

Real-Time IoT Malware Detection Pipeline

NYU Big Data Final Project

Arsh Panesar (ap9332)
Charlotte Xiu (yx2867)
Namit Surana (ns6518)
Yize Liu (yl7401)

December 17, 2025

Abstract

This report details the design and implementation of a real-time streaming pipeline for detecting malware in Internet of Things (IoT) network traffic. Leveraging the CTU-IoT-Malware-Capture dataset, the system utilizes Apache Kafka for data ingestion, Apache Spark for distributed processing and machine learning, and MongoDB for result storage. The pipeline successfully processes network connection logs, applies feature engineering transformations, and employs a Gradient Boosting Tree (GBT) classifier to identify malicious activities with high precision. The system is fully containerized using Docker, ensuring reproducibility and scalability.

1 Introduction

The proliferation of Internet of Things (IoT) devices has exponentially expanded the attack surface for cyber threats, necessitating robust and real-time security mechanisms. Traditional batch processing methods are often insufficient for detecting active attacks in the high-velocity data streams generated by modern networks. This project aims to address this challenge by building a scalable, real-time malware detection pipeline using Big Data technologies.

1.1 Background and Motivation

We utilize the CTU-IoT-Malware-Capture dataset, which contains a variety of data captured from network traffic. Crucially, this dataset clearly marks malicious packets associated with various attack vectors, including Distributed Denial of Service (DDoS), Command and Control (C&C), Heartbeat, and Port Scans. This labeled data allows for reliable analysis and the construction of scalable detection heuristics.

1.2 Project Significance

The primary goal of this project is to develop a big-data-oriented malware traffic analysis pipeline capable of processing tens of millions of network flow records. By performing

statistical operations—such as identifying port ranges frequently targeted by malicious packets and analyzing the size and timing distributions of these packets—we aim to create a comprehensive “heuristic image” of malicious traffic.

This approach is significant because such datasets are continuously generated by operational computer systems, particularly in large-scale environments like data centers. A scalable solution allows for the creation of tailored heuristic images for specific systems, empowering network administrators and security professionals to analyze attack patterns effectively and take timely, informed actions.

2 System Architecture

The project infrastructure is built upon a microservices architecture orchestrated via Docker Compose. The key components include:

- **Data Source:** The system simulates real-time network traffic using the CTU-IoT-Malware-Capture dataset.
- **Message Broker (Kafka):** A Python-based producer streams connection logs to a Kafka topic `network-traffic`, decoupling data generation from processing.
- **Stream Processing (Spark):** Apache Spark Structured Streaming consumes messages from Kafka, parses JSON data, and applies machine learning models in real-time.
- **Storage (MongoDB):** Predicted results, including probability scores and attack labels, are stored in a MongoDB database for persistence.
- **Visualization:** A Streamlit dashboard and Jupyter notebooks provide real-time monitoring and post-hoc analysis of the detected threats.

3 Big Data Operations

The core of this project relies on the seamless integration of big data frameworks to handle the velocity and volume of IoT network traffic.

3.1 Apache Kafka for Data Ingestion

Apache Kafka serves as the backbone for data ingestion, providing a fault-tolerant and high-throughput message bus.

- **Producer Implementation:** A custom Python producer reads historical CSV logs and replays them as a live stream. It injects a simulated timestamp (`timestamp_simulated`) into each record to mimic real-time event generation.
- **Serialization:** Data is serialized into JSON format before being pushed to the `network-traffic` topic, ensuring a schema-agnostic transport layer that decouples the data source from the consumer.

3.2 Apache Spark Structured Streaming

Apache Spark is utilized for its distributed computing capabilities, enabling low-latency processing of the data stream.

- **Micro-Batch Processing:** The system employs Spark Structured Streaming, which processes data in small micro-batches. This allows for high throughput while maintaining near real-time latency.
- **Schema Enforcement:** To ensure data quality, a strict schema is applied to the incoming JSON stream. Malformed records are handled gracefully to prevent pipeline failures.
- **ML Pipeline Integration:** The project leverages Spark MLlib's `PipelineModel`. The exact feature engineering steps (indexing, encoding, scaling) fitted during the training phase are applied to the streaming data. This ensures consistency between training and inference environments and eliminates training-serving skew.
- **Checkpointing:** Spark's checkpointing mechanism is configured to save the state of the stream to a reliable storage. This ensures fault tolerance, allowing the application to recover from failures and continue processing from the last committed offset.

3.3 Data Persistence Strategy

Processed results are stored in MongoDB, a NoSQL database chosen for its flexibility and write performance.

- **Batch Insertion:** To optimize write throughput, the system aggregates predictions within each Spark micro-batch. These batches are converted to Pandas DataFrames and bulk-inserted into MongoDB using the `pymongo` driver, bypassing the overhead of individual record insertions.

4 Methodology

4.1 Data Ingestion

The raw dataset consists of labeled network connection logs (CSV format). A custom producer script reads these files and publishes them to Kafka, simulating a live network feed. Over 156,000 records were successfully streamed during the testing phase.

4.2 Feature Engineering

Raw network data requires significant preprocessing to be suitable for machine learning. The feature engineering pipeline, implemented in Spark MLlib, includes:

- **String Indexing:** Converting categorical variables like `proto` (protocol) and `conn_state` (connection state) into numerical indices.
- **One-Hot Encoding:** Transforming indices into binary vectors to prevent ordinal relationships in categorical data.

- **Vector Assembly:** Combining numerical features (e.g., `duration`, `orig_bytes`, `resp_bytes`) with encoded categorical vectors into a single feature vector.
- **Standard Scaling:** Normalizing features to ensure uniform contribution to the model.

4.3 Model Training

Two primary algorithms were evaluated: Random Forest and Gradient Boosting Trees (GBT). The models were trained on a batch-loaded subset of the data. The GBT model was selected as the final candidate for the real-time pipeline due to its superior performance in distinguishing between benign and malicious traffic.

5 Results

The pipeline demonstrated robust performance in processing high-velocity data streams. Key achievements include:

- **End-to-End Latency:** The system processes records and generates predictions with minimal latency suitable for near real-time monitoring.
- **Detection Capability:** The model successfully identifies various attack types present in the dataset.
- **Scalability:** The use of Spark and Kafka allows the system to scale horizontally to handle increased data loads.

6 Conclusion

This project successfully demonstrates the application of Big Data technologies for cybersecurity. By integrating Kafka, Spark, and Machine Learning, we created a scalable solution capable of detecting IoT malware in real-time. Future work could involve integrating more complex deep learning models and expanding the dashboard capabilities for granular threat analysis.