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

Contents

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), and Port Scans, etc. 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 Design & Architecture

The project infrastructure is built upon a microservices architecture orchestrated via Docker Compose. This section details the core components and the big data operations that enable the seamless handling of high-velocity IoT network traffic.

2.1 High-Level Architecture

The system comprises the following key components:

- **Data Source**: CTU IoT Malware captures (12 files, 3.2GB total). Dev split: 23MB (156K rows); full test: 1.3GB (10.45M rows).
- **Message Broker (Kafka)**: Decouples data generation from processing using a publish-subscribe model.
- **Stream Processing (Spark)**: Consumes messages, parses data, and applies ML models in real-time.
- **Storage (MongoDB)**: Persists predicted results for historical analysis and dashboarding.
- **Visualization**: Provides real-time monitoring via Streamlit and post-hoc analysis via Jupyter notebooks.

2.2 Data Ingestion Layer (Apache Kafka)

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.

2.3 Stream Processing Layer (Apache Spark)

Apache Spark Structured Streaming is utilized for its distributed computing capabilities, enabling low-latency processing.

- **Micro-Batch Processing**: The system processes data in small micro-batches, balancing high throughput with near real-time latency.
- **Schema Enforcement**: A strict schema is applied to the incoming JSON stream to ensure data quality and handle malformed records gracefully.
- **ML Pipeline Integration**: The project leverages Spark MLlib's `PipelineModel`. Feature engineering steps fitted during training are applied directly to the streaming data, ensuring consistency and eliminating training-serving skew.
- **Checkpointing**: Spark's checkpointing mechanism saves the stream state to reliable storage, ensuring fault tolerance and recovery from failures.

2.4 Storage Layer (MongoDB)

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

- **Batch Insertion**: To optimize throughput, predictions are aggregated within each Spark micro-batch and bulk-inserted using the `pymongo` driver, bypassing individual record insertion overhead.

3 Implementation Methodology

This section outlines the data science workflow, from preprocessing raw logs to training the machine learning models.

3.1 Data Preprocessing & 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 raw numerical features (e.g., `duration`, `orig_bytes`, `resp_bytes`) and derived features (e.g., `total_bytes`, `bytes_per_sec`, `hour_of_day`) with encoded categorical vectors into a single feature vector.
- **Standard Scaling**: Normalizing features to ensure uniform contribution to the model.

3.2 Model Development

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.

4 Results & Discussion

The pipeline demonstrated robust performance in processing high-velocity data streams, successfully identifying various attack types present in the dataset.

4.1 System Performance

- **End-to-End Latency**: The system processes records and generates predictions with minimal latency, suitable for near real-time monitoring.
- **Scalability**: The decoupled nature of Spark and Kafka allows the system to scale horizontally to handle increased data loads.

4.2 Real-Time Dashboard

To provide actionable insights, a real-time dashboard was developed using Streamlit, connecting directly to the MongoDB storage layer. Figure 1 illustrates the dashboard interface during a live capture session.

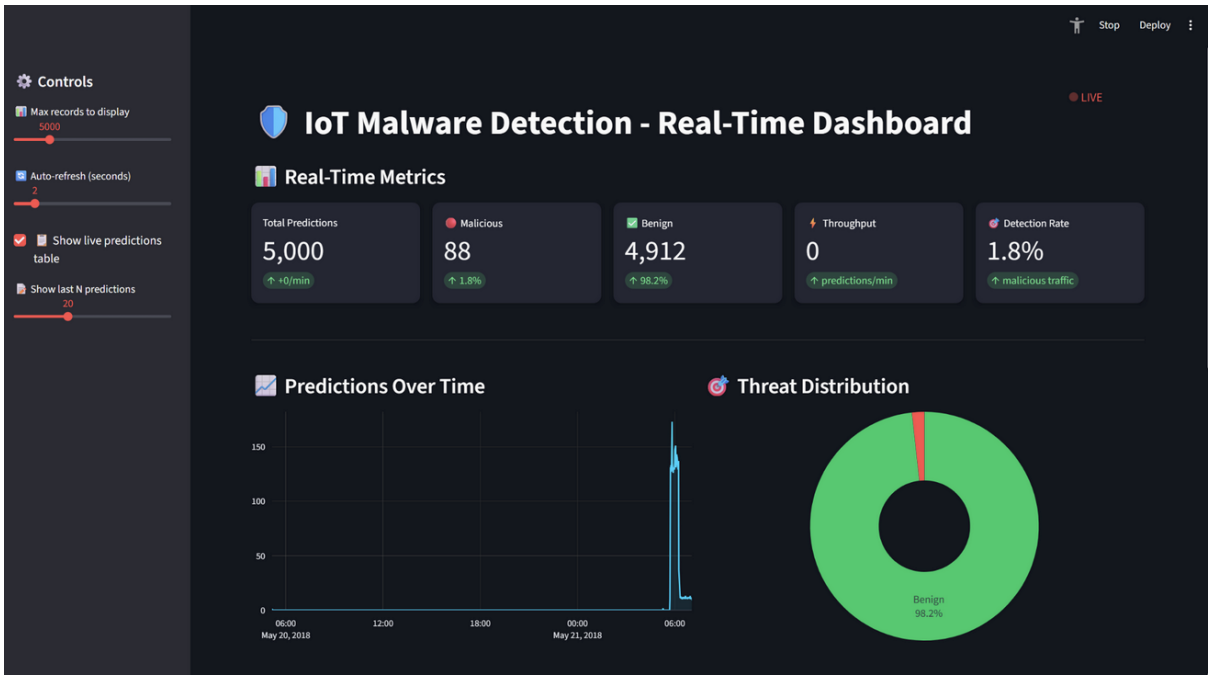

Figure 1: Real-Time Streamlit Dashboard showing live traffic metrics and attack distribution.

This interface is designed to empower network administrators and security analysts by converting high-velocity data into actionable intelligence. Its utility lies in the following capabilities:

- **Continuous Situational Awareness**: The dashboard auto-refreshes to fetch the latest predictions, allowing security teams to monitor the network's health in real-time without querying raw databases.
- **Rapid Threat Assessment**: By displaying a breakdown of detected threats (Benign vs. Malicious), operators can instantly gauge the severity of ongoing attacks.
- **Visual Anomaly Detection**: Interactive time-series charts allow analysts to visually correlate traffic spikes with malicious activity. This is crucial for identifying volumetric attacks (like DDoS) or scanning patterns that might otherwise be missed in textual logs, enabling faster incident response.

5 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.