**NETWORK FLOW AND THREAT DETECTION SYSTEM**

**Introduction:**

Critical infrastructure systems perform essential functions for our national economy, health, and security, directly impacting commerce, government operations, and the daily lives of people. With the increasing complexity of cyber threats, there is a pressing need for a scalable, reliable, and robust system to monitor and detect potential threats within these infrastructure networks to detect and mitigate potential security incidents. This project utilizes three distinct network traffic datasets IOT23, CTU13, and CESNET-TLS22 to develop a comprehensive machine learning-based system for network anomaly and threat detection. IOT23 captures network traffic data from IoT devices, focusing on malicious and benign traffic in smart environments. CTU13 provides a diverse set of network flow data, including malicious behavior such as botnet attacks. CESNET-TLS22 offers detailed network traffic features with a particular emphasis on encrypted communication channels. By integrating data from these varied sources, the project aims to create a model that can accurately identify malicious network patterns across different network architectures and protocols. Experimental results using real-world data demonstrate the effectiveness of this system in network monitoring, anomaly detection, and threat mitigation, making it an essential tool for securing critical infrastructure systems against evolving cyber threats. This system will play a critical role in advancing cybersecurity measures by detecting threats in both IoT and conventional networks.

**Data preparation:**

In the data preparation process for Dataset-1 (IOT23), we began by loading and combining multiple dataset files into a single Data Frame for unified analysis. The target column, originally named tunnel\_parents label detailed-label, was renamed to label for easier reference. To standardize the target labels, we mapped them to either "benign" or "malicious" by identifying relevant keywords in the labels. Unnecessary columns, such as Unnamed: 0, were removed to streamline the dataset. We handled missing values by filling them with placeholders where appropriate, and converted data types to ensure numerical columns, such as orig\_bytes and resp\_bytes, were treated correctly. The timestamp column (ts) was transformed into a datetime format to facilitate time-based analysis. Additionally, categorical variables were encoded using one-hot encoding to prepare the dataset for machine learning models. These steps collectively ensured that the dataset was clean, consistent, and ready for further exploratory data analysis and model training.

In contrast, for Dataset-2 (CTU-13 Dataset), we accessed multiple. binetflow files across different folders within the main directory. Each file was iteratively loaded into a Data Frame, where the relevant columns were specified using predefined names: StartTime, Dur, Protocol, SrcAddr, Sport, Dir, DstAddr, Dport, State, sTos, dTos, TotPkts, TotBytes, SrcBytes, and Label. The data loading was performed with error handling to skip empty files and only add non-empty Data Frames to a list, which was then concatenated into a single combined flows Data Frame for comprehensive analysis. After combining the files, the resulting Data Frame structure was inspected, revealing 19,976,687 entries across 15 columns. The Protocol column was analysed to understand the distribution of different network protocols, such as UDP, TCP, and ICMP, among others. Preliminary data cleaning steps included converting the Start Time column to a datetime format for time-based analysis and handling any missing or anomalous values in fields like SrcBytes and DstBytes to ensure consistency across numerical and categorical features. These steps prepared the dataset for exploratory data analysis, feature engineering, and further modelling.

Whereas in data preparation for dataset 3, we began by loading the CESNET-TLS22 dataset using Pandas. The dataset contains network flow data with over 5 million entries, making it quite large. We loaded the data from a CSV file located on the local system, which includes columns such as ID, BYTES, BYTES\_REV, PACKETS, PACKETS\_REV, DURATION, PPI, APP, CATEGORY, and various TCP flag fields. These fields provide details on data transfer metrics, packet details, application types, and network categories. Upon examining the dataset with info(), we observed that it has 30 columns with a mix of int64, float64, and object data types, and the memory usage is over 1.2 GB, indicating its considerable size. We also inspected the PPI, APP, and CATEGORY columns to understand the data distributions within these fields. The PPI column contains complex nested lists that describe packet inspection properties, while the APP and CATEGORY columns contain categorical information about applications and types of services or APIs associated with the traffic, showing significant frequency variations across different categories.

Finally, we displayed the first 10 rows to get a quick look at the data, which helped us verify the data structure and content. This initial exploration is essential to identify any data preprocessing steps that may be required, such as handling nested structures, transforming categorical data, or dealing with any missing or erroneous values.

**DATA ANALYSIS AND VISUALIZATION**

In the exploratory data analysis (EDA) for Dataset-1, we started by examining its dimensions and structure. Initially, the dataset contained 16,898 records and 22 columns, which increased to 24 columns after data preparation, indicating the addition or modification of attributes for improved quality. We performed a statistical analysis using the .describe() method to gain insights into numerical columns like id.orig\_p, id.resp\_p, missed\_bytes, and others. This revealed central tendencies and outliers that may need handling during preprocessing.

Next, we computed the correlation matrix to assess linear dependencies between numerical columns, guiding feature selection by identifying redundant variables. We also examined the data types of each column, which will inform preprocessing steps like normalization for numerical data and encoding for categorical data. For categorical fields such as local\_orig, service, and proto, we used .value\_counts() to analyze value distributions. This revealed dominant values (e.g., dns and - in the service column), and placeholders like - in duration and orig\_bytes suggested missing or undefined data.

Lastly, a count plot of the target variable new\_label showed a balanced distribution of benign (8,646) and malicious (8,252) records, indicating suitability for binary classification without significant class imbalance. This EDA process provided valuable insights into the dataset’s structure, feature distributions, and areas for preprocessing, aiding in the development of more effective machine learning models.

A blue rectangular bars with white text

Description automatically generated

**Fig-1**

**Fig-1** illustrates the count plot helps to visually assess the balance of classes in the target variable. It will show how many instances belong to each class (e.g., benign vs. malicious) in the new\_label column. A well-balanced distribution indicates that the dataset is suitable for training a classification model without requiring significant adjustments for class imbalance.

A graph of numbers and lines

Description automatically generated with medium confidence

**Fig-2**

Fig-2 shows the pair plot provides a comprehensive view of the relationships between pairs of features and their distributions across the two target classes (M and B). It helps in identifying:

* **Feature distributions**: Whether the features are evenly distributed or skewed for each class.
* **Class separability**: How well the features can differentiate between the two classes (malicious vs benign).
* **Correlations**: The potential relationships between different pairs of features.

In the initial steps of the EDA in Dataset-2, we examined the structure of the dataset. We then analyzed the statistical characteristics of the data, including summary statistics and correlations, to better understand the distribution and relationships between features.We began by checking the shape of the dataset, confirming it had 135,041 records and 16 columns after preparation. Using the .info() method, we also reviewed the data types, noting the presence of numerical and categorical columns, which will guide the data preprocessing steps.

Next, we analyzed the distribution of values in the categorical columns. We used .value\_counts() to count the occurrences of unique values in columns like StartTime, Protocol, SrcAddr, Sport, Dir, DstAddr, Dport, and State. For example, the Protocol column contained predominantly udp and tcp values, with a small number of icmp entries. Similarly, the SrcAddr and DstAddr columns showed repetitive values, indicating the frequent use of certain IP addresses.We then computed summary statistics using .describe(), which provided the mean, standard deviation, and range for the numerical columns. We observed that some columns, such as sTos and dTos, had consistent values (0.0) for a large number of records, which could suggest potential issues with the data that may need to be addressed during preprocessing.

Furthermore, we checked for missing values using .isnull().sum(). The dTos column had 24,250 missing values, highlighting the need for imputation or removal during data cleaning.Finally, we examined the distribution of the target variable (new\_label), which had a class imbalance with 108,807 "normal" entries and 26,234 "botnet" entries. The class distribution was visualized using a count plot, which indicated that while the dataset was somewhat imbalanced, the difference was not extreme, and the dataset could still be suitable for binary classification.

A blue rectangular bar graph

Description automatically generated

**FIG-3**

**FIG-3** shows sns.countplot(x='new\_label', data=filtered\_df) creates a count plot to visualize the distribution of the new\_label column, showing the frequency of each class (e.g., "normal" and "botnet"). The title is set using plt.title(), and plt.show() displays the plot. This helps in understanding the balance between the different classes in the dataset.

A graph of a number of blue and red dots

Description automatically generated with medium confidence

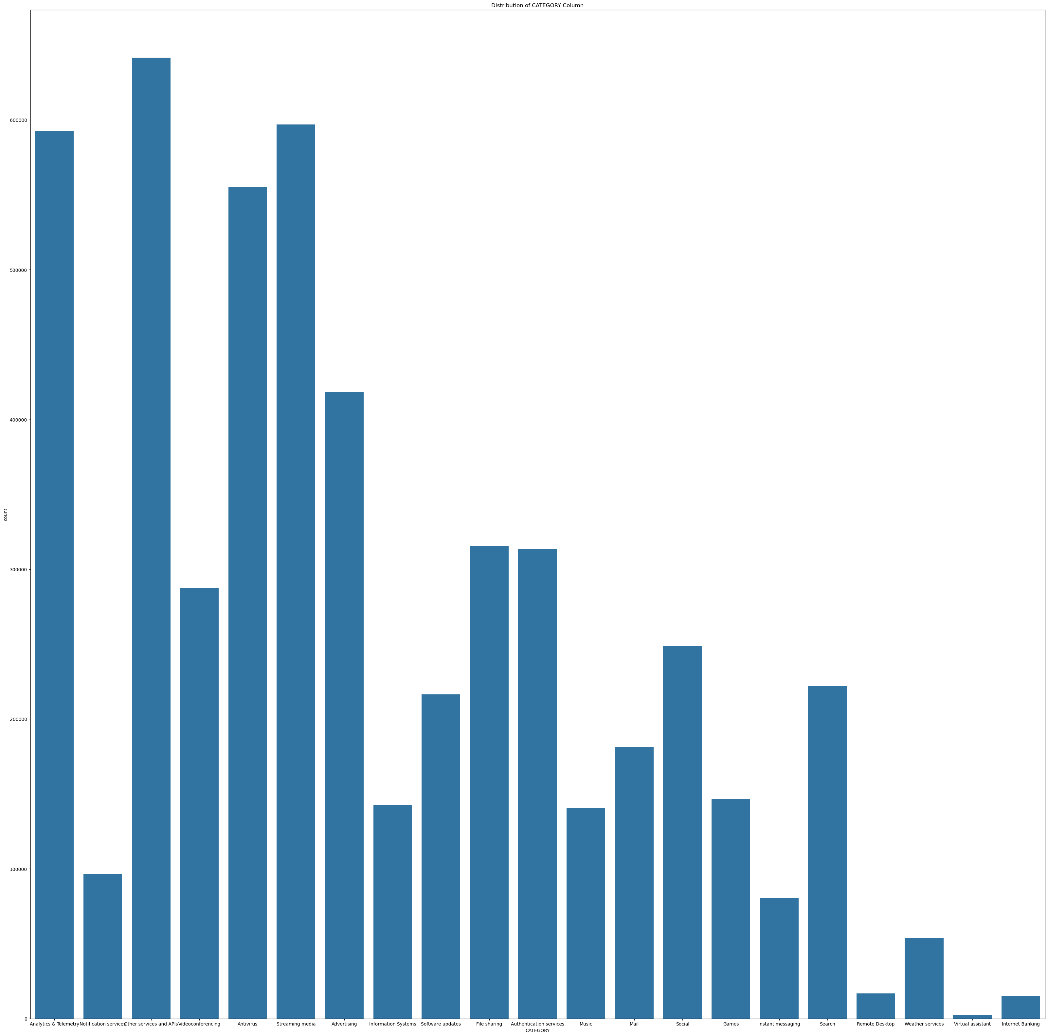
**FIG-4**

**FIG-4** shows resulting pair plot generated by sns.pairplot(filtered\_df, hue='new\_label', vars=['Dur', 'sTos', 'dTos', 'TotPkts', 'TotBytes', 'SrcBytes']) shows the relationships between multiple features in the dataset, with the points colored based on the new\_label (e.g., "normal" or "botnet").

* Each diagonal subplot displays the distribution of a specific feature, with overlaid histograms or kernel density plots for the two classes (normal and botnet).
* Off-diagonal subplots show scatter plots between pairs of features, allowing us to visually inspect the relationships and potential separations between the two classes.

In the initial stage of the exploratory data analysis (EDA), the dataset-3 was examined to understand its structure. The dataset originally had 5,284,744 records with 30 columns, which was reduced to 24 columns after data preparation. The dataset included a mix of numerical and categorical columns, such as BYTES, PACKETS, DURATION, and several flags, while some columns like PPI and APP were of object type. The statistics of the dataset were analysed, which provided insights into the distribution and range of values across different columns. It was observed that most columns had no missing values, indicating a clean dataset.

The dataset also includes various flags and features related to networking and traffic patterns, which were further analysed for correlations. The target variable, CATEGORY, was checked for class distribution, showing that the dataset was imbalanced with categories like "Other services and APIs" having significantly more instances than categories like "Weather services" and "Remote Desktop." These analyses were essential for understanding the dataset’s structure and identifying potential challenges such as imbalanced classes, which would need to be addressed during the preprocessing stage.



**FIG-5**

**FIG-5** shows a count plot to visualize the distribution of the CATEGORY column in the dataset cesnet\_tls22\_df using Seaborn. The figure size is set to a very large dimension of 40 inches by 40 inches, ensuring that the plot is sufficiently large for clear viewing. The sns.countplot() function counts the occurrences of each unique value in the CATEGORY column and displays them in a bar chart. The plot is titled "Distribution of CATEGORY Column" for context, and plt.show() is used to display the plot. However, the figure size may need to be adjusted based on the actual content and preferences for readability, with smaller dimensions like figsize=(12, 8) being a more common choice.

A graph of a number of blue and red dots

Description automatically generated with medium confidence

**FIG-6**

The provided fig-6 shows a pair plot using Seaborn to visualize the relationships between multiple numerical variables in the filtered\_df dataset.

**DATA PREPROCESSING**

**pre-processing stage the dataset-1:**

The data pre-processing stage the dataset-1 involves converting categorical columns to numerical values for model compatibility, specifically for the "new\_label" column. After this conversion, the dataset is split into training and testing sets using an 80-20 train-test split ratio. This ensures that a significant portion of the data is available for training the model, while a smaller but representative portion is reserved for testing its performance.

Subsequently, a standard scaler is applied to standardize the numerical features, aligning them to a common scale. This normalization is crucial for maintaining consistency and optimizing model performance, particularly for algorithms sensitive to feature scale. Standardization also mitigates the impact of feature disparities during model training, improving convergence rates and enhancing the accuracy of predictions.

## ***Data Exploration and Column Exclusion and Balancing***

In exploring the data, columns like “id.resp\_p” and “id.orig\_p” are analysed, revealing frequency distributions across different port numbers. To streamline the dataset, unnecessary columns—such as 'Unnamed: 0,' 'tunnel\_parents label detailed-label,' 'label,' 'uid,' 'id.orig\_h,' 'ts,' and 'service'—are removed. This focus on the most pertinent features reduces noise and simplifies further analysis.

For balanced representation across target classes, the dataset is further refined by balancing samples from each "new\_label" category. The dataset is grouped by "new\_label," and a minimum of 8,252 samples per category is taken to ensure uniformity. This step prevents model bias towards any specific class, especially when the dataset is imbalanced. The resulting balanced dataset, stored in balanced\_df, provides a more equitable basis for model training and performance evaluation.

## ***Handling Missing and Null Values***

After removing the extraneous columns, the dataset undergoes further cleaning to address missing and null values. Initial rows with any null values are removed. Additionally, certain columns contain "-" values, which are not meaningful for analysis. These placeholder values are replaced with NaN (Not a Number), and columns with high nullity, such as 'local\_orig' and 'local\_resp,' are subsequently excluded. This refined dataset ensures that only relevant, non-null data is passed forward for modeling.

## ***Data Type Conversion***

To improve data compatibility, specific columns are converted to appropriate data types. Continuous variables like 'duration,' 'resp\_bytes,' and 'orig\_bytes' are cast to floats, given their quantitative nature. Meanwhile, 'id.orig\_p' and 'id.resp\_p,' being identifiers, are retained as object types to preserve their distinct categorical values. This final cleaned and transformed dataset is now well-prepared for further analysis, feature engineering, or model training steps, maximizing both efficiency and accuracy in the subsequent stages of the pipeline.

## **pre-processing phase of the (CTU13) dataset-2:**

In the pre-processing phase of the CTU13 dataset, we began by converting categorical columns to numerical values for compatibility with machine learning models. The target variable, "new\_label," was label-encoded to transform its values into numerical labels. Additionally, categorical columns such as "Protocol," "SrcAddr," "Sport," "Dir," "DstAddr," "State," and "Dport" were encoded using the LabelEncoder function, ensuring that each unique value was assigned a corresponding numerical label. This step helps improve the dataset's compatibility with algorithms that require numerical inputs.

### ***Removing Irrelevant Columns***

After encoding, we examined the dataset for irrelevant or redundant columns. Columns such as "StartTime," "Label," "sTos," and "dTos" were removed to focus on essential features. To maintain data quality, any rows with missing values were removed. We further scaled the numerical features using a standard scaler to normalize the data, ensuring a consistent scale across features, which enhances the stability and performance of machine learning models.

### ***Handling Missing Values***

To address class imbalance, we balanced the dataset by taking a minimum of 10,000 samples from each class in the target variable, "new\_label." This was achieved using the groupby function, which grouped the data by each label, followed by sampling from each group. This step ensured a balanced dataset, reducing potential bias in model predictions due to unequal class distributions. This final pre-processed dataset provides a clean, balanced, and numerically encoded foundation for further analysis and model training.

## **pre-processing phase of the dataset-3:**

The data pre-processing and validation steps for the **cesnet\_tls22\_df** dataset began with encoding the target variable, **"new\_label"**, into numerical values to ensure compatibility with machine learning models. Next, the dataset was divided into training and testing subsets using an **80/20 split**, allocating 80% of the data for training and 20% for testing. This split helps provide a basis for evaluating model performance on unseen data. Additionally, a **Standard Scaler** was applied to standardize the numerical data, ensuring uniform feature scales and reducing potential biases in model training.

To further clean the dataset, an inspection was conducted using **head()** and **info()** functions, followed by the removal of irrelevant columns such as **"ID"**. This step helped maintain a focus on relevant features and reduce unnecessary data. Two categorical columns, **"APP"** and **"PPI,"** were then converted to numerical values using label encoding. For consistency, the target column **"CATEGORY"** was also label-encoded to ensure uniform data structure.

Class imbalance in the dataset was addressed by balancing it through sampling a minimum of **10,000 records** from each class in **"new\_label"**. Using **groupby** and **sample** functions, this approach ensured that the dataset was evenly distributed across classes, thus preparing it for effective model training. Overall, the data has been pre-processed, cleaned, standardized, and balanced, making it ready for subsequent model training and validation.

**FEATURE EXTRACTION**