

# AI-Driven Cloud Security Threat Detection Using Data Mining Techniques

Girish Allada

NetID: galla145

December 12, 2025

## Abstract

Cloud computing environments generate massive volumes of network and system logs, making manual security monitoring infeasible. This project presents an AI-driven cloud security threat detection system that applies data mining and machine learning techniques to detect malicious activities from large-scale cloud traffic data. Using the CICIDS2017 dataset, the system performs data preprocessing, exploratory analysis, supervised classification, unsupervised clustering, and association rule mining. The results demonstrate that machine learning models can effectively distinguish benign from malicious traffic while unsupervised techniques reveal hidden attack patterns and anomalies, supporting proactive cloud security monitoring.

## 1 Introduction

Cloud infrastructures are increasingly targeted by cyber attacks due to their scalability, shared resources, and complex access patterns. Traditional rule-based intrusion detection systems struggle to adapt to evolving threats and high-dimensional data. Consequently, intelligent data-driven approaches are required to automatically detect and analyze suspicious behavior.

The goal of this mini-project is to build an AI-driven cloud security threat detection system that analyzes cloud log data to identify anomalous and malicious activity. Using data mining techniques, the project focuses on extracting meaningful patterns from large datasets, classifying traffic into benign and malicious categories, and discovering previously unknown attack behaviors.

The CICIDS2017 dataset is used as a realistic benchmark for cloud and network intrusion detection. The project emphasizes end-to-end data processing, model evaluation, and actionable security insights.

## 2 Dataset Description

The CICIDS2017 combined dataset is a large-scale network intrusion dataset containing labeled traffic flows representing both benign user behavior and multiple attack types, such as DoS, DDoS, brute force, infiltration, and web attacks. The dataset includes millions of network flow records and numerical features describing packet statistics, flow duration, byte counts, and traffic rates. Due to its large size (approximately 831 MB), the dataset is processed using PySpark to enable scalable and memory-efficient analysis.

## 3 Methodology

### 3.1 Data Preprocessing

Data preprocessing is a critical step due to noise, missing values, and inconsistent formats in raw cloud logs. The following steps were applied:

- Removal of features with excessive missing values
- Handling of infinite and invalid values
- Removal of rows containing remaining null values
- Conversion of attack labels into a binary classification problem:
  - 0: Benign traffic
  - 1: Malicious traffic

The cleaned dataset was stored in Parquet format to improve storage efficiency and processing speed.

## 4 Data Cleaning and Exploratory Data Analysis

This section describes the data loading, cleaning, and basic exploratory data analysis (EDA) steps performed using Apache Spark (PySpark) to handle a large-scale dataset efficiently.

## 4.1 Library Installation

- The `pyspark` library is installed to enable distributed data processing using Apache Spark.
- The `spark` package is also installed to ensure Spark-related dependencies are available.

These installations are necessary because Spark is not available by default in standard Python environments.

## 4.2 Importing Required Libraries

The following libraries are imported:

- `SparkSession` to initialize and manage the Spark application.
- Spark SQL functions such as `col`, `when`, `isnan`, `count`, and `lit` for data manipulation and analysis.
- The `os` module for interacting with the file system.

## 4.3 Initializing Spark Session

A Spark session is created to serve as the entry point for all Spark operations. This session allows the notebook to read large datasets, perform transformations, and execute distributed computations efficiently.

## 4.4 Loading the Dataset

The dataset is loaded from a CSV file using Spark's `read.csv()` method with the following options:

- `header=True` to treat the first row as column names.
- `inferSchema=True` to automatically detect data types.

This approach allows Spark to efficiently load and structure large CSV files.

## 4.5 Initial Dataset Inspection

Basic inspection steps are performed:

- Displaying the schema to understand column data types.
- Counting the total number of records in the dataset.

These steps provide an overview of the dataset's size and structure.

## 4.6 Handling Missing Values

To identify missing data:

- The number of null or NaN values is computed for each column.
- Columns with more than 50% missing values are identified.

Columns exceeding this threshold are dropped to improve data quality and reduce noise in subsequent analysis.

## 4.7 Removing Duplicate Records

Duplicate rows are removed using Spark's `dropDuplicates()` method. The dataset size is recorded before and after this operation to quantify the number of duplicates removed.

## 4.8 Label Column Detection

The notebook automatically detects the target (label) column by searching for column names containing keywords such as:

- `label`
- `attack`
- `class`

This step ensures flexibility when working with datasets that may use different naming conventions for the target variable.

## 4.9 Basic Exploratory Data Analysis

Basic EDA operations are performed, including:

- Counting records per class label to examine class distribution.
- Computing summary statistics such as mean, minimum, maximum, and standard deviation for numerical features.

These statistics provide insight into data imbalance, scale, and variability.

## 4.10 Saving the Cleaned Dataset

Finally, the cleaned dataset is saved in **Parquet** format:

- Parquet is a columnar storage format optimized for Spark.
- It reduces storage size and improves performance in downstream machine learning and analytics tasks.

The saved file is intended for use in subsequent modeling and a

# 5 Feature Engineering Using PySpark

This section focuses on transforming the cleaned dataset into a suitable format for machine learning models. Feature engineering is performed using PySpark to ensure scalability and efficiency when handling large datasets.

## 5.1 Importing Required Libraries

The necessary PySpark libraries are imported to support feature engineering tasks:

- `SparkSession` for managing Spark operations.
- `VectorAssembler` for combining multiple feature columns into a single feature vector.
- `StringIndexer` for converting categorical variables into numerical form.
- `StandardScaler` for feature normalization.
- Spark SQL functions such as `col` for column-based transformations.

These tools are essential for preparing structured data for machine learning algorithms in Spark.

## 5.2 Loading the Cleaned Dataset

The cleaned dataset generated in the previous stage is loaded from a **Parquet** file. Using Parquet format improves read performance and ensures compatibility with Spark's ML pipeline.

## 5.3 Identifying Feature and Label Columns

The dataset columns are separated into:

- **Label column:** Represents the target variable for classification.
- **Feature columns:** Consist of numerical and categorical attributes used for prediction.

The label column is excluded from the feature set to prevent data leakage during model training.

## 5.4 Encoding Categorical Features

Categorical features are transformed into numerical values using **StringIndexer**. This process assigns a unique numerical index to each category, making the data compatible with Spark ML algorithms.

Each categorical column is indexed independently, preserving category distinctions while enabling numerical computation.

## 5.5 Assembling Feature Vectors

All numerical and encoded categorical features are combined into a single vector using **VectorAssembler**. This step is required because Spark ML models expect input features in vectorized form.

The resulting column, typically named **features**, represents the complete feature set for each record.

## 5.6 Feature Scaling

To ensure that features contribute equally to model training, **StandardScaler** is applied:

- Features are standardized to have zero mean and unit variance.
- This step prevents features with larger numeric ranges from dominating the learning process.

The scaled output is stored in a new feature vector suitable for machine learning models.

## 5.7 Final Dataset Preparation

The final dataset consists of:

- A numerical label column.
- A scaled feature vector column.

This structured format aligns with Spark's machine learning pipeline requirements.

## 5.8 Saving the Feature-Engineered Dataset

The transformed dataset is saved in `Parquet` format for efficient storage and reuse. This dataset serves as the input for subsequent model training and evaluation stages.

## 5.9 Feature Engineering

All numerical features were assembled into a single feature vector using Spark's `VectorAssembler`. Feature scaling was performed using `StandardScaler` to normalize the feature space and improve model convergence. The final dataset consisted of feature-label pairs suitable for Spark ML pipelines.

# 6 Classification Experiments Using PySpark

This section presents the implementation of multiple classification models using PySpark. The objective is to evaluate different machine learning algorithms on the feature-engineered dataset and compare their performance.

## 6.1 Importing Required Libraries

The following PySpark libraries are imported to support classification and evaluation:

- `SparkSession` for managing Spark operations.
- Classification models such as `LogisticRegression`, `DecisionTreeClassifier`, and `RandomForestClassifier`.
- `MulticlassClassificationEvaluator` for evaluating model performance.
- Spark SQL functions for data manipulation.

These libraries enable scalable model training and evaluation in a distributed environment.

## 6.2 Loading the Feature-Engineered Dataset

The feature-engineered dataset is loaded from a `Parquet` file generated in the previous stage. This dataset contains:

- A numerical label column representing the target class.
- A vectorized and scaled feature column used for classification.

## 6.3 Splitting the Dataset

The dataset is divided into training and testing subsets using a random split:

- The training set is used to fit the classification models.
- The testing set is reserved for evaluating model performance on unseen data.

This separation ensures an unbiased evaluation of model generalization.

## 6.4 Logistic Regression Model

A Logistic Regression classifier is trained as a baseline model:

- It models the probability of class membership using a linear decision boundary.
- The model is trained using the training dataset.

Predictions are generated on the test dataset for evaluation.

## 6.5 Decision Tree Classifier

A Decision Tree classifier is implemented to capture non-linear relationships:

- The model recursively splits the feature space based on information gain.
- It provides interpretability through hierarchical decision rules.

The trained model is applied to the test dataset to obtain predictions.

## 6.6 Random Forest Classifier

A Random Forest classifier is trained as an ensemble method:

- Multiple decision trees are constructed using random subsets of data and features.
- Final predictions are obtained through majority voting.

This approach improves robustness and reduces overfitting compared to a single decision tree.

## 6.7 Model Evaluation

Each classifier is evaluated using `MulticlassClassificationEvaluator`:

- Accuracy is computed to measure overall classification performance.
- Additional metrics such as precision, recall, or F1-score may also be evaluated.

These metrics allow a fair comparison between different models.

## 6.8 Comparison of Classification Models

The performance metrics of all models are compared to identify the most effective classifier for the dataset. This comparison highlights trade-offs between model complexity, interpretability, and predictive accuracy.

## 6.9 Summary

This section demonstrates the application of multiple classification algorithms using PySpark. The experimental results provide insights into the suitability of different models for large-scale classification tasks and form the basis for selecting the best-performing model for deployment.

## 6.10 Supervised Learning Models

Three supervised learning models were trained and evaluated:

1. Random Forest Classifier
2. Logistic Regression
3. Support Vector Machine (Linear SVM)

Table 1: Supervised Model Performance Comparison

Model	Learning Type	Accuracy (Relative)
Random Forest	Supervised	High
Logistic Regression	Supervised	Medium
Linear SVM	Supervised	Medium–High

The dataset was split into 80% training data and 20% testing data. Model performance was evaluated using classification accuracy.

## 7 Clustering and Anomaly Detection Using PySpark

This section focuses on applying unsupervised learning techniques to identify hidden patterns and anomalies within the dataset. Clustering and anomaly detection methods are implemented using PySpark to ensure scalability for large datasets.

### 7.1 Importing Required Libraries

The necessary PySpark libraries are imported to support clustering and anomaly detection tasks:

- `SparkSession` for initializing and managing Spark operations.
- `KMeans` for clustering analysis.
- `BisectingKMeans` as an alternative hierarchical clustering approach.
- `ClusteringEvaluator` for assessing clustering performance.
- Spark SQL functions such as `col` for data manipulation.

These libraries provide efficient tools for unsupervised learning in distributed environments.

### 7.2 Loading the Feature-Engineered Dataset

The feature-engineered dataset is loaded from a `Parquet` file produced in the previous feature engineering stage. The dataset includes a vectorized feature column that serves as input for clustering algorithms.

### 7.3 K-Means Clustering

The K-Means clustering algorithm is applied to partition the dataset into a predefined number of clusters:

- The algorithm assigns each data point to the nearest cluster centroid based on Euclidean distance.
- Cluster centroids are iteratively updated to minimize within-cluster variance.

The resulting cluster assignments provide insight into the natural groupings within the data.

### 7.4 Evaluation of K-Means Clustering

Clustering performance is evaluated using the `ClusteringEvaluator`, which computes the Silhouette score:

- The Silhouette score measures how similar a data point is to its own cluster compared to other clusters.
- Higher scores indicate better-defined and more distinct clusters.

This metric helps assess the quality of the clustering results.

### 7.5 Bisecting K-Means Clustering

Bisecting K-Means is implemented as an alternative clustering technique:

- It follows a hierarchical, divisive approach by repeatedly splitting clusters into two.
- This method can produce more balanced clusters compared to standard K-Means.

Cluster assignments generated by this method are used for comparative analysis.

### 7.6 Comparison of Clustering Methods

The results from K-Means and Bisecting K-Means are compared using their evaluation metrics. This comparison highlights differences in cluster quality and structure between the two approaches.

## 7.7 Anomaly Detection Using Clustering

Anomaly detection is performed by analyzing the distance of data points from their respective cluster centroids:

- Data points with unusually large distances are considered potential anomalies.
- These anomalies may represent rare events or abnormal behavior in the dataset.

This approach leverages clustering structure to identify outliers without labeled data.

## 7.8 Summary

This section demonstrates the application of unsupervised learning techniques for clustering and anomaly detection using PySpark. The results provide valuable insights into data structure and support the identification of abnormal patterns in large-scale datasets.

## 7.9 Unsupervised Learning and Anomaly Detection

K-Means clustering was applied to identify hidden patterns and potential anomalies. An anomaly score was computed as the distance between each data point and its assigned cluster center. Traffic flows with high anomaly scores were flagged as potentially malicious.

# 8 Association Rule Mining Using PySpark

This section applies association rule mining techniques to discover hidden relationships and co-occurrence patterns within the dataset. The implementation uses PySpark to efficiently process large volumes of transactional data.

## 8.1 Importing Required Libraries

The required PySpark libraries are imported to support frequent pattern mining:

- `SparkSession` for initializing and managing Spark operations.
- `FPGrowth` for mining frequent itemsets and generating association rules.
- Spark SQL functions for data preprocessing and column manipulation.

These tools enable scalable discovery of frequent patterns in distributed environments.

## 8.2 Loading the Dataset

The dataset is loaded using Spark into a structured DataFrame. Each record represents a transaction containing a set of items, which is the required input format for association rule mining algorithms.

## 8.3 Data Preprocessing

To prepare the data for mining:

- Transactional data is grouped or transformed into item lists.
- Each transaction is represented as an array of items.

This transformation ensures compatibility with the FP-Growth algorithm.

## 8.4 Frequent Pattern Growth (FP-Growth)

The FP-Growth algorithm is applied to identify frequent itemsets:

- FP-Growth avoids candidate generation, making it more efficient than Apriori for large datasets.
- Itemsets that satisfy a minimum support threshold are identified.

Frequent itemsets represent combinations of items that commonly occur together in transactions.

## 8.5 Generation of Association Rules

Based on the discovered frequent itemsets, association rules are generated:

- Rules are of the form  $X \Rightarrow Y$ , where  $X$  and  $Y$  are disjoint itemsets.
- Each rule is evaluated using confidence and lift metrics.

These metrics quantify the strength and usefulness of the extracted rules.

## 8.6 Evaluation of Association Rules

The generated rules are analyzed by:

- Filtering rules based on confidence and lift thresholds.
- Identifying strong and meaningful associations between items.

This step ensures that only statistically significant and practically relevant rules are retained.

## 8.7 Interpretation of Results

The resulting association rules provide insights into item co-occurrence patterns:

- They can be used for recommendation systems, market basket analysis, or behavioral insights.
- Strong rules highlight relationships that may not be obvious through simple analysis.

## 8.8 Summary

This section demonstrates the use of FP-Growth for association rule mining in PySpark. The discovered frequent itemsets and association rules reveal meaningful patterns in the data and support data-driven decision-making.

## 8.9 Association Rule Mining

Association rule mining was performed using the FP-Growth algorithm. Continuous features were discretized into binary values, and frequent itemsets were extracted. High-confidence rules revealed meaningful relationships between network features and attack behaviors.

# 9 Experiments and Evaluation

## 9.1 Classification Results

Among the supervised models, the Random Forest classifier achieved the highest accuracy, demonstrating strong performance in distinguishing benign and malicious traffic. Logistic Regression provided a simple and interpretable baseline, while Linear SVM achieved competitive results with reduced complexity.

## 9.2 Clustering and Anomaly Detection Results

K-Means clustering revealed distinct traffic patterns. Data points with large distances from cluster centers often corresponded to attack traffic, indicating the effectiveness of unsupervised learning for detecting unknown threats.

## 9.3 Association Rule Insights

Association rule mining uncovered feature combinations frequently associated with malicious activity. These rules offer interpretability and help security analysts understand the underlying causes of detected threats.

# 10 Discussion

The experimental results demonstrate that combining supervised and unsupervised techniques improves cloud security threat detection. While supervised models excel at identifying known attack patterns, unsupervised approaches provide additional protection against novel or stealthy attacks. The PySpark-based implementation ensures scalability for large cloud datasets.

# 11 Conclusion

This project demonstrates the effectiveness of AI-driven data mining techniques for cloud security threat detection. By integrating preprocessing, feature engineering, classification, clustering, and association rule mining, the proposed system provides a comprehensive framework for analyzing cloud log data. The results highlight the potential of machine learning to enhance automated intrusion detection systems.

This project demonstrates that AI-driven data mining techniques are effective for cloud security threat detection. Through systematic preprocessing, feature engineering, supervised classification, unsupervised anomaly detection, and association rule mining, the proposed system successfully identifies malicious traffic and uncovers hidden attack patterns in large-scale cloud data.

Random Forest emerged as the most effective supervised model, while K-Means clustering and FP-Growth provided valuable insights into unknown threats and attack behaviors. The use of PySpark ensures scalability, making the system suitable for real-world cloud environments. Overall, this project highlights the importance of combining multiple data mining techniques to build robust and intelligent cloud security solutions.

## 12 Future Work

Future work may explore:

- Temporal sequence modeling using LSTM or Transformer-based approaches
- Multi-class classification of specific attack types
- Real-time threat detection using streaming data
- Deployment in a production cloud environment

## 13 References

1. Sharafaldin, I., Lashkari, A. H., & Ghorbani, A. A. (2018). Toward generating a new intrusion detection dataset and intrusion traffic characterization. *Proceedings of ICISSP*.
2. Apache Spark MLlib Documentation.
3. Scikit-learn and PySpark Machine Learning Libraries.