Anomaly Detection for Cybersecurity Data Streams

Version 1.0

Fathia Alfajr

January 5, 2025

Abstract

Cybersecurity remains a critical concern as organizations face increasingly sophisticated threats. This project focuses on developing an anomaly detection system for cybersecurity data streams using Machine Learning. By using PyTorch, the system automates the detection of anomalies in both labeled and unlabeled datasets, offering customizable metrics such as ROC curves and precision to ensure transparency and reliability. A user-friendly dashboard, built with Streamlit, enables practitioners to analyze datasets, visualize results, and interact with models efficiently. The system's modular design, encompassing data preprocessing, model training, and evaluation, addresses limitations of traditional tools like Wireshark and Snort by reducing manual effort and enhancing accuracy. The results demonstrate the system's potential to empower cybersecurity practitioners and students with a robust and scalable tool for anomaly detection. Future work aims to extend the system's capabilities to support real-time data streams, advanced visualization techniques, and cloud deployment for broader applicability.

Contents

1 Introduction

With only 4% of organizations confident that their security measures adequately protect against cyberattacks, the need for advanced anomaly detection in cybersecurity has never been greater. Traditional security systems often fail to detect novel threats, leaving organizations exposed to emerging attacks. Anomaly detection offers a solution by identifying deviations from normal behavior, providing early warnings, and reducing false positives through machine learning techniques. This makes it a vital tool to strengthen cybersecurity defenses.

This project uses Machine Learning to address the challenges of anomaly detection in cybersecurity data streams. By training models on labeled or unlabeled datasets, the system automates the identification of anomalies with adjustable metrics and interactive analysis options. Unlike manual benchmarking, which is prone to errors, this project uses AI, specifically PyTorch, to create reliable benchmarks and provide transparent performance metrics, including true positives, false positives, and other evaluation measures. Users can save models with timestamps and select them during inference for customized analysis.

The potential impact of this project lies in its ability to empower cybersecurity practitioners and students with an efficient and accessible tool to analyze large datasets. By reducing manual efforts and improving reliability, the system enhances decision-making processes and contributes to the broader goal of fortifying defenses against evolving cyber threats.

2 Background

This project was initiated from the developer's personal motivation to explore Machine Learning. After working on a prior project involving API integration with artificial intelligence, Fathia Alfajr, the author and developer of this documentation, decided to extend her knowledge by exploring Machine Learning applications in cybersecurity.

With her computer engineering background and a keen interest in cybersecurity, she identified an opportunity to bridge a gap in network data analysis. During her studies, she extensively used Wireshark, a popular tool for network capture and packet inspection. However, she identified a gap: analyzing large data capture files manually using Wireshark tools is both time consuming and error-prone, particularly for students learning the basics of cybersecurity.

Using Machine Learning, this project aims to simplify the process of identifying anomalies in cybersecurity-related data. It bridges the gap between traditional tools and automated analysis, offering a reliable foundation for both practitioners and students to explore advanced anomaly detection techniques.

3 Problem Statement

- **The Growing Need for Anomaly Detection:**
  - Cyber threats are becoming increasingly sophisticated, requiring advanced methods to detect novel attacks and deviations from normal behavior.
  - Only 4% of organizations are confident their existing security measures adequately protect against cyberattacks, highlighting a significant gap in current defenses.
- **Limitations of Existing Tools:**
  - **Manual Effort:** Tools like Wireshark require significant manual effort to analyze large data capture files, making them time consuming and error prone.
  - **Rule-Based Constraints:** Systems such as Snort rely on predefined rules, which are less effective in identifying unknown threats and lack flexibility for file-based analysis.
  - **Transparency Issues:** Current tools often fail to provide clear performance metrics, such as false positives, false negatives, and overall reliability.
- **The Opportunity:**
  - A Machine Learning-based system can automate the process of anomaly detection, reducing manual efforts and enhancing accuracy.
  - By integrating customizable benchmarking, advanced metrics (e.g., ROC curves and confusion matrices), and interactive analysis options, this project addresses critical gaps in existing solutions.

4 Methodology

4.1 Overview of the Approach

This project uses Machine Learning to detect anomalies in cybersecurity-related datasets. The methodology consists of three main phases: data preparation, model training, and inference. The system allows users to train models on labeled or unlabeled datasets and analyze cybersecurity data by selecting appropriate metrics and benchmarks.

4.2 Data Preparation

- **Data Collection:** Datasets are sourced from publicly available cybersecurity repositories.
- **Preprocessing:**
  - Data is normalized to ensure consistency.

  - Missing values are handled using statistical imputation techniques.
  - All features from the dataset are extracted and made available for analysis, allowing users to select specific features for visualization, such as x-axis and y-axis comparisons in the dashboard.

4.3 Model Development

- **Model Architecture:** The system uses PyTorch to implement supervised and unsupervised learning models, such as:
  - *Unsupervised Models:* Autoencoders are implemented for anomaly detection in unlabeled datasets, leveraging reconstruction errors to identify anomalies.
  - *Supervised Mode:* The same autoencoder architecture can be used in supervised settings where labeled datasets are provided. Labels are used to separate normal and anomalous samples for evaluation purposes.
- **Training Process:**
  - Models are trained using cybersecurity datasets.
  - Default hyperparameters, such as a learning rate of 0.001 and a batch size of 32, were selected based on standard practices to ensure stable training and convergence.
  - Training metrics include accuracy, precision, recall, and F1-score.

4.4 Inference and Analysis

- During inference, users can input file paths to the system for anomaly detection.
- Users can select from saved models (stored with timestamps) and specify analysis metrics, such as ROC curves, confusion matrices, or anomaly scores.
- The system provides an interactive visualization feature that allows users to choose which variables to display on the x-axis and the y-axis for detailed comparisons.
- Outputs include true positives, false positives, and anomaly scores, presented both numerically and visually for ease of interpretation.

4.5 Tools and Frameworks

- **Programming Language:** Python
- **Libraries and Frameworks:**
  - PyTorch: For implementing and training machine learning models.
  - Pandas and NumPy: For data manipulation and preprocessing.
  - Scikit-learn: For supervised and unsupervised learning algorithms.
  - Matplotlib and Seaborn: For visualization of metrics and results.
- **Dashboard Development:** Streamlit is used to create an interactive user interface for model selection, data input, and analysis visualization.

5 System Architecture

The system architecture is illustrated in the figure below:

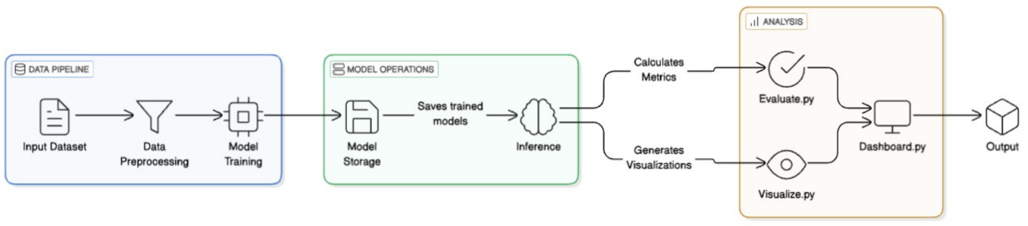
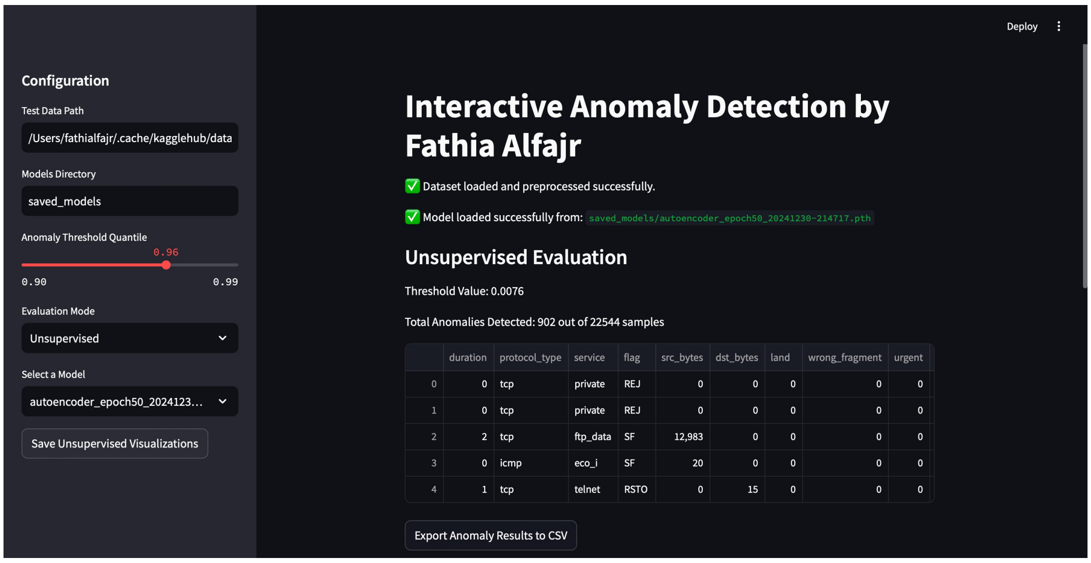


Figure 1: System Architecture for the Anomaly Detection Project

The system architecture for the anomaly detection project is designed in three distinct stages: **Data Pipeline**, **Model Operations**, and **Analysis**. In the **Data Pipeline**, input datasets undergo preprocessing to normalize, clean, and extract features before being used for model training. The trained models are then stored in the **Model Operations** stage, where they are saved with timestamps for future use. During the inference phase, the stored models are utilized to analyze new datasets. In the **Analysis** stage, inference results are processed using `Evaluate.py` to calculate metrics like ROC and precision, while `Visualize.py` generates graphical representations of the data. Finally, `Dashboard.py` consolidates these outputs, offering an interactive user interface to display metrics and visualizations, ensuring the system is transparent, user-friendly, and effective in anomaly detection.

6 Results and Evaluation

The interactive dashboard, shown in Figure 2, provides users with a comprehensive interface to analyze anomalies in cybersecurity datasets. It includes configuration options for specifying test data paths, selecting saved models, and adjusting anomaly thresholds. The evaluation results, including total anomalies detected and detailed breakdowns, are displayed in tabular format, with an option to export the results as a CSV file.


Deploy
Save Unsupervised Visualizations
Export Anomaly Results to CSV

Configuration

Test Data Path: /Users/fathialfajr/.cache/kagglehub/data

Models Directory: saved_models

Anomaly Threshold Quantile: 0.96 (0.90 – 0.99)

Evaluation Mode: Unsupervised

Select a Model: autoencoder_epoch50_2024123…

Interactive Anomaly Detection by Fathia Alfajr

✅ Dataset loaded and preprocessed successfully.

✅ Model loaded successfully from: `saved_models/autoencoder_epoch50_20241230-214717.pth`

Unsupervised Evaluation

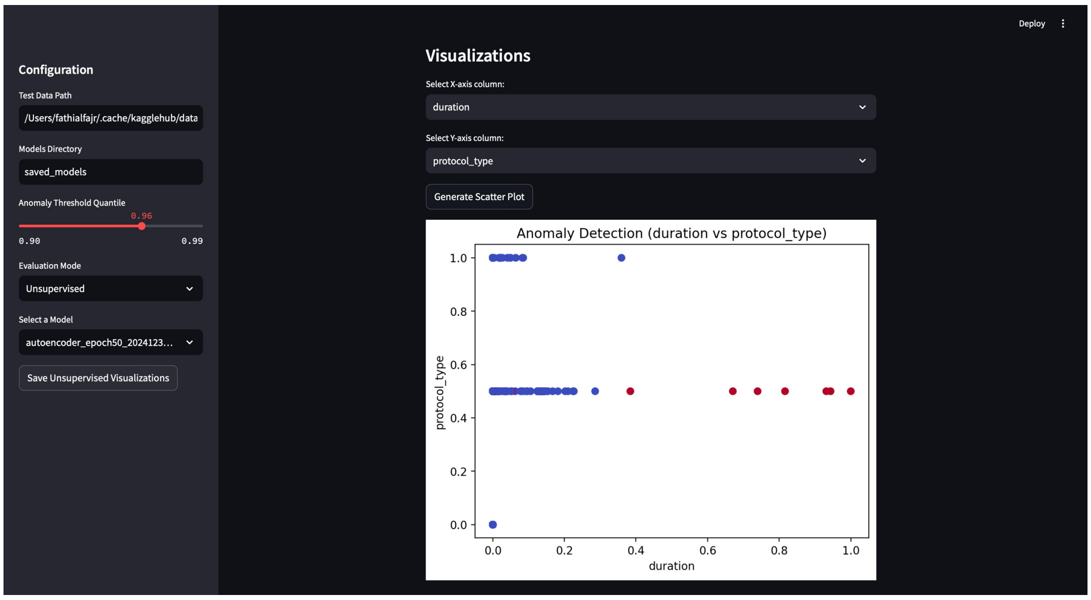
Threshold Value: 0.0076

Total Anomalies Detected: 902 out of 22544 samples

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent
0	0	tcp	private	REJ	0	0	0	0	0
1	0	tcp	private	REJ	0	0	0	0	0
2	2	tcp	ftp_data	SF	12,983	0	0	0	0
3	0	icmp	eco_i	SF	20	0	0	0	0
4	1	tcp	telnet	RSTO	0	15	0	0	0

Figure 2: Configuration and Evaluation Results Panel

Users can also visualize anomalies interactively by selecting x-axis and y-axis variables for scatter plots, as shown in Figure 3. This feature enhances usability by providing intuitive insights into the anomaly distribution.



Configuration

Test Data Path: /Users/fathialfajr/.cache/kagglehub/data

Models Directory: saved_models

Anomaly Threshold Quantile: 0.96 (0.90 – 0.99)

Evaluation Mode: Unsupervised

Select a Model: autoencoder_epoch50_2024123…

Visualizations

Select X-axis column: duration

Select Y-axis column: protocol_type

Figure 3: Visualization Panel: Scatter Plot of Anomalies

The dashboard improves transparency and usability, making the system accessible for both technical and non-technical users. Its modular design supports efficient analysis and visualization, facilitating better decision-making in anomaly detection.

7 Challenges and Limitations

7.1 Transitioning from Manual to AI-Powered Benchmarking

Initially, the system relied on manual benchmarking and tuning. To create a more adaptive and self-improving anomaly detection system, the methods were transitioned to an AI-powered benchmark, following best practices in Machine Learning. This process required significant restructuring and optimization of the codebase.

7.2 Framework Compatibility Issues

The initial implementation used TensorFlow, but compatibility issues with the Python version led to difficulties in development. This challenge prompted a switch to PyTorch, requiring additional effort to learn and implement its framework. The training and inference processes were modularized into separate scripts (`train.py` and `inference.py`) for better maintainability.

7.3 Enhancing Usability with Metrics

To improve the project's transparency and reliability, metrics such as ROC curves, precision, false positives, and true positives were introduced. However, maintaining a clear and understandable code flow was challenging. To address this, the original `inference.py` file was further broken down into separate files: `evaluate.py`, for calculating and displaying performance metrics, and `visualize.py`, for generating visualizations. Additionally, utility scripts were created to organize shared logic, and modular files (`download.py`, `train.py`, `evaluate.py`, `visualize.py`) were developed for specific tasks.

7.4 Dashboard Development

Before the `dashboard.py` file was created, manual testing of individual scripts like `evaluate.py` and `visualize.py` was tedious, especially due to the dual modes (supervised and unsupervised). The introduction of the dashboard significantly improved usability by allowing users to test the system holistically and intuitively, streamlining both development and debugging.

8 Future Work

While the current system achieves its goal of detecting anomalies in cybersecurity datasets, several key areas can be improved to enhance its functionality and impact:

- **Real-Time Anomaly Detection:** Extend the system to support real-time data streams, enabling immediate detection of anomalies in network traffic and broader applicability in cybersecurity.
- **Enhanced Visualization:** Improve the visualization capabilities by incorporating time-series analysis, heatmaps, and interactive dashboards, providing deeper insights into anomaly patterns.

- **Model Explainability:** Implement interpretability techniques, such as SHAP or LIME, to provide users with insights into the model's decision-making process, increasing trust and transparency.
- **Cloud Deployment:** Deploy the system on a cloud platform to handle larger datasets, allow multi-user access, and support API integration for real-world applications.

9 References

- Sommer, R., & Paxson, V. (2010). *Outside the closed world: On using machine learning for network intrusion detection.* 2010 IEEE Symposium on Security and Privacy, 305–316. https://doi.org/10.1109/SP.2010.25
- Ahmed, M., Mahmood, A. N., & Hu, J. (2016). *A survey of network anomaly detection techniques.* Journal of Network and Computer Applications, *60*, 19–31. https://doi.org/10.1016/j.jnca.2015.11.016
- Davis, J., & Goadrich, M. (n.d.). *The Relationship Between Precision-Recall and ROC Curves.* Retrieved from https://ftp.cs.wisc.edu/machine-learning/shavlik-group/davis.icml06.pdf
- Shi, Y., Liu, Y., Tong, H., He, J., Yan, G., & Cao, N. (2020). *Visual Analytics of Anomalous User Behaviors: A Survey.* IEEE Transactions on Big Data, 1–1. https://doi.org/10.1109/tbdata.2020.2964169
- Peremore, K. (2023). *Why healthcare organizations should maintain both paper and digital records.* Paubox.com. https://doi.org/1098040/CLEAN-6-1-theme_child
- Anomaly Detection in Machine Learning: Examples, Applications & Use Cases — IBM. (2024, July 18). *Www.ibm.com.* Retrieved from https://www.ibm.com/think/topics/machine-learning-for-anomaly-detection
- Gupta, A. (2019, May 27). *Machine Learning for Anomaly Detection.* GeeksforGeeks. Retrieved from https://www.geeksforgeeks.org/machine-learning-for-anomaly-