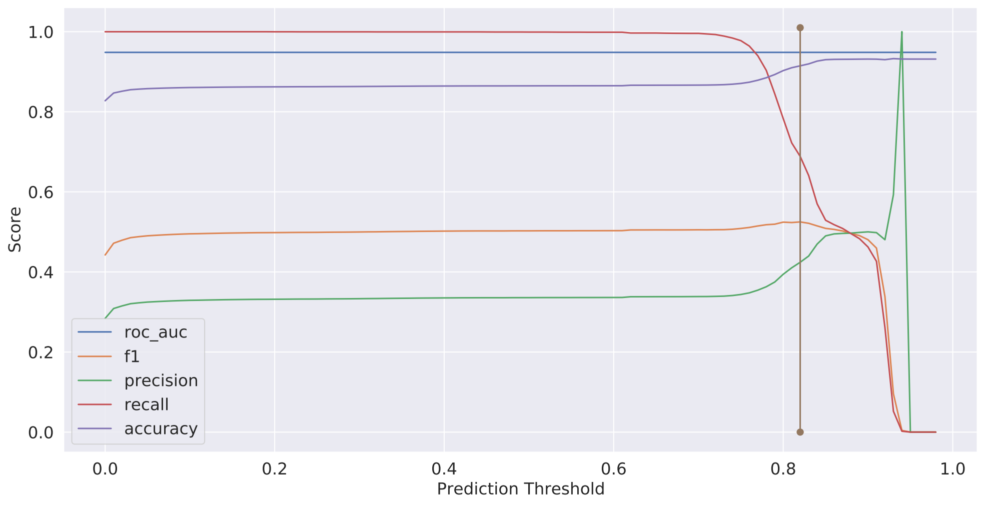

Figure 2: Effect of changing prediction threshold.

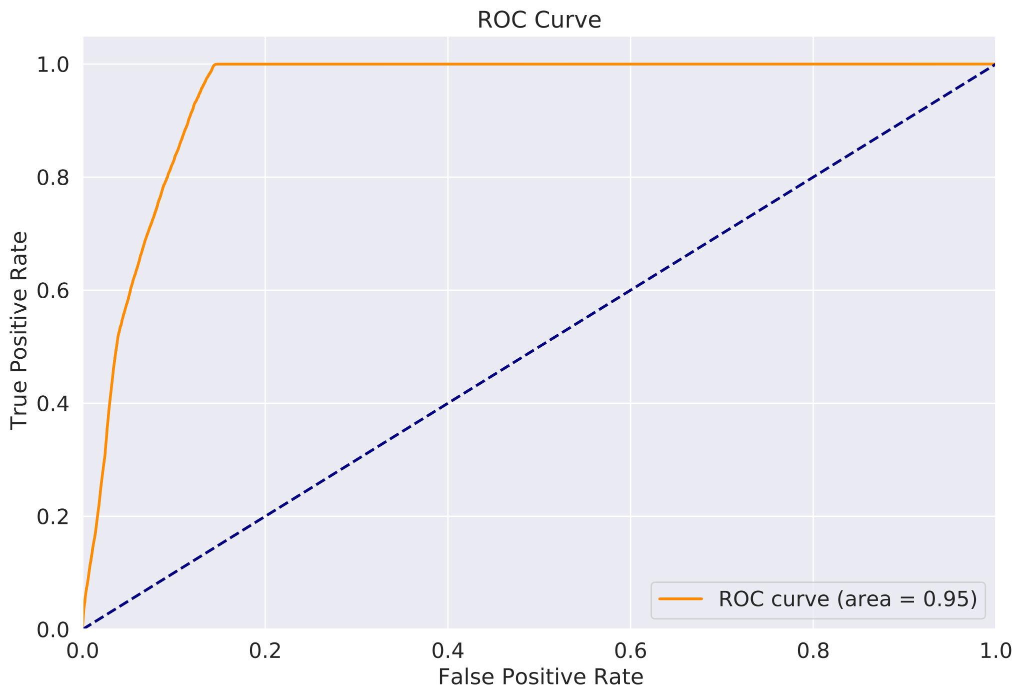

Figure 3: ROC-AUC curve

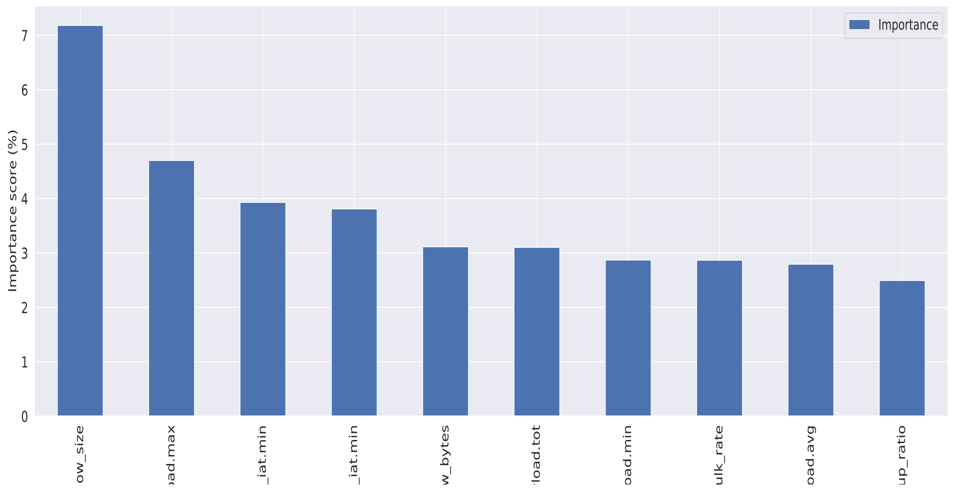

Figure 4: Top ten feature importance.

trigger alarms even for benign traffics. However, we show that by adjusting prediction threshold is it possible to get reasonable balance between precision and recall while still maintaining good score in other metrics (Table 5).

8 Future Work

The future work possible on this dataset is to perform multi class classification. Also try using deep learning models like Variational AutoEncoders (VAE) to see how they perform for this dataset. We have tried CNN and DNN on this data, but didn't see better performance compared to best performing models here. So VAE can be a good option to try improving the results.

9 Conclusion

In conclusion, in this work we perform network intrusion detection on the HIKARI-2021 Ferriyan et al. (2021) dataset. Our results show the limitation in the prior work on the same dataset and how to overcome that. We perform extensive analysis of the dataset and mention five related metrics to evaluate our model. Our result is validated on a separate test set and hence better represents the model's behavior to unseen dataset. Our work would be hugely beneficial to the network security community in evaluating NIDS on this dataset.

The replication package are uploaded to our Github repository IDS-on-HIKARI-2021.

References

Ahmed, M., Mahmood, A. N., and Hu, J. (2016). A survey of network anomaly detection techniques. *Journal of Network and Computer Applications*, 60:19–31.

Ashiq, M. I., Bhowmick, P., Hossain, M. S., and Narman, H. S. (2019). Domain flux based dga botnet detection using feedforward neural network. In *IEEE Military Communications (MILCOM)*, Norfolk, VA, USA. IEEE.

Dahiya, P. and Srivastava, D. K. (2018). Network intrusion detection in big dataset using spark. *Procedia computer science*, 132:253–262.

Ferriyan, A., Thamrin, A. H., Takeda, K., and Murai, J. (2021). Generating network intrusion detection dataset based on real and encrypted synthetic attack traffic. *Applied Sciences*, 11(17):7868.

Hanif, S., Ilyas, T., and Zeeshan, M. (2019). Intrusion detection in iot using artificial neural networks on unsw-15 dataset. In *16th International Conference on Smart Cities: Improving Quality of Life Using ICT & IoT and AI)*, pages 152–156. IEEE.

Islam, M. K., Hridi, P., Hossain, M. S., and Narman, H. S. (2020). Network anomaly detection using lightgbm: A gradient boosting classifier. In *2020 30th International Telecommunication Networks and Applications Conference (ITNAC)*, pages 1–7. IEEE.

Jonas, M. A., Islam, R., Hossain, M. S., Narman, H. S., and Atiquzzaman, M. (2019). An intelligent system for preventing ssl stripping-based session hijacking attacks. In *IEEE Military Communications (MILCOM)*, Norfolk, VA, USA. IEEE.

Koroniotis, N., Moustafa, N., Sitnikova, E., and Slay, J. (2017). Towards developing network forensic mechanism for botnet activities in the iot based on machine learning techniques. In *International Conference on Mobile Networks and Management*, pages 30–44. Springer.

Meftah, S., Rachidi, T., and Assem, N. (2019). Network based intrusion detection using the unsw-nb15 dataset. *International Journal of Computing and Digital Systems*, 8(5):478–487.

Moustafa, N., Creech, G., and Slay, J. (2018). Anomaly detection system using beta mixture models and outlier detection. In *Progress in Computing, Analytics and Networking*, pages 125–135. Springer.

Moustafa, N., Hu, J., and Slay, J. (2019). A holistic review of network anomaly detection systems: A comprehensive survey. *Journal of Network and Computer Applications*, 128:33–55.

Moustafa, N. and Slay, J. (2017). A hybrid feature selection for network intrusion detection systems: Central points. *arXiv preprint arXiv:1707.05505*.

Nawir, M., Amir, A., Yaakob, N., and Lynn, O. B. (2019). Effective and efficient network anomaly detection system using machine learning algorithm. *Bulletin of Electrical Engineering and Informatics*, 8(1):46–51.

Sharafaldin, I., Lashkari, A. H., and Ghorbani, A. A. (2018). Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp*, 1:108–116.

Suleiman, M. and Issac, B. (2018). Performance comparison of intrusion detection machine learning classifiers on benchmark and new datasets. In *28th International Conference on Computer Theory and Application*, pages 447–489.

Vinayakumar, R., Alazab, M., Soman, K., Poornachandran, P., Al-Nemrat, A., and Venkatraman, S. (2019). Deep learning approach for intelligent intrusion detection system. *IEEE Access*, 7.