Unified Mentor Project Documentation: Web Attack Traffic Analysis and Detection

- Adimalla Nithin Siddhartha

Introduction: This document details the process undertaken to analyze web traffic data and build a preliminary model for detecting web attacks. The project involved loading and cleaning the data, engineering relevant features, performing exploratory data analysis to understand traffic patterns, and training a machine learning model to identify potential threats. The dataset used for this project is a CloudWatch traffic log containing various attributes related to network connections.

1. Data Loading and Initial Inspection:

- **Objective:** To load the dataset into a usable format and get a first look at its structure and content.
- **Process:** We began by using the pandas library in Python to read the data from the provided CSV file, located at CloudWatch_Traffic_Web_Attack.csv. The pd.read_csv() function was used for this purpose, creating a pandas DataFrame named data.
- **Initial Look:** We then used the data.head() method to display the first five rows of the DataFrame. This allowed us to quickly examine the columns present, the data types, and a sample of the values within each column. This initial inspection is crucial for understanding the raw data and identifying potential issues early in the process.
- **Observation:** The initial inspection revealed columns such as bytes_in, bytes_out, creation_time, end_time, src_ip, src_ip_country_code, protocol, response.code, dst_port, dst_ip, rule_names, observation_name, source.meta, source.name, time, and detection_types. We also observed that the time-related columns were initially loaded as object data types, which would require conversion for time-series analysis.

2. Data Cleaning:

- **Objective:** To ensure the quality and consistency of the data by handling duplicates, correcting data types, and standardizing text fields.
- **Process:**

- **Removing Duplicate Rows:** We used the df.drop_duplicates() method to identify and remove any rows that were exact duplicates across all columns. This step is important to prevent biased analysis and model training that could result from redundant data points. After removing duplicates, we printed the shape of the DataFrame to see how many rows remained. In this case, no duplicates were found, and the DataFrame retained its original 282 entries.
- **Correcting Data Types:** The columns creation_time, end_time, and time were initially loaded as object types. For any analysis involving time intervals or time-series plotting, these columns needed to be in a datetime format. We used the pd.to_datetime() function to convert these columns to datetime objects. The df.info() method was used after this conversion to verify that the data types were correctly updated to datetime64[ns, UTC].
- **Standardizing Text Data:** To ensure consistency in categorical text fields, we iterated through a predefined list of text columns (src_ip, src_ip_country_code, protocol, rule_names, observation_name, source.meta, source.name, detection_types). For each of these columns, we converted all entries to lowercase using .str.lower() and removed leading and trailing whitespace using .str.strip(). This standardization helps in accurate grouping and analysis of categorical data, preventing variations due to capitalization or extra spaces. We again used df.info() to confirm that the data types remained as object (string) but with standardized content.

- **Observation:** The data cleaning steps successfully prepared the dataset for further analysis and modeling by ensuring data uniqueness, correct data types for time-based operations, and consistent formatting of text data. The dataset size remained at 282 entries, confirming no duplicate records were present.

3. Feature Engineering:

- **Objective:** To create new features from existing ones that could provide more valuable information for detecting web attacks.

- **Process:** Based on the available columns, we engineered two potentially informative features:
- **Duration:** We calculated the duration of each network event by subtracting the creation_time from the end_time. The .dt.total_seconds() accessor was used to get the duration in seconds. This feature, stored in a new column named duration, could be relevant as attack traffic might have different duration patterns compared to normal traffic.
- **Total Bytes:** We calculated the total number of bytes transferred in each event by summing the values in the bytes_in and bytes_out columns. This new feature, named total_bytes, provides a single metric for the volume of data exchanged during a connection, which could be indicative of unusual activity.
- **Observation:** The creation of duration and total_bytes added new dimensions to the dataset, providing aggregate measures of time and data volume for each event. Displaying the head of the new DataFrame (df_fe) confirmed the successful addition of these columns.

4. Exploratory Data Analysis (EDA):

- **Objective:** To gain insights into the characteristics of the data, identify patterns, and understand the distribution of different features.
- **Process:** We used visualizations to explore the distributions of various features:
- **Distribution of Total Bytes and Duration:** We created histograms for both total_bytes and duration using seaborn.histplot. These plots helped us understand the frequency distribution of data volume and event length.
- **Distribution of Source IP Country Code and Protocol:** We used seaborn.countplot to visualize the distribution of src_ip_country_code and protocol. These bar plots showed us the most frequent source countries and protocols in the dataset.
- **Distribution of Response Codes and Destination Ports:** Similar count plots were generated for response.code and dst_port to understand the common server responses and targeted ports.

- **Distribution of Other Categorical Features:** We also visualized the distributions of rule_names, observation_name, source.meta, source.name, and detection_types using count plots to see the frequency of different rule triggers, observations, data sources, and detection methods.
- **Observations from EDA:**
    - The distribution of total_bytes was heavily skewed, with a large number of events having low byte counts and a few outliers with very high values. These outliers could potentially represent unusual or malicious traffic.
    - The duration distribution was concentrated at a single value (600 seconds), suggesting a fixed interval for data logging rather than variable connection durations. This limits the usefulness of duration as a distinguishing feature in this dataset.
    - The src_ip_country_code plot showed that traffic originated from several countries, with a significant portion coming from the United States (us) and Canada (ca).
    - The protocol distribution was almost entirely HTTPS, indicating that the recorded traffic primarily uses this secure protocol.
    - The response.code and dst_port plots were dominated by a single category (200 for response code and 443 for destination port), suggesting that most of the observed traffic involved successful HTTPS connections on the standard web port.
    - The analysis of rule_names, observation_name, source.meta, source.name, and detection_types revealed that the vast majority of records shared the same values across these columns ('suspicious web traffic', 'adversary infrastructure interaction', 'aws_vpc_flow', 'prod_webserver', and 'waf_rule'). This strong homogeneity in these "detection" related columns is a key observation.

5. Data Preprocessing for Modeling:

- **Objective:** To transform the data into a format suitable for training a machine learning model.

- **Process:** This involved several steps:

  Feature Selection: We selected the numerical features (bytes_in, bytes_out, duration, total_bytes) and the categorical features (src_ip_country_code, protocol, response.code, dst_port, rule_names, observation _name, source.meta, source.name, detection_types) to be used as input for the model.

  - **Handling Categorical Features (One-Hot Encoding):** Machine learning algorithms typically require numerical input. We used pd.get_dummies() to perform one-hot encoding on the selected categorical columns. This process creates new binary columns for each unique category within a categorical feature. The drop_first=True argument was used to avoid multicollinearity.
  - **Scaling Numerical Features:** To prevent features with larger values from disproportionately influencing the model, we scaled the numerical features using StandardScaler. This method standardizes features by removing the mean and scaling to unit variance, resulting in features with a mean of 0 and a standard deviation of 1.

- **Observation:** The data preprocessing steps successfully transformed the raw and engineered features into a numerical and scaled format ready for model training. The one-hot encoding expanded the number of columns significantly due to the creation of binary columns for each category.

6. Model Training and Evaluation (Random Forest Classifier):

- **Objective:** To train a machine learning model to classify web traffic as either normal or an attack and evaluate its performance.
- **Process:**
- **Defining Target Variable:** We identified the detection_types column as our target variable (y), which indicates whether a traffic event was flagged as a 'waf_rule'. The remaining preprocessed features formed our input data (X).

- **Splitting Data:** We split the data into training and testing sets using train_test_split from sklearn.model_selection. A test_size of 0.2 (20% of the data for testing) and random_state=42 for reproducibility were used. We also used stratify=y to ensure that the proportion of the target variable is the same in both the training and testing sets, which is important for imbalanced datasets.
- **Model Training:** We chose a Random Forest Classifier (RandomForestClassifier from sklearn.ensemble) for its robustness and ability to handle both numerical and categorical data (after encoding). We initialized the model with n_estimators=100 (number of trees in the forest) and random_state=42. The model was then trained on the training data using model.fit(X_train, y_train).
- **Model Evaluation:** After training, we used the trained model to predict the detection_types on the test set (y_pred = model.predict(X_test)). We then evaluated the model's performance using
- **Accuracy:** Calculated using accuracy_score(y_test, y_pred), which measures the proportion of correctly classified instances.
- **Classification Report:** Generated using classification_report(y_test, y_pred), which provides precision, recall, and f1-score for each class, as well as overall accuracy and macro/weighted averages.
- **Confusion Matrix:** Created using confusion_matrix(y_test, y_pred) and visualized as a heatmap using seaborn.heatmap. The confusion matrix shows the number of true positive, true negative, false positive, and false negative predictions.
- **Observations from Model Training and Evaluation:**
    - The Random Forest Classifier achieved an accuracy of 1.0, indicating perfect classification on the test set.
    - The classification report showed perfect scores (1.0) for precision, recall, and f1-score for the 'waf_rule' class.

- The confusion matrix displayed all predictions on the test set falling into the true positive or true negative categories for the 'waf_rule' class, with no misclassifications.
-

7. Visualizations of Analysis Results:

- **Objective:** To visually represent key findings from the analysis, including the distribution of detection types by country and web traffic over time.
- **Process:**
  - **Detection Types by Country:** We grouped the data by src_ip_country_code and detection_types and counted the occurrences. A bar plot was then generated using seaborn.barplot to visualize the distribution of detection types across different source countries.
  - **Web Traffic Over Time:** We counted the occurrences of events at each unique timestamp in the time column and sorted the results by time. A line plot was created using matplotlib.pyplot.plot to show the trend of web traffic volume over time.
- **Observations from Visualizations:**
  - The barchart of detection types by country code clearly showed that 'waf_rule' was the only detection type present in the dataset and that it was distributed across various source countries, with the US having the highest count. This reinforces the observation from EDA about the homogeneity of the detection_types column.
  - The plot of web traffic over time showed fluctuations in the number of events recorded at different time points within the dataset's timeframe. While there were variations, there wasn't a clear, dramatic spike that would unequivocally indicate a large-scale, sustained attack based solely on event volume.

Overall Conclusion and Reflections:

The project successfully demonstrated the process of analyzing web traffic data and building a classification model. The data cleaning and feature engineering steps were crucial in preparing the

data for modeling. The exploratory data analysis provided valuable insights into the characteristics of the traffic and the distribution of different attributes.

The Random Forest Classifier achieved perfect accuracy on the test set. However, this result should be interpreted with caution due to the nature of the dataset. The strong homogeneity in the 'detection_types' column, where almost all records are labeled as 'waf_rule', suggests that the model is primarily learning to predict this dominant class. While it effectively identifies instances matching the patterns present in this specific dataset, its ability to generalize to a more diverse range of web attacks or to distinguish between different types of attacks is limited by the dataset's scope.