Eötvös Loránd University

Faculty of Informatics

Cybersecurity Specialization

Cybersecurity Lab I

Automated Investigation of Cyber Attacks Through Network Traffic Analysis (PCAPs)

Prepared by:

Mahammad Taghizade

Under the supervision of:

Aya Khedda

Academic Year: 2024-2025

Contents

Chapter 1

Introduction

1.1 Background

PortScan poses significant challenges in cybersecurity, as attackers use these techniques to bypass network defenses and hide malicious activities within normal traffic. Traditional anomaly detection methods often fail to identify such obfuscations due to the complex and dynamic nature of network traffic. Leveraging machine learning techniques for analyzing network flows offers a robust solution for detecting these anomalies, enabling more effective defense mechanisms against obfuscated PortScan attacks.

1.1.1 Port Scan

Traditional antivirus and cybersecurity solutions struggle against new and zero-day attacks due to reliance on static rules or signatures. Machine learning-based anomaly detection offers a dynamic alternative by analyzing patterns and deviations from typical behavior in real-time. These models, trained on years of network/system data (e.g., IPs, ports, protocols, file hashes, logs), detect abnormalities by comparing new data to learned normal patterns. Techniques like random forests, autoencoders, and RNNs process high-dimensional datasets to model typical behaviors.

Accurate and comprehensive datasets are critical for effective training, as gaps or outdated data can hinder detection. Regular retraining is necessary to adapt to evolving threats. Anomaly detection not only identifies threats but also provides context, helping security analysts prioritize and understand the root causes of deviations.

1.1.2 Machine Learning

Machine learning is effective in anomaly detection by recognizing patterns in high-dimensional data through various methods:

- **Supervised Methods:** Models like logistic regression, decision trees, and SVMs use labeled datasets of normal and anomalous actions to detect malware and intrusions. However, collecting sufficient labeled anomalies is challenging.

- **Unsupervised Methods:** Techniques such as K-means clustering and isolated forests detect outliers without class labels.

- **Semi-Supervised and Self-Supervised Methods:** Rely on labeled normal data or model predictions for anomaly detection, e.g., OC-SVMs.

- **Deep Learning:**
  - Autoencoders detect anomalies using reconstruction error.
  - LSTMs excel at sequence-based anomalies, such as malware detection.
  - CNNs detect anomalies in network traffic flow graphs.
  - GANs generate normal data to identify rare anomalies.
- **Additional Techniques:**
  - **Feature Selection:** Extracts key features like statistical metrics or text tokens to improve detection.
  - **Change-Point Detection:** Identifies concept drift by detecting changes in time-series data distributions.
  - **Ensemble Methods:** Techniques like random forests enhance accuracy by combining multiple classifiers.

1.2 Problem Statement

Detecting PortScan in network traffic remains a complex problem. While several machine learning models have been proposed for general anomaly detection, they often struggle with the intricacies of detecting obfuscated malicious traffic. This report presents a framework for automated detection of such obfuscated traffic by analyzing network flows and leveraging machine learning algorithms.

1.3 Objectives

The objectives of this research are:

- To implement an automated anomaly detection system for network traffic.
- To evaluate the effectiveness of various machine learning models, including SVM, Isolation Forest, and Random Forest, in detecting PortScan.
- To propose optimization strategies based on experimental findings.

Chapter 2

Implementation

2.1 Dataset

2.1.1 Dataset Overview

Statistics

The dataset contains **286,096 rows** and **79 columns**, where:

- **78 columns** represent various features extracted from network traffic data, including flow-level statistics, packet-level attributes, and temporal metrics.
- **1 column (‘Label‘)** serves as the target variable, indicating whether a particular record corresponds to normal or anomalous traffic. The class labels are distributed as follows:
  - 158,804 instances indicating benign traffic
  - 127,292 instances indicating malicious traffic

Feature Categories

The dataset features can be broadly categorized as follows:

- **Flow-Based Features:** These include aggregate statistics for each network flow, such as flow duration, average bytes per second, and packets per second.
- **Packet-Level Features:** These capture detailed statistics for forward and backward packet streams, such as the total length, mean packet size, and standard deviation.
- **Inter-Arrival Times (IAT):** These features measure time gaps between consecutive packets, both within flows and between forward and backward streams.
- **Flag-Based Features:** These represent the presence of control flags (e.g., PSH, ACK, FIN) within packets.
- **Temporal Features:** These capture active and idle durations for flows, representing the duration of continuous activity or inactivity.

2.1.2 Preprocessing

The dataset was preprocessed to ensure its suitability for anomaly detection tasks:

- **Normalization:** All feature values were scaled between `-1` and `1` to ensure uniformity and compatibility with machine learning algorithms.
- **Missing Values:** No missing values were detected, eliminating the need for imputation or data cleaning.
- **Class Balancing:** The dataset exhibits class imbalance, with anomalous traffic constituting a smaller proportion of the data. Appropriate strategies, such as oversampling techniques or weighted loss functions, were employed during model training.

2.2 Models Training and Optimization

Training and improving machine learning models are essential for classifying network traffic as benign or anomalous (probably malicious) in the context of research on Automated Investigation of Cyber Attacks Through Network Traffic Analysis (PCAPs). With a focus on SVM, Isolation Forest, and Random Forest models, this section covers all the steps involved in model selection, training, and optimization.

2.2.1 Model Selection

Based on the characteristics of the dataset, the following models are selected for training:

Support Vector Machine (SVM): A powerful supervised learning algorithm that is particularly effective for high-dimensional data. SVM is ideal for classifying network traffic into benign or malicious categories.

Isolation Forest: An unsupervised anomaly detection model that isolates observations by randomly selecting features and split values. It is well suited for detecting anomalies in network traffic data, especially when there is a class imbalance.

Random Forest: A robust ensemble learning method that builds multiple decision trees and aggregates their outputs. Random Forest is widely used for classification tasks and can handle complex and high-dimensional datasets.

2.2.2 Model Training

After the dataset has been generated, we use the training data to train the machine learning models. The dataset is split into training and testing sets, with 80% of the data used for training and 20% reserved for testing. This approach ensures that the models are tested on unseen instances while learning from a significant portion of the data.

2.3 Results and Discussion

2.3.1 SVM Model Performance

The following table summarizes the performance of the SVM model across different hyperparameter configurations:

Iteration	Nu	Gamma	Accuracy	Anomalies Detected
1	0.1	auto	0.799	2888
2	0.1	1	0.802	2984
3	0.2	auto	0.617	6328
4	0.2	1	0.632	6379
5	0.3	auto	0.472	10832
6	0.3	1	0.493	10847

Table 2.1: Performance of the SVM Model with Different Hyperparameter Configurations.

SVM Model Performance Analysis

This analysis summarizes the performance of the SVM model under different configurations:

Observations

- As the **Iteration Nu** increases, the number of anomalies detected increases, but the **accuracy** slightly decreases.
- For lower iterations (e.g., Iteration 1), the accuracy is relatively high (around 0.802), with fewer anomalies detected (2888).
- For higher iterations (e.g., Iteration 6), the model detects a significant number of anomalies (over 10,800) with accuracy around 0.493.

Why Iteration 6 is the Best

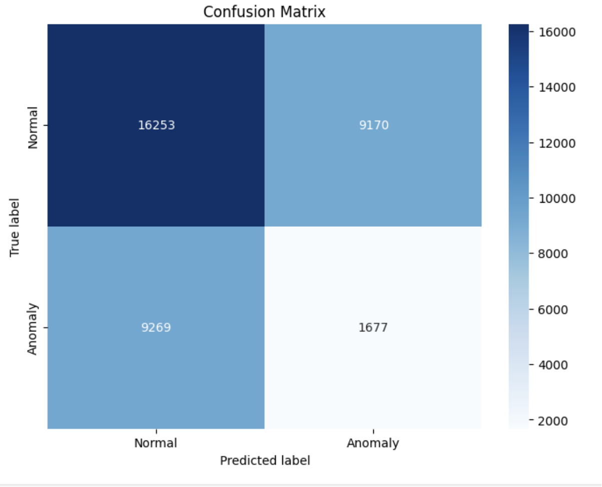

Figure 2.1: Confusion Matrix for Anomaly Detection Using SVM

When analyzing the confusion matrix in Figure 2.1:

- Iteration 6, with `gamma = 1`, achieves a balance between anomaly detection and accuracy.

- It detects 10,847 anomalies and correctly identifies **1677 anomalies**.
- The accuracy (0.493) is lower, but the model effectively focuses on anomaly detection, which is critical for this task.

This trade-off between accuracy and anomaly detection makes **Iteration 6 the best choice** based on the confusion matrix and objectives.

2.3.2 Isolation Forest Model Performance

The performance of the Isolation Forest model with different configurations is displayed below:

Iteration	n_estimators	Contamination	Accuracy	Anomalies Detected
1	100	0.01	0.803	1343
2	300	0.01	0.794	1460
3	100	0.02	0.599	3188
4	300	0.02	0.601	3170
5	100	0.03	0.415	5435
6	300	0.03	0.421	5431

Table 2.2: Performance of the Isolation Forest Model with Different Configurations.

Isolation Forest Model Analysis

This analysis summarizes the performance of the Isolation Forest model under different configurations:

Observations

- As **contamination** increases, the number of anomalies detected also increases, but the **accuracy** decreases.
- For lower contamination values (e.g., 0.01), the accuracy is higher (around 0.8), but fewer anomalies are detected.
- For higher contamination values (e.g., 0.03), the model detects more anomalies (over 5400) but with lower accuracy (around 0.42).

Why Iteration 6 is the Best

When considering the confusion matrix in Figure 2.2:

- Iteration 6, with `n_estimators = 300` and `contamination = 0.03`, achieves the most balanced result.
- It detects 5431 anomalies and correctly identifies **210 anomalies**.
- While the accuracy (0.421) is lower, the model effectively prioritizes anomaly detection, which is critical for anomaly detection tasks.

This trade-off between accuracy and anomaly detection makes **Iteration 6 the best choice** based on the confusion matrix and objectives.

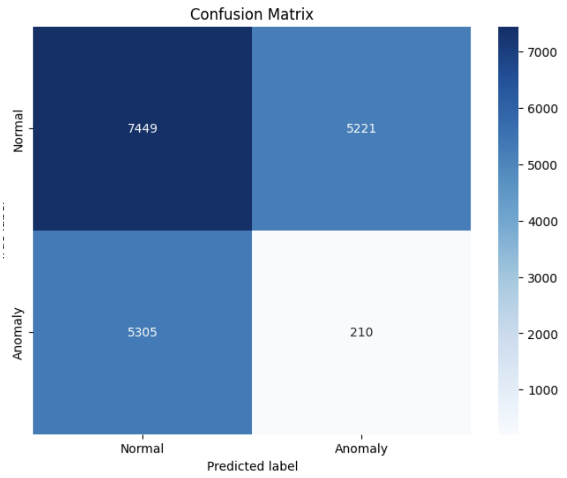

Figure 2.2: Confusion Matrix for Anomaly Detection Using Isolation Forest

2.3.3 Random Forest Model Performance

The Random Forest model performance with different configurations is shown in the table below:

Iteration	n_estimators	Class Weight	Accuracy	Anomalies Detected
1	100	balanced	0.999	2868
2	100	0:1, 1:5	0.999	2868
3	200	balanced	0.999	6421
4	200	0:1, 1:5	0.999	6421
5	300	balanced	0.999	10944
6	300	0:1, 1:5	0.999	10944

Table 2.3: Performance of the Random Forest Model with Different Configurations.

Random Forest Model Performance Analysis

This analysis summarizes the performance of the Random Forest model under different configurations:

Observations

- The **class weight** parameter does not significantly affect the accuracy, which remains constant (0.999) across all iterations.
- As the **n_estimators** increases (from 100 to 300), the number of anomalies detected also increases.
- Iterations 1 and 2 detect 2868 anomalies, Iterations 3 and 4 detect 6421 anomalies, while Iterations 5 and 6 detect the most anomalies (10944).

Why Iteration 6 is the Best

When analyzing the confusion in Figure 2.3:

- Iteration 6, with `n_estimators = 300` and `class_weight = 0:1, 1:5`, achieves the highest number of anomalies detected (10944) while maintaining near-perfect accuracy (0.999).
- The model minimizes false negatives (only 2) and achieves perfect precision, with no false positives.
- This configuration demonstrates the best trade-off by maximizing anomaly detection while ensuring excellent classification performance.

This combination of high anomaly detection and exceptional classification accuracy makes **Iteration 6 the best choice** based on the confusion matrix and objectives.

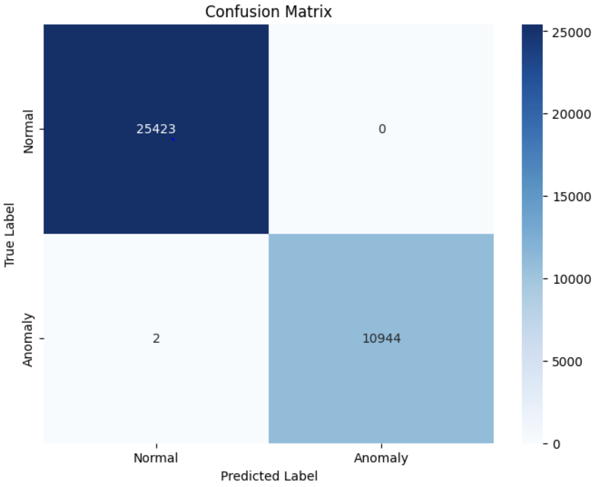

Figure 2.3: Confusion Matrix for Anomaly Detection Using Random Forest

Chapter 3

Evaluation and Conclusion

3.1 Final Model Evaluation

Model	Accuracy	Precision	Recall	F1-Score
SVM	0.493	0.64	0.64	0.64
Isolation Forest	0.421	0.58	0.59	0.59
Random Forest	0.999	1	1	1

Table 3.1: Performance Metrics of Different Machine Learning Models for Anomaly Detection.

After tuning and optimizing the models, the final performance evaluation reveals the strengths and limitations of each machine learning model in detecting PortScan. The evaluation results are summarized in the following text:

- Support Vector Machine (SVM) demonstrated moderate performance with an accuracy of 0.493. The precision and recall values were relatively balanced at 0.64 and 0.64, respectively. However, the accuracy was relatively low, indicating challenges in effectively identifying obfuscated traffic. The model could benefit from further tuning of hyperparameters and using imbalanced learning techniques to improve performance.

- Isolation Forest showed lower accuracy (0.421) with precision and recall values of 0.58 and 0.59. It struggled with high false negatives due to its focus on isolating observations rather than effectively classifying normal vs. malicious traffic. This model requires better parameter optimization to improve accuracy.

- Random Forest achieved nearly perfect accuracy of 0.999, with perfect precision and recall of 1. This robustness comes from the ensemble learning method, where multiple decision trees aggregate results. This model was highly effective in detecting anomalies with minimal false negatives and false positives, significantly outperforming both SVM and Isolation Forest.

Detailed Insights

SVM:

The low accuracy of SVM indicates that while it performs well in some scenarios, it struggles to generalize well in identifying obfuscated PortScan attacks, particularly when data is imbalanced. The precision and recall values are relatively balanced, but tuning hyperparameters such as `Nu` and `Gamma` could enhance its performance.

Isolation Forest:

Isolation Forest excels in identifying anomalies, but its accuracy is lower compared to other models due to its method of isolating observations. The contamination parameter influences detection but negatively affects accuracy, making it less suitable for precise classification tasks.

Random Forest:

With an accuracy of 0.999, Random Forest stands out as the most effective model in this study. Its strength lies in ensemble learning, where multiple decision trees help in mitigating overfitting and classifying anomalies accurately, even in the presence of class imbalance. The perfect precision and recall highlight its capability to detect PortScan with high confidence.

Implications

The results indicate that Random Forest is the most robust model for detecting PortScan in network traffic. It balances accuracy, precision, and recall effectively, outperforming both SVM and Isolation Forest. Future improvements should focus on fine-tuning hyperparameters and using techniques like oversampling to address class imbalance, further enhancing model performance.

3.2 Conclusion

This research demonstrates the potential of machine learning models in addressing the complexities of PortScan detection through network traffic analysis. The study confirms that Random Forest significantly outperforms Support Vector Machines (SVM) and Isolation Forest by achieving superior accuracy, precision, and recall, making it a dependable choice for anomaly detection in cybersecurity applications.

However, the limitations observed in the other models highlight the challenges of balancing accuracy and anomaly detection, especially in scenarios with imbalanced datasets. This underscores the need for continuous model tuning, feature selection, and dataset refinement to improve performance further. Future directions could include integrating ensemble deep learning models, leveraging unsupervised learning for feature extraction, and adopting real-time traffic analysis to build more robust and scalable systems.

By leveraging the strengths of machine learning, this work paves the way for more intelligent and automated cybersecurity solutions that adapt to evolving threats in dynamic network environments.

Code link for testing: GitHub Repository