Firewall Threat Monitoring and Anomaly Detection Report

---

1. Introduction

In today's digital infrastructure, firewalls are essential for safeguarding networks from cyberattacks. However, with vast volumes of logs being generated continuously, manual identification of incidents becomes impractical. To address this, we developed a system that detects anomalies in firewall logs and presents threat summaries over different timeframes via a user-friendly dashboard.

This document outlines the methodology, data preprocessing, anomaly detection modeling, and development of an interactive dashboard using Streamlit. Special emphasis is placed on how we handled timestamps during post-processing to support temporal summaries.

---

2. Data and Exploratory Data Analysis (EDA)

The dataset used (log2.csv) contains firewall logs capturing various connection details such as source/destination ports, byte counts, packets, and action taken (ALLOW or DENY).

To understand the behaviour of firewall traffic and identify areas that may need further inspection, several exploratory visualizations were conducted:

- **Action Distribution**
    - A simple bar plot shows how many sessions were allowed versus denied.
    - Helps highlight potential misconfigurations or abnormal volumes of blocked traffic that could indicate scanning or brute-force attempts.
- **Top Destination Ports**
    - Plots the frequency of traffic across destination ports.
    - Common ports like 80 (HTTP), 443 (HTTPS), and 3389 (RDP) are expected. Repeated use of uncommon or high-numbered ports may signal tunneling, unauthorized access, or malware communication.
- **Session Duration (Elapsed Time)**
    - A histogram illustrates the distribution of session durations.
    - Very short sessions may imply failed login attempts or scans, while overly long sessions can indicate persistent malware, data exfiltration, or streaming activity.
- **Data Transfer Patterns**
    - Includes both boxplots and histograms for bytes, bytes_sent, bytes_received, pkts_sent, and pkts_received.
    - Useful to catch inconsistencies such as high sent bytes with very low received bytes (or vice versa), or cases where packet counts are inflated but data size is small — both possible signs of suspicious behavior or Denial-of-Service attempts.

- **Elapsed Time vs. Traffic Scatter Plot**
    - Plots session duration on one axis and data volume (bytes/packets) on the other.
    - Designed to verify if longer sessions correlate with higher data exchange. Outliers here may be idle connections kept alive for suspicious reasons or slow exfiltration attacks.
- **Correlation Heatmap**
    - A matrix-style visualization showing how different numeric features relate to each other.
    - Helps identify which variables are strongly related (e.g., packets and bytes), aiding in feature selection and understanding redundancy.

---

3. Data Preprocessing and Feature Engineering

To prepare the firewall session data for anomaly detection, a structured preprocessing pipeline was implemented. The key steps are summarized below:

- **Data Cleaning**
    - The raw CSV file is loaded from a static path.
    - Missing values are handled as follows:
        - For numeric fields: missing values are filled using the column median.
        - For non-numeric fields: filled using the most frequent (mode) value.
    - Any rows still containing missing values after imputation are dropped to ensure model robustness.
- **Categorical Encoding**
    - All categorical features (e.g. actions) are label encoded to numerical form for compatibility with ML algorithms.
- **Selective Feature Scaling**
    - Only a subset of raw numeric features is standardized using StandardScaler. This ensures features like session time or byte counts are on a comparable scale without distorting engineered features.
- **Engineered Features**
  Several domain-relevant features were created to enhance anomaly detection:
    - byte_ratio: Computed as Bytes Sent / Bytes Received (with smoothing to avoid division by zero).
        - **Why it matters**:
            - A high ratio suggests outbound-heavy sessions — possibly data exfiltration or malware beaconing.

        - A low ratio implies large inbound data, which could signal malware delivery or flooding.
    - packet_ratio: Similar logic as above, using Packets Sent / Packets Received.
        - **Why it's useful**:
            - Balanced packet counts are typical.
            - Extreme imbalances (e.g., high outgoing packets) may indicate scanning behaviour or volumetric attacks.
    - src_port_entropy_bin and dst_port_entropy_bin:
        - Categorize ports into **Well-known (0–1023)**, **Registered (1024–49151)**, and **Dynamic (49152–65535)**.
        - These bins reflect the expected use cases of ports:
            - Well-known ports are tied to standard services (e.g., HTTP, SSH).
            - Dynamic ports are ephemeral and often used by clients, but may also be abused.
        - Identifying sessions using rare or unexpected port types can highlight abnormal usage.
    - nat_port_shift:
        - Calculated as the absolute difference between the **NAT Destination Port** and the **actual Destination Port**.
        - **Why it's interesting**:
            - In well-configured systems, this difference is often small or predictable.
            - Large or inconsistent shifts may signal network misconfigurations or evasive routing tactics.
    - is_suspicious_port:
        - A binary flag (1 or 0) set based on whether the destination port falls within a pre-defined suspicious list (e.g., associated with malware C2 servers or unwanted remote access tools).

---

4. Anomaly Detection Modeling

The goal of the model was to detect unusual or malicious activity without supervised labels. Three unsupervised learning algorithms were considered:

- **Isolation Forest**: Efficient for high-dimensional data; identifies anomalies by isolating observations.
- **One-Class SVM**: Learns the frontier of normal data points; sensitive to outliers.

- **DBSCAN**: Density-based clustering used to label sparse regions as anomalies.

These algorithms were implemented in separate functions and invoked from main.py. After training, each algorithm outputs a binary flag per row: -1 for anomaly, 1 for normal.

Threat Tagging Rules

After detecting anomalies using the ML model, each anomalous session is categorized into a specific **threat type** based on rule-based logic. The tagging is performed using the following checks:

- **Normal**
    - If the session is not flagged as an anomaly (anomaly == 1), it's considered normal traffic.
- **Intrusion Attempt**
    - If the action was deny or reset-both, it likely indicates an attempt that was blocked or forcefully terminated — commonly seen in brute-force or scanning attacks.
- **Traffic Spike**
    - If the total Bytes transferred is in the top 1% of all sessions, the traffic is unusually high — could indicate data exfiltration or flooding.
- **Packet Drop / Error**
    - If either pkts_sent or pkts_received is zero, it suggests incomplete or failed communication — possibly a misconfiguration or blocked session.
- **Suspicious Port Activity**
    - If the destination port is in a known list of risky or sensitive ports (risky_ports), the traffic is flagged as potentially dangerous.
- **Malware Communication**
    - If none of the above apply but the session is still anomalous, it's labeled as suspicious traffic possibly tied to malware or command-and-control behavior.

Each session is assigned one of these threat types to help downstream analysis and dashboard reporting.

Note on Timeframe Handling

Initially, the ML model was trained without any notion of time. The dataset lacked timestamps, and since anomaly detection is unsupervised, the model operates purely on feature space.

To enable time-based summaries in the dashboard, we post-processed the model output by assigning synthetic timestamps. Each row was assumed to represent a minute-long event. A new column timestamp was added as a datetime sequence starting from a fixed point (e.g., 2024-01-01 00:00:00) using:

pd.date_range(start='2024-01-01', periods=len(df), freq='T')

This crucial design decision allowed us to simulate real-time monitoring without impacting model performance.

---

5. Summary Generation and Utility Functions

In src/utils.py, we created two key functions:

- get_summary(df, timeframe): Groups anomalies and threats by selected timeframe (1H, 12H, 24H) using pandas resample.
- get_anomaly_summary_text(df): Computes global anomaly statistics (e.g., % of anomalies, total records).

These utilities abstract core logic needed for reporting and enable dynamic dashboard interaction.

---

6. Dashboard Development with Streamlit

The app was built in app.py using Streamlit. Key features include:

Sidebar Options

- **Algorithm Selector**: Users can choose between Isolation Forest, One-Class SVM, and DBSCAN.
- **Timeframe Selector**: Allows filtering incident summaries by 1H, 12H, or 24H windows.

Data Loading

To avoid repeated model retraining, the model prediction is cached using @st.cache_data. After predicting anomalies, timestamps are added for reporting.

Threat Summary Table

The output of get_summary() is shown as a dataframe with counts of different threats and anomalies.

Interactive Plot

A stacked bar chart using Plotly shows how various incident types evolve over the selected timeframes.

Global Anomaly Stats

The get_anomaly_summary_text() is rendered as a Markdown table, providing users with a snapshot of how many anomalies were found in the entire dataset.

Raw Data Viewer

An expandable section displays the first 100 rows of the processed data, useful for debugging and audits.

---

7. Technical Stack and Compatibility

The following Python libraries were used:

- pandas, numpy, scikit-learn, plotly, streamlit
- All packages are compatible with **Python 3.9**.

Pinned versions were carefully selected to ensure stability:

pandas==1.5.3

numpy==1.23.5

scikit-learn==1.2.2

plotly==5.14.1

streamlit==1.27.2

---

8. Project Structure

project/

├── app.py ← Streamlit dashboard

├── EDA_notebook/

│ └──eda_firewall_logs.ipynb ← EDA notebook

├── config/

│ └── config.yaml ← configuration

├── data/raw/

│ └── log2.csv ← Static firewall log data

├── src/

│ ├── preprocessing.py ← Preprocessing

│ └── utils.py ← Summaries and helper functions

│ └── models.py ← models used for anomaly detection

├── main.py

├── report /

│ └── Firewall Threat Monitoring and Anomaly Detection Report.doc

---

9. Conclusion and Future Work

This project successfully combines unsupervised anomaly detection with real-time-inspired monitoring for firewall logs. Although timestamps are synthetic, they effectively simulate monitoring scenarios. The Streamlit dashboard provides intuitive insights and supports security analysts in identifying patterns over different windows.

Future enhancements may include:

- Integrating live data ingestion
- Expanding threat detection rules
- Export options (PDF reports)
- Alert notifications for high-severity threats
