NetGuard

CIS4914 Senior Project

Advisor: Dr. Wilson - jnw@cise.ufl.edu
Project Manager - Austin Rhee
Scrum Master - Brian McDowell
Back End Developer - Dillon Kilgallon
Back End Developer - Michael Tumminia
Back End Developer - Safeah Hammosh

Final Presentation: https://youtu.be/G7HpmwgW2UE

Keywords

Intrusion Detection System, Machine Learning, Artificial Intelligence, Threat Detection, Anomaly Detection, Network Security, Cyber Security, CIC-IDS2017.

Project Overview

Project Name and Description

NetGuard is an Intrusion Detection System (IDS) designed to identify malicious network activity using machine learning techniques. The system analyzes network traffic, including stateful and statistical TCP features (Figure 2A), to improve detection accuracy across various attack categories. NetGuard emphasizes efficient model training, thorough feature evaluation, and reliable performance measurement using tools such as confusion matrices, ROC curves, and SHAP-based analysis.

The purpose of this project is to build an adaptive, efficient, and dependable IDS that could be applicable in real-world network environments. The system aims to address challenges in modern cybersecurity, including high data volume, class imbalance, and the difficulty of accurately detecting minority attack types.

Project Goals and Objectives

- Data Preprocessing and Structuring
- Model Training and Development
- Rigorous Evaluation and Validation

Project Scope

The scope of the NetGuard project focuses on developing a complete machine learning pipeline for intrusion detection and evaluating its performance on large-scale network traffic.

Team members and their roles and responsibilities

Austin Rhee (Project Manager): PCAP Data Inspection, KNN model

Brian McDowell (Scrum Master): Managing JIRA pipeline, Random Forest model, RNN/LSTM model.

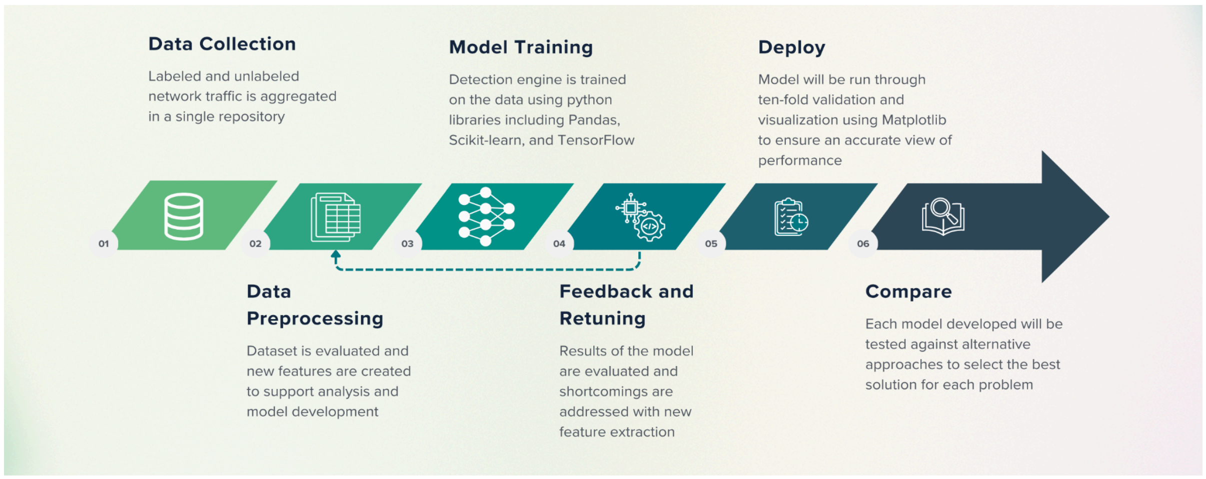
Dillon Kilgallon (Back End Developer): XGBClassifier, LGBMClassifier, GradientBoostingClassifier, SHAP, and SMOTE.

Michael Tumminia (Back End Developer): SVM model, cross-fold validation, PCAP inspection, TCP stream analysis.

Safeah Hammosh (Back End Developer): Naive Bayes Model, Model Inspection, Dataset Preprocessing, and Performance Validation.

Deliverables

This project work was organized through a Jira Kanban format. We created work tasks for all major milestones and moved them through a traditional TO DO -> IN PROGRESS -> DONE workflow. Tasks were developed to support our planned architecture that would progress sequentially from data collection through final comparison.


Data Collection
Labeled and unlabeled network traffic is aggregated in a single repository
Model Training
Detection engine is trained on the data using python libraries including Pandas, Scikit-learn, and TensorFlow
Deploy
Model will be run through ten-fold validation and visualization using Matplotlib to ensure an accurate view of performance
01
02
03
04
05
06
Data Preprocessing
Dataset is evaluated and new features are created to support analysis and model development
Feedback and Retuning
Results of the model are evaluated and shortcomings are addressed with new feature extraction
Compare
Each model developed will be tested against alternative approaches to select the best solution for each problem

With this foundation, our Kanban board shows 24 completed tasks including:

- Data collection tasks including literature review, data sourcing, data inspection, data import scripting, and cloud hosting for high-speed repeated use
- Preprocessing tasks including feature selection, feature extraction, data organization, data visualization, and data segregation into discreet folds
- Model training of conventional ml models including Support Vector Machine, K-Nearest Neighbors, Naive Bayes, Random Forest, and several variants of Gradient Boosting including XGB and LGBM, followed by analysis using classification reports, Confusion Matrix, and Receiver Operator Characteristic graphs
- Feedback and retuning tasks including Receiver Operating Characteristics, additional feature extraction from PCAP data, SHapley Additive exPlanations (SHAP) and Synthetic Minority Oversampling Technique (SMOTE)

- Model training of recurrent neural networks focused on Long Short-Term Memory (LSTM) followed by analysis using classification reports and Confusion Matrix graphs

Tasks which were left incomplete or unstarted consisted of stretch goals such as automating the model comparisons and more advanced models like Convolutional Neural Networks and Feed Forward Neural Networks.

The next steps of this project will return to hyperparameter tuning and model specialization in order to improve the detection engine (DE) accuracy for existing models prior to moving to more advanced models such as CNN and FFNN. It is acknowledged that the high levels of false negative prediction errors found in our current models is the greatest risk to the deployment of any production DE tool, and as such this must be addressed.

Technical Details

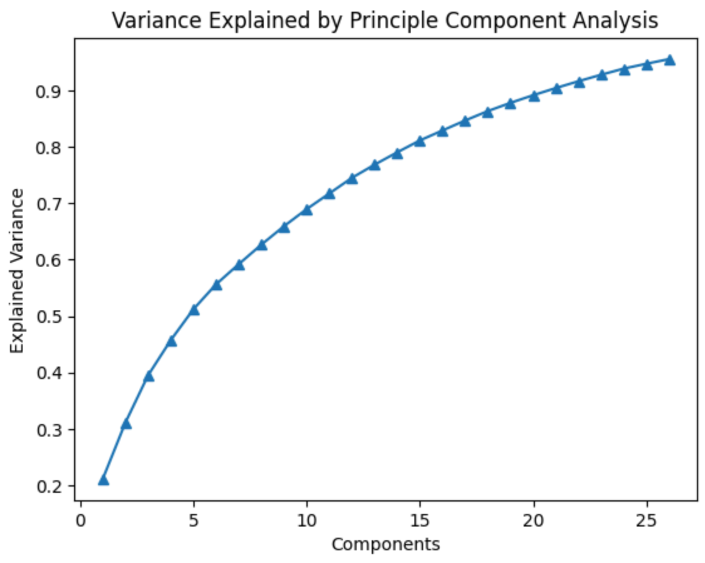
This project was primarily conducted in Google Colab using default High-RAM instances. It was developed in Python and relies on standard machine-learning libraries such as NumPy, Pandas, Matplotlib, Scikit-learn, and SHAP, with results and code managed through GitHub.

Due to the research-oriented nature of our project, the system architecture was intentionally simple: a collection of scripts for data preprocessing, model training, evaluation, and explainability. No database was required, as the CIC dataset was read from CSV files, processed in memory, and output back to CSV. Overall, the project focused on exploring models, data processing methods, evaluation metrics, and explainability techniques. Key models, methods, and findings are summarized below.

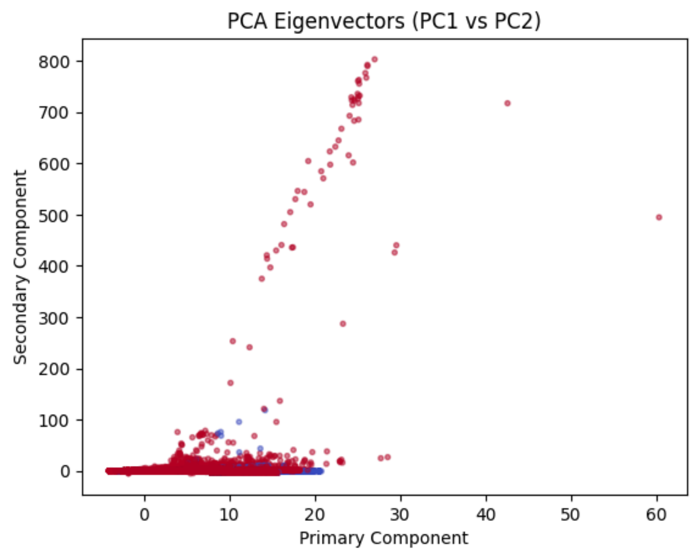
Preprocessing Techniques

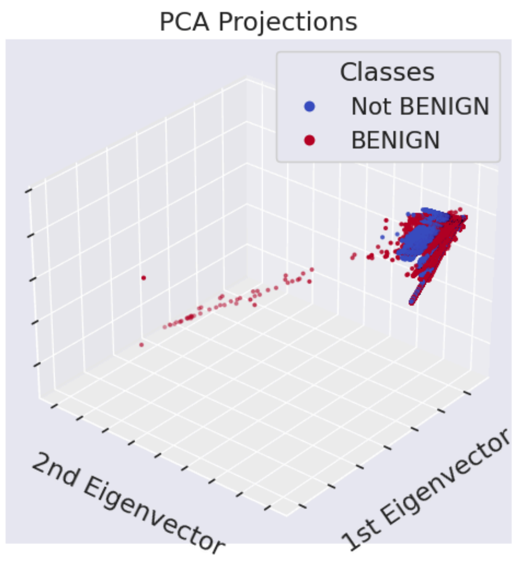
Principal Component Analysis

The technique of PCA was employed to determine the relative weights of various features and to look for clear separation between classes. As shown in the graphs below, much of the discrimination between classes was determined by just a few dimensions, with later components adding relatively little to the classifier decision.



Plotting the principal components shows that, while there are eigenvectors that can be identified, there is significant overlap between the components that suggests that a classifier will struggle to clearly distinguish between labels.





SMOTE

Synthetic Minority Oversampling Technique is a sampling method used to address imbalanced datasets, such as ours. The technique creates “synthetic” examples of minority classes by creating a new sample whose features are interpolated between a minority-class sample and its nearest minority-class neighbors. This helps the model learn the decision boundaries of minority classes more effectively and typically improves their metrics. The trade off is the potential to affect the performance of nearby majority-class samples, although in our case we found these results appeared acceptable.

PCAP Data Ingest

PCAP Data (packet capture data) was inspected for possible integration during this project. This data provided unique insights into the raw data, allowing us to inspect individual packets (Figure 3A). We also determined that the data contained certain proprietary packets used for routing, which were removed before model training.

Despite the usefulness in manual packet inspection, the sheer volume of PCAP data was determined to be untenable for our compute resources. Totaling over 30GB+, model training would simultaneously require large amounts of RAM and long compute times.

Models

XGBClassifier

The XGBClassifier model was selected for its strong performance on classification tasks and its ability to scale to very large datasets. Its gradient-boosting framework provides high accuracy, particularly with structured features like those in our dataset. Potential drawbacks include sensitivity to overfitting if not properly regularized and the need for careful hyperparameter tuning, which we were limited in doing for this project. Although XGBoost can struggle with high-dimensional sparse features, our SHAP-based feature reduction helps mitigate this risk. Training time can be substantial on large datasets, but this was manageable using default configurations and subset sampling for SHAP computations.

LGBMClassifier

This model was considered for its fast training speed and memory efficiency, which is advantageous when working with large datasets such as CIC-IDS. It handles multiclass classification well with structured features and generally performs effectively without extensive hyperparameter tuning. Potential drawbacks include some sensitivity to overfitting, although our large dataset mitigates this, and decreased performance when many irrelevant features are present. We addressed this by reducing the feature space using SHAP analysis.

GradientBoostingClassifier

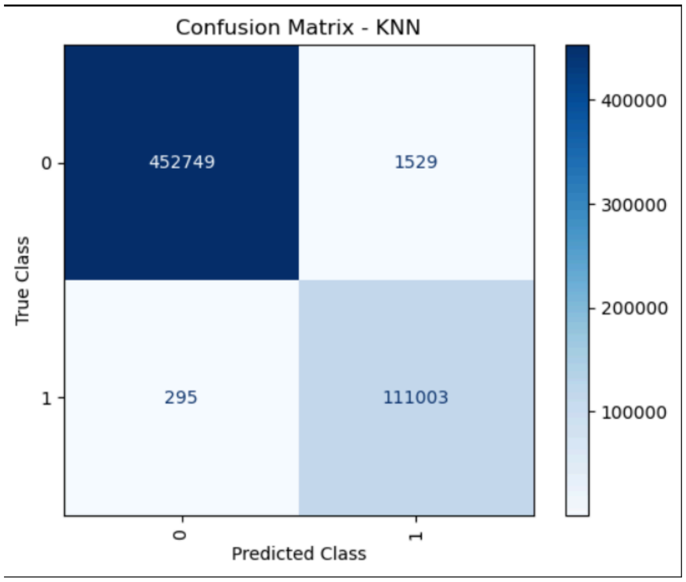
A gradient boosting approach is well-suited for tabular classification and provides good accuracy even with default parameters. It is especially appealing when hyperparameter tuning is limited, as in our project. Drawbacks include longer training times on large datasets and susceptibility to overfitting if parameters are not carefully chosen. While the model benefits from simplicity, large datasets can make training computationally expensive compared with XGBoost or LightGBM.

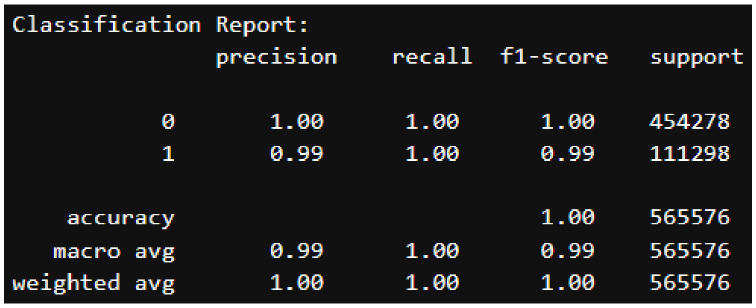
K-Nearest Neighbors

K-Nearest Neighbors was initially chosen due to its relative simplicity in implementation. This model was initially trained before in-depth preprocessing in an attempt to gauge both speed of training and dimensional complexity.

Despite initially quick model training times, we quickly found that KNN suffered significantly against under-represented classes (Figure 1A). We also noted an immediate issue with this type of classifier, namely abnormally high success rates. This however proved to be a twofold issue: chiefly, the model attempted to classify nearly all traffic as benign (due to noted imbalances) and secondarily it utilized features which were determined to cause data leakage (Figure 2A). This data leakage was further investigated during our PCA and SMOTE analyses.

After completing our data analyses, this model was then re-evaluated to determine its general ability to classify all types of attack classes vs benign traffic. This yielded far better results, with a high level of benign traffic being accurately classified. Issues did however arise with false negatives, and this generalist nature prevented closer analysis of accuracy on a per-attack basis.



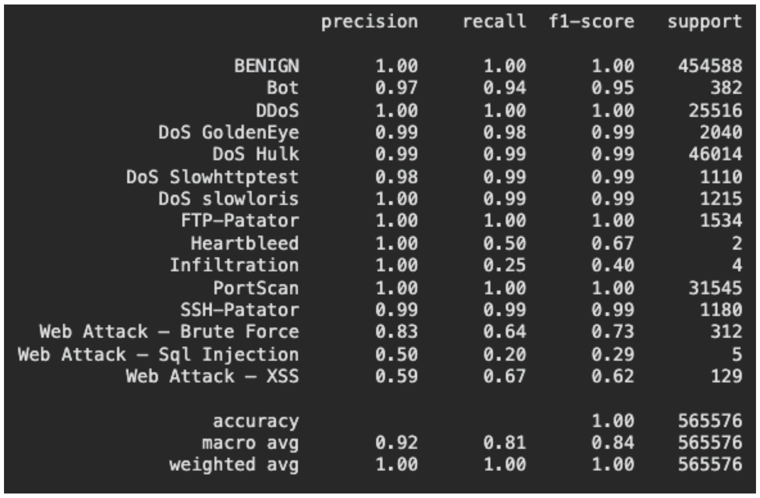


```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    454278
           1       0.99      1.00      0.99    111298

    accuracy                           1.00    565576
   macro avg       0.99      1.00      0.99    565576
weighted avg       1.00      1.00      1.00    565576
```

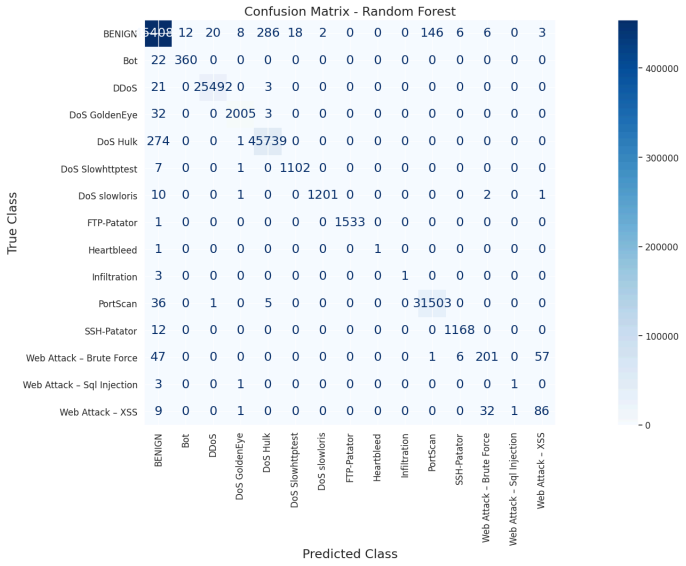
Random Forest

The Random Forest model was selected due to its expectation of high accuracy with high-dimensional data. After preprocessing we were left with 26 dimensions and from that we were attempting to accurately choose between 15 categories - this is an excellent target for the multiple decision tree approach of Random Forest models.

In practice we found that it was computationally expensive to build the model and that performance suffered on under-represented classes (Figure 1A). This was especially noticeable in the various web attacks that delivered coinflip or worse performance within this model. Where substantial training and test data was available, the model performed reasonably well and delivered precision and recall scores of .98 or better for all categories where 1000 or more test cases exist.



```
                        precision    recall  f1-score   support

              BENIGN       1.00      1.00      1.00    454588
                 Bot       0.97      0.94      0.95       382
                DDoS       1.00      1.00      1.00     25516
       DoS GoldenEye       0.99      0.98      0.99      2040
            DoS Hulk       0.99      0.99      0.99     46014
    DoS Slowhttptest       0.98      0.99      0.99      1110
       DoS slowloris       1.00      0.99      0.99      1215
         FTP-Patator       1.00      1.00      1.00      1534
          Heartbleed       1.00      0.50      0.67         2
        Infiltration       1.00      0.25      0.40         4
            PortScan       1.00      1.00      1.00     31545
         SSH-Patator       0.99      0.99      0.99      1180
  Web Attack – Brute Force       0.83      0.64      0.73       312
Web Attack – Sql Injection       0.50      0.20      0.29         5
    Web Attack – XSS       0.59      0.67      0.62       129

            accuracy                           1.00    565576
           macro avg       0.92      0.81      0.84    565576
        weighted avg       1.00      1.00      1.00    565576
```

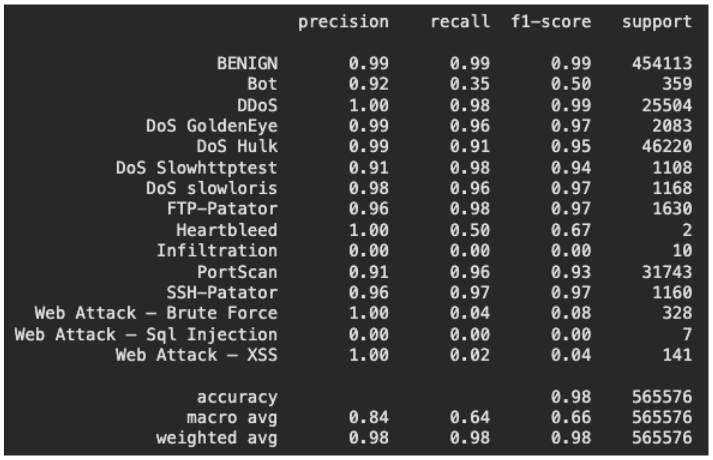


	BENIGN	Bot	DDoS	DoS GoldenEye	DoS Hulk	DoS Slowhttptest	DoS slowloris	FTP-Patator	Heartbleed	Infiltration	PortScan	SSH-Patator	Web Attack – Brute Force	Web Attack – Sql Injection	Web Attack – XSS
BENIGN	5408	12	20	8	286	18	2	0	0	0	146	6	6	0	3
Bot	22	360	0	0	0	0	0	0	0	0	0	0	0	0	0
DDoS	21	0	25492	0	3	0	0	0	0	0	0	0	0	0	0
DoS GoldenEye	32	0	0	2005	3	0	0	0	0	0	0	0	0	0	0
DoS Hulk	274	0	0	1	45739	0	0	0	0	0	0	0	0	0	0
DoS Slowhttptest	7	0	0	1	0	1102	0	0	0	0	0	0	0	0	0
DoS slowloris	10	0	0	1	0	0	1201	0	0	0	0	0	2	0	1
FTP-Patator	1	0	0	0	0	0	0	1533	0	0	0	0	0	0	0
Heartbleed	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0
Infiltration	3	0	0	0	0	0	0	0	0	1	0	0	0	0	0
PortScan	36	0	1	0	5	0	0	0	0	0	31503	0	0	0	0
SSH-Patator	12	0	0	0	0	0	0	0	0	0	0	1168	0	0	0
Web Attack – Brute Force	47	0	0	0	0	0	0	0	0	0	1	6	201	0	57
Web Attack – Sql Injection	3	0	0	1	0	0	0	0	0	0	0	0	0	1	0
Web Attack – XSS	9	0	0	1	0	0	0	0	0	0	0	0	32	1	86

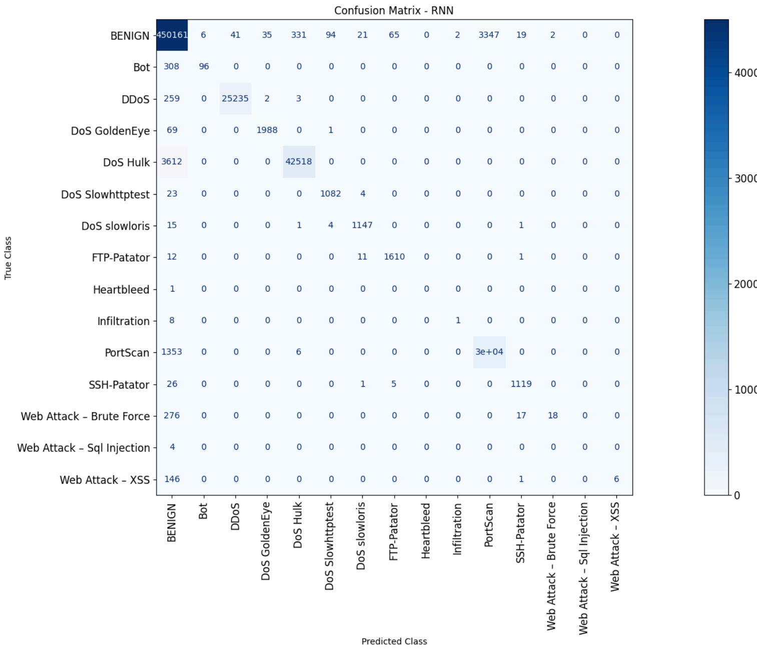
RNN/LSTM

Neural networks have gained popularity as classifiers due to their ability to identify connections between features that may otherwise be overlooked. In looking at options for this project, our research documentation suggested that a Long Short-Term Memory (LSTM) model would likely outperform traditional recurrent neural networks as they are able to retain information over a period of time which resolves the vanishing gradient problem that may otherwise occur. The major drawbacks to this model are the incredibly high variability of weights and biases in conjunction with extended training time. Even after activating a A100 GPU, training time for this model regularly ran over an hour which limited the amount of hyperparameter tuning that could be performed.

In practice our LSTM model delivered only 84% precision (true positives/ all predicted positives) and 64% recall (true positives/ all real positives). LSTM models are typically known to be resilient against underrepresented categories, however our model performed incredibly poorly against these cases. Most inaccurate predictions were false negatives, where attacks were predicted to be benign, but there were also an inexcusably high number of false positives where benign flows were incorrectly predicted to be various attack types.



```
                          precision    recall  f1-score   support

                  BENIGN       0.99      0.99      0.99    454113
                     Bot       0.92      0.35      0.50       359
                    DDoS       1.00      0.98      0.99     25504
           DoS GoldenEye       0.99      0.96      0.97      2083
                DoS Hulk       0.99      0.91      0.95     46220
        DoS Slowhttptest       0.91      0.98      0.94      1108
           DoS slowloris       0.98      0.96      0.97      1168
             FTP-Patator       0.96      0.98      0.97      1630
              Heartbleed       1.00      0.50      0.67         2
            Infiltration       0.00      0.00      0.00        10
                PortScan       0.91      0.96      0.93     31743
             SSH-Patator       0.96      0.97      0.97      1160
  Web Attack – Brute Force       1.00      0.04      0.08       328
Web Attack – Sql Injection       0.00      0.00      0.00         7
        Web Attack – XSS       1.00      0.02      0.04       141

                accuracy                           0.98    565576
               macro avg       0.84      0.64      0.66    565576
            weighted avg       0.98      0.98      0.98    565576
```



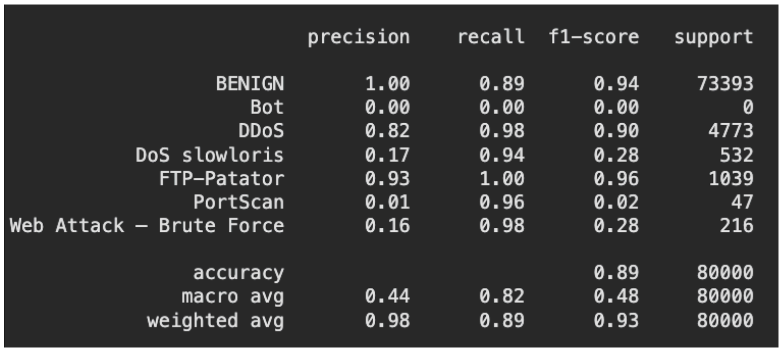
	BENIGN	Bot	DDoS	DoS GoldenEye	DoS Hulk	DoS Slowhttptest	DoS slowloris	FTP-Patator	Heartbleed	Infiltration	PortScan	SSH-Patator	Web Attack – Brute Force	Web Attack – Sql Injection	Web Attack – XSS
BENIGN	450161	6	41	35	331	94	21	65	0	2	3347	19	2	0	0
Bot	308	96	0	0	0	0	0	0	0	0	0	0	0	0	0
DDoS	259	0	25235	2	3	0	0	0	0	0	0	0	0	0	0
DoS GoldenEye	69	0	0	1988	0	1	0	0	0	0	0	0	0	0	0
DoS Hulk	3612	0	0	0	42518	0	0	0	0	0	0	0	0	0	0
DoS Slowhttptest	23	0	0	0	0	1082	4	0	0	0	0	0	0	0	0
DoS slowloris	15	0	0	0	1	4	1147	0	0	0	0	1	0	0	0
FTP-Patator	12	0	0	0	0	0	11	1610	0	0	0	1	0	0	0
Heartbleed	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Infiltration	8	0	0	0	0	0	0	0	0	1	0	0	0	0	0
PortScan	1353	0	0	0	6	0	0	0	0	0	3e+04	0	0	0	0
SSH-Patator	26	0	0	0	0	0	1	5	0	0	0	1119	0	0	0
Web Attack – Brute Force	276	0	0	0	0	0	0	0	0	0	0	17	18	0	0
Web Attack – Sql Injection	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Web Attack – XSS	146	0	0	0	0	0	0	0	0	0	0	1	0	0	6

Naive Bayes Model

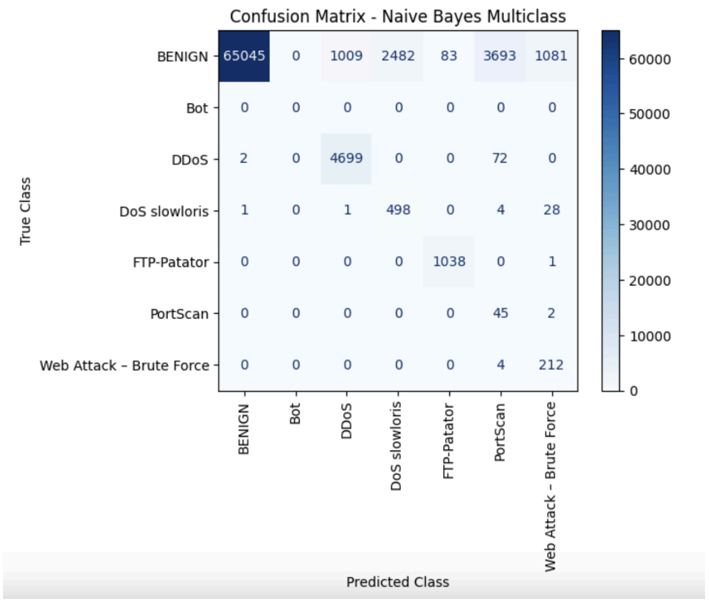
The Naive Bayes model was chosen as our baseline classifier because it is fast, simple to use, and generally performs well on high-dimensional datasets. After preprocessing and feature scaling, we trained the model using the GaussianNB estimator and applied RandomOverSampler (ROS) to help address the strong class imbalance in the training split. We also ran hyperparameter tuning with GridSearchCV, which selected 1e−09 as the best var_smoothing value. The final model was trained to separate the malicious categories in our CIC-IDS-2017 subset and the benign traffic.

In practice, Naive Bayes worked well on the classes that had plenty of training examples. It reached an overall accuracy of 0.89 and showed high recall for attacks like DDoS (0.98) and FTP-Patator (1.00). However, the model struggled with under-represented and sequential attacks. DoS slowloris had a precision of only 0.17, and Web Attack – Brute Force had 0.16, even though both had high recall. This happened because the model frequently predicted these attacks on benign flows.

After inspecting the results more closely, we found the main reason for this issue: Naive Bayes is stateless and relies completely on simple probability distributions. When benign and malicious flows look statistically similar, the model cannot tell them apart. This became obvious when we compared the 2,482 false positives for DoS slowloris to the 498 true positives; both groups had almost identical packet statistics, including the same median of 3.0 forward packets. These results show the limitations of using Naive Bayes for subtle or low-frequency attacks and motivated our move toward more advanced, context-aware approaches, including stateful feature extraction and custom stream-based analysis.



```
                         precision    recall  f1-score   support

                BENIGN       1.00      0.89      0.94     73393
                   Bot       0.00      0.00      0.00         0
                  DDoS       0.82      0.98      0.90      4773
         DoS slowloris       0.17      0.94      0.28       532
           FTP-Patator       0.93      1.00      0.96      1039
              PortScan       0.01      0.96      0.02        47
Web Attack – Brute Force       0.16      0.98      0.28       216

              accuracy                           0.89     80000
             macro avg       0.44      0.82      0.48     80000
          weighted avg       0.98      0.89      0.93     80000
```



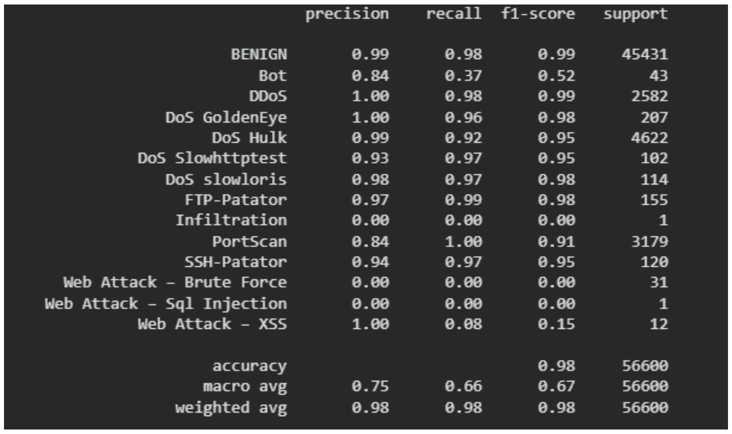
	BENIGN	Bot	DDoS	DoS slowloris	FTP-Patator	PortScan	Web Attack – Brute Force
BENIGN	65045	0	1009	2482	83	3693	1081
Bot	0	0	0	0	0	0	0
DDoS	2	0	4699	0	0	72	0
DoS slowloris	1	0	1	498	0	4	28
FTP-Patator	0	0	0	0	1038	0	1
PortScan	0	0	0	0	0	45	2
Web Attack – Brute Force	0	0	0	0	0	4	212

Support Vector Machine

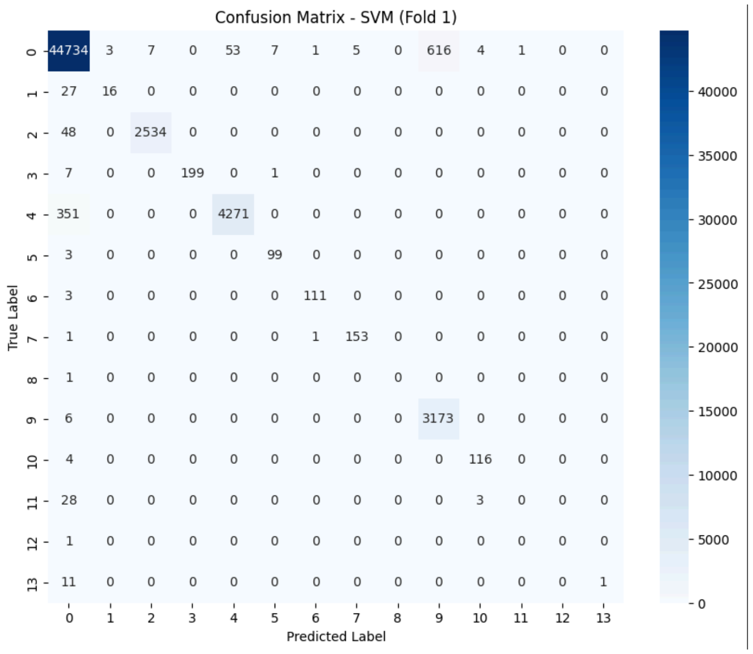
The support vector model was chosen for its strong footing in high-dimensional classification problems and its ability to construct versatile decision boundaries using different Kernel methods. After preprocessing which included scaling and PCA-based dimensionality reduction our SVM was tuned using a halving grid search to effectively provide us with the most effective hyperparameters in fields such as kernel type, C value, gamma value, and class-weighting type. Through testing our best parameters found would be a RBF kernel type, ‘scale’ gamma type, a C value of 100, and lastly a class weight of None. With these values our final model achieved high performance on a majority of high density classes.

Precision and recall values reached near or above 0.97 for over-represented classes such as DDos, DoS-Hulk, and Slowloris (Figure 1A) while still averaging about 0.98 overall accuracy. Due to the computation cost of SVM training, requiring roughly 20+ hours when using the full dataset, we instead evaluated our final model using a single cross-validation fold. The fold was constructed to preserve a proportional representation of all attack categories to ensure the training set still reflected the overall class distribution of the full dataset.

Unfortunately we found the SVM mode struggles significantly with most minority classes such as Bot and Web based attacks, in some cases even yielding 0.00 precision and recall in these categories. While this outcome demonstrates effectiveness in detecting high density attack types it also perfectly exemplifies a known limitation of SVMs which struggle significantly with imbalance datasets, leading to weak discrimination of minority attack types.



```
                           precision    recall  f1-score   support

                  BENIGN       0.99      0.98      0.99     45431
                     Bot       0.84      0.37      0.52        43
                    DDoS       1.00      0.98      0.99      2582
           DoS GoldenEye       1.00      0.96      0.98       207
                DoS Hulk       0.99      0.92      0.95      4622
        DoS Slowhttptest       0.93      0.97      0.95       102
           DoS slowloris       0.98      0.97      0.98       114
             FTP-Patator       0.97      0.99      0.98       155
            Infiltration       0.00      0.00      0.00         1
                PortScan       0.84      1.00      0.91      3179
             SSH-Patator       0.94      0.97      0.95       120
  Web Attack – Brute Force       0.00      0.00      0.00        31
Web Attack – Sql Injection       0.00      0.00      0.00         1
        Web Attack – XSS       1.00      0.08      0.15        12

                accuracy                           0.98     56600
               macro avg       0.75      0.66      0.67     56600
            weighted avg       0.98      0.98      0.98     56600
```



Model Head-to-head Comparison

Recognizing that the values of precision, recall, and f1-score can be misleading when viewed without context, these scores do still provide some insight into the relative performance of disparate models. Using this metric, we found that simplifying our approach to a binary classification using the KNN model provided the best overall of BENIGN vs NOT-BENIGN with scores of 0.99 or 1.0 for all categories of evaluation. All other models performed similarly to each other and all struggled with attack types where sample data was limited below 1000 examples.

Testing Information

Techniques used

Precision:

Precision measures how many predicted positives are actually correct, helping evaluate how well a model avoids false positives.

Recall:

Recall measures how many actual positives the model successfully identifies, showing how well it avoids missing important cases.

F1-Score:

F1-Score combines precision and recall into a single metric, useful when you need a balanced view in the presence of class imbalance.

Confusion Matrix:

Confusion Matrix summarizes predictions into true/false positives and negatives, giving a clear picture of where the model makes mistakes.

ROC Curve:

Receiver Operating Characteristic (ROC) Curve shows the trade-off between true-positive and false-positive rates across thresholds, helping assess a classifier's ability to separate classes.

SHAP:

SHapley Additive exPlanations is an approach that can be used to assign an importance value to features. This can help identify which features are useful in conveying information and which are noise or otherwise can be removed without degrading model performance.

Cross-fold validation:

Cross fold validation is a resampling technique used to divide a dataset into multiple "folds" of data that can then be used for repeated training and validation of a model. In our implementation a model is trained 10 times each using a different fold as the validating set while training on other folds. Cross validation can provide useful insights when working with significant class imbalance, as is the case with the CIC-IDS2017 dataset (Figure 1A).

Cross-fold validation played a key role in the model training of Neural network and Support vector models. This technique was especially useful for models which are particularly sensitive to class imbalance. We primarily used cross fold validation during our tuning phase, as applying it to our full dataset would be computationally expensive given the scale of our dataset.

Issues and Limitations

Analysis of the CIC dataset revealed several challenges that impacted model performance. The dataset exhibits significant class imbalance, causing minority classes to be poorly represented and resulting in lower predictive performance for these categories. Additionally, most of the information available to the models is concentrated in a few primary features, meaning that much of the model complexity contributes little to overall performance. For example, on the XGB Classifier, dropping up to 40% of the feature space had negligible effect on results.

Another limitation is that all attacks in the dataset originate from outside the victim network, which may reduce the model's ability to identify internal attacks. Furthermore, review of ROC curves indicated near-perfect discriminative ability across most models. While this may initially appear desirable, it is likely unrealistic and could point to issues such as data leakage or overfitting. Cross-fold validation was employed to assess consistency, with results suggesting that the root cause may lie in the simplicity of the dataset, however further testing is needed to confirm this.

User Documentation

As this is a primarily research based project, this report in conjunction with the code used to generate these models serves as the user documentation. All models described in this document can be recreated by uploading the .ipynb file to Colab, connecting to a high-ram instance, and running the cells in order.

Transition Plan/Next Steps

Next steps for this project include improving model performance and explainability and ultimately deploying the model in a network monitoring system. Performance improvements would involve further hyperparameter tuning and exploring additional models, including ensemble methods, which will require additional time and compute resources. For explainability, we need to conduct deeper analysis and address issues identified in the CIC dataset, such as class imbalance and the concentration of information in only a few features. This may also require collecting more data or engineering more meaningful features.

We may further enhance the model by synthesizing or gathering data for additional attack types. Finally, although the current model performs adequately, we did not have time to design a deployment environment. Building a functional monitoring system and a test network with simulated attacks will require substantial additional development, research, and resources.

Acknowledgments

We extend our sincere gratitude to our faculty advisor, Dr. Joseph Wilson, for his continuous support and guidance throughout this project. His clarity in emphasizing data understanding, model evaluation, and simplicity in engineering decisions helped shape the direction of our work. Dr. Wilson's feedback on cross-validation, feature analysis, and performance evaluation, especially his emphasis on confusion matrices and ROC interpretation, was crucial to our progress and greatly strengthened the quality of our final system.

We also thank Dr. Thomas for her guidance in managing the course structure and ensuring that project expectations were clear. Her organization of presentation requirements, team roles, and documentation standards helped our group stay aligned and understand our responsibilities throughout the semester.

Biography

Austin Rhee (Project Manager): Austin Rhee is a lover of all things computer science, with a particular focus on server administration and networking. Hands on experience providing technical support, coupled with experimentation in server management & cybersecurity has provided foundational experience to draw on. Austin is currently exploring career opportunities related to technical support for on-premises and cloud based server management, in addition to cybersecurity roles.

Brian McDowell (Scrum Master): Brian McDowell has a professional background in Facilities Operations and Maintenance with a focus on Mission Critical facilities. In this capacity Brian leverages an array of technologies to enable teams of engineers, technicians, and project managers in the construction, operation, and maintenance of buildings and assets to provide high assurance of system availability to end users.

Dillon Kilgallon (Back End Developer): Dillon Kilgallon is an IT professional with a strong foundation in technical support and system administration, now focused on transitioning into cybersecurity. Passionate about understanding how systems operate and securing them, he is building on his experience with networking, cloud technologies, and IT infrastructure to move toward roles that combine practical IT expertise with a focus on safeguarding digital environments. Dillon is committed to continuous learning and professional growth, actively pursuing certifications and training to advance in the field of cybersecurity.

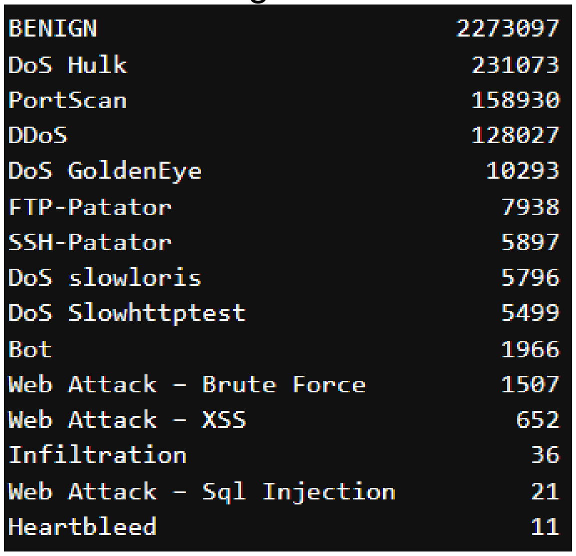
Michael Tumminia (Back End Developer): Michael Tumminia is a graduating Computer Science major at the University of Florida’s Herbert Wertheim College of Engineering with experience in software engineering, systems programming, and cybersecurity. He previously interned for two summers at Custom Earth Promos, where he worked on React-based front-end development and SEO-driven improvements for e-commerce platforms. Throughout his studies, he has contributed to projects ranging from a Unity-based 3D audio visualization tool to a large-scale Pokémon battle simulator, reflecting his passion for the intersection of music and technology. He is currently seeking full-time software engineering opportunities and enjoys producing electronic music and exploring audio programming in his free time.

Safeah Hammosh (Back End Developer): Safeah Hammosh is a graduating Computer Science student at the University of Florida with broad interests that span software development, data analysis, and user interface design. She enjoys learning across different areas of technology and is especially motivated by roles that combine problem-solving with creativity. Throughout her studies, she has contributed to a variety of projects that strengthened her skills in programming, analytical thinking, and designing user-friendly interfaces. As she prepares for graduation, she is seeking opportunities in software engineering, UI design, or other technical roles that allow her to continue developing her skills and discovering what she enjoys most.

Appendix A

Dataset Features

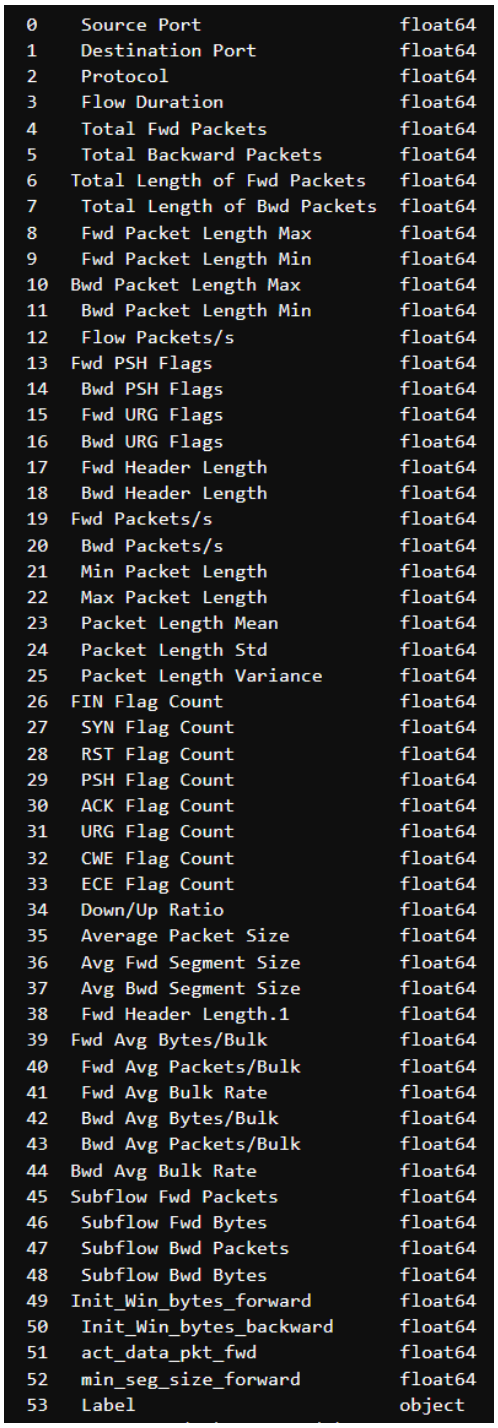
Figure 1A



```
BENIGN                          2273097
DoS Hulk                         231073
PortScan                         158930
DDoS                             128027
DoS GoldenEye                     10293
FTP-Patator                        7938
SSH-Patator                        5897
DoS slowloris                      5796
DoS Slowhttptest                   5499
Bot                                1966
Web Attack – Brute Force           1507
Web Attack – XSS                    652
Infiltration                         36
Web Attack – Sql Injection           21
Heartbleed                           11
```

Prevalence of attack classes

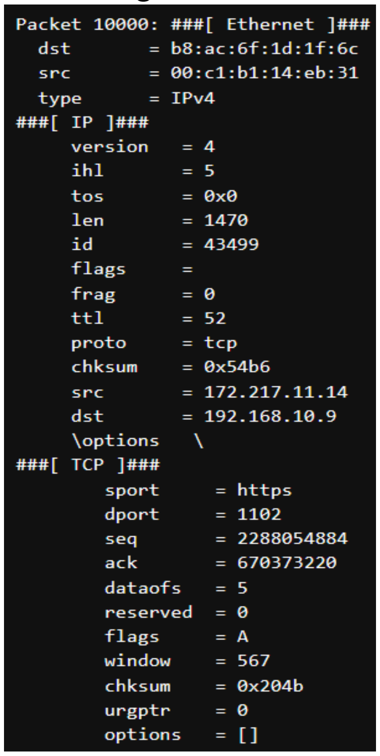
Figure 2A



```
0    Source Port                   float64
1    Destination Port              float64
2    Protocol                      float64
3    Flow Duration                 float64
4    Total Fwd Packets             float64
5    Total Backward Packets        float64
6   Total Length of Fwd Packets    float64
7    Total Length of Bwd Packets   float64
8    Fwd Packet Length Max         float64
9    Fwd Packet Length Min         float64
10  Bwd Packet Length Max          float64
11   Bwd Packet Length Min         float64
12   Flow Packets/s                float64
13  Fwd PSH Flags                  float64
14   Bwd PSH Flags                 float64
15   Fwd URG Flags                 float64
16   Bwd URG Flags                 float64
17   Fwd Header Length             float64
18   Bwd Header Length             float64
19  Fwd Packets/s                  float64
20   Bwd Packets/s                 float64
21   Min Packet Length             float64
22   Max Packet Length             float64
23   Packet Length Mean            float64
24   Packet Length Std             float64
25   Packet Length Variance        float64
26  FIN Flag Count                 float64
27   SYN Flag Count                float64
28   RST Flag Count                float64
29   PSH Flag Count                float64
30   ACK Flag Count                float64
31   URG Flag Count                float64
32   CWE Flag Count                float64
33   ECE Flag Count                float64
34   Down/Up Ratio                 float64
35   Average Packet Size           float64
36   Avg Fwd Segment Size          float64
37   Avg Bwd Segment Size          float64
38   Fwd Header Length.1           float64
39  Fwd Avg Bytes/Bulk             float64
40   Fwd Avg Packets/Bulk          float64
41   Fwd Avg Bulk Rate             float64
42   Bwd Avg Bytes/Bulk            float64
43   Bwd Avg Packets/Bulk          float64
44  Bwd Avg Bulk Rate              float64
45  Subflow Fwd Packets            float64
46   Subflow Fwd Bytes             float64
47   Subflow Bwd Packets           float64
48   Subflow Bwd Bytes             float64
49  Init_Win_bytes_forward         float64
50   Init_Win_bytes_backward       float64
51   act_data_pkt_fwd              float64
52   min_seg_size_forward          float64
53   Label                         object
```

Features used for model training

Figure 3A



```
Packet 10000: ###[ Ethernet ]###
  dst       = b8:ac:6f:1d:1f:6c
  src       = 00:c1:b1:14:eb:31
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 1470
     id        = 43499
     flags     =
     frag      = 0
     ttl       = 52
     proto     = tcp
     chksum    = 0x54b6
     src       = 172.217.11.14
     dst       = 192.168.10.9
     \options   \
###[ TCP ]###
        sport     = https
        dport     = 1102
        seq       = 2288054884
        ack       = 670373220
        dataofs   = 5
        reserved  = 0
        flags     = A
        window    = 567
        chksum    = 0x204b
        urgptr    = 0
        options   = []
```

Example packet