Acta Polytechnica **66**(5):553–564, 2026
https://doi.org/10.14311/AP.2026.66.0553
© 2026 The Author(s). Licensed under a CC-BY 4.0 licence
Published by the Czech Technical University in Prague

ANOMALY, ATTACK, AND VULNERABILITY DETECTION IN TRANSMITTED NETWORK TRAFFIC USING MACHINE LEARNING ALGORITHMS

Erika Abigail Katonova, Rastislav Petija*, Peter Fecilak, Tatiana Goloviznina, Radoslav Cider

Technical University of Kosice, Letna 1/9, 040 01 Kosice, Slovakia

* corresponding author: rastislav.petija@cnl.sk

Abstract. This research focuses on developing of a network monitoring tool designed to detect vulnerabilities and suspicious patterns in the communication networks of two levels: application and standard user data flows. We demonstrate that the ML algorithm can improve the real-time detection of security threats within network infrastructures. A CNN-LSTM autoencoder mode was proposed, which learned to reconstruct network flows from normal data and recognise anomalies based on reconstruction errors. Also, a LSTM and Vulnerability classifier was proposed for further classification of attacks and vulnerabilities in anomalous data. The results of the tests demonstrated the model's high success rate in detecting anomalous traffic. The proposed system was able to detect anomalies, four attack types (Portscan, SQL injection, XSS, and DoS) and three vulnerability classes (SQL, Redis, and sensitive data transmitted in cleartext). The F1 score for anomaly detection was 0.89 and the error rate for attack detection was in an interval of 1–5 %.

Keywords: Anomaly detection, machine learning, standard network profile, unsupervised learning, network security.

1. Introduction

Anomaly detection in network traffic is a critical aspect of cybersecurity, as it enables the identification of irregular data flows that could indicate potential threats. Cyberattacks such as unauthorized access, Distributed Denial-of-Service (DDoS) attacks, and malicious intrusions pose significant risks to the confidentiality, integrity, and availability of data, known as the CIA triad, which forms the foundation of information security.

The rapid advancement of network technologies introduced new challenges for detection systems, requiring real-time monitoring and methods capable of recognising previously unseen traffic patterns. Machine learning (ML) has proven to be effective in dynamic network environments, offering the ability to quickly identify deviations from the standard behaviour. This adaptability is especially important given the complexity and variability of modern data structures. ML techniques can significantly improve detection accuracy and responsiveness, benefiting not only large-scale infrastructures, but also individual users.

As threats continue to evolve, traditional detection methods often fail to address sophisticated attacks. While standard statistical models can identify simple anomalies, they lack the flexibility to adapt to complex behavioural patterns common in today's networks. This limitation requires the integration of adaptive and intelligent approaches. Unlike rule-based systems that rely on predefined behaviour, ML algorithms can effectively detect unknown threats, making them a vital tool in modern data protection strategies.

While there are many known attack methods, networks are often resilient against them. However, in large networks, human factors can introduce vulnerabilities, creating opportunities for malicious actors. Additionally, as systems grow, new weaknesses may emerge as developers focus on new functionalities rather than security. Continuous network monitoring is, therefore, essential in preventing attacks. Manual inspection is impractical, as it's impossible to track all data frames, and humans may overlook malicious patterns. Thus, an automated and reliable tool is necessary.

Some companies lack the resources or time to develop their own monitoring solutions, leaving them vulnerable. They need easily integrable systems that can accurately identify weaknesses.

The goal of this research is to provide a secure and easily integrated solution for networks of any size. It continuously monitors network traffic, detect suspicious patterns, and alert users to potential data security threats or exploitable risks before attackers can gain unauthorised access.

2. Anomaly detection techniques and standard network profiles

This section reviews the current techniques for anomaly detection in network traffic, with a focus on unsupervised learning and standard network profiling. The goal is to highlight the most effective approaches

that do not require labelled datasets and to examine the types of network profiles used as baselines to identify abnormal behaviour.

2.1. Overview of anomaly detection techniques

Unsupervised learning methods enable anomaly detection without the need for labelled training data. This reduces the reliance on manual annotations and improves scalability. Common unsupervised techniques include clustering, autoencoders, and hybrid deep learning models.

Traditional threshold-based methods, while simple, are often inadequate in dynamic environments due to their limited adaptability [1]. By contrast, machine learning approaches adapt better to evolving data distributions and uncover subtle irregularities.

2.1.1. Clustering-based methods

Clustering techniques, such as *k-means*, hierarchical clustering, and DBSCAN, are widely used for anomaly detection in network traffic. These methods attempt to partition data into groups of similar behaviour, identifying anomalies as points deviating significantly from any cluster.

K-means clustering is computationally efficient and ideal for environments with stable traffic patterns. However, it assumes a fixed number of symmetric clusters, which may not reflect the real-world variability in network data [2]. Moreover, it performs poorly in the presence of noise or non-uniform cluster distributions.

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) addresses these limitations by using density thresholds rather than fixed cluster counts. This makes DBSCAN better suited for heterogeneous datasets and dynamic network environments. It identifies low-density regions as potential anomalies, allowing detection in noisy or irregularly structured data [2].

Clustering has proven valuable in various applications. In industrial networks, DBSCAN has been used to analyse sensor data streams and detect operational anomalies such as irregular temperature or voltage readings [3]. Similarly, *k-means* has been applied to energy management systems for classifying consumption patterns and identifying outlier behaviour [4].

2.1.2. Autoencoders for detection

Autoencoders are neural networks trained to reconstruct their input data. Anomalies are identified based on a reconstruction error – if the output differs significantly from the input, the data point is likely anomalous. These models can learn complex relationships between high-dimensional data, making them suitable for cybersecurity applications.

Autoencoders have been used to detect botnet activity in IoT networks, where they capture behavioural patterns of normal traffic and flag deviations as potential threats [5]. They are also used in system log analyse. For instance, the AutoLog tool leverages reconstruction errors to classify log entries as normal or suspicious [6].

Deep Neural Network (DNN), Convolutional Neural Network (CNN) [7], and recurrent models such as Long Short-Term Memory (LSTM) have been widely studied for their effectiveness in identifying time-dependent or sequential attack behaviours. For example, Kitsune [8] employed a stream of autoencoders for online anomaly detection in Local Area Network (LAN) environments.

Although resource-intensive, autoencoders are highly flexible and can be retrained to adapt to new traffic patterns. Their deep representations allow accurate detection of subtle anomalies, even in complex or evolving network environments.

2.1.3. Information-theoretic methods

Information-theoretic approaches provide a powerful means to detect anomalies by analysing the distribution and complexity of network data flows. Techniques such as Shannon entropy and Kullback-Leibler divergence are commonly used to measure the unpredictability and divergence in traffic behaviour.

Shannon entropy, for instance, quantifies the randomness in traffic flows and has been used to detect distributed denial-of-service (DDoS) attacks by identifying abrupt changes in traffic patterns that signify chaotic behaviour. According to [9], the use of entropy in dynamic networks enabled real-time identification of anomalous behaviour and improved the effectiveness of DDoS protection.

Kullback-Leibler divergence, on the other hand, measures the difference between two probability distributions. In network security, it can compare the baseline profile of normal traffic with observed flows to pinpoint deviations that may indicate attacks or other anomalies [10]. This technique is especially useful for scenarios where labelling data is impractical or impossible.

Information-theoretic methods also support feature selection in high-dimensional datasets, effectively identifying the most relevant attributes that contribute to anomaly detection while reducing computational complexity [11]. However, these methods often demand significant processing power, which can limit their real-time applicability in resource-constrained environments. They are best suited for deployment in settings with substantial computational resources or as part of hybrid approaches that combine multiple detection strategies.

2.1.4. Hybrid deep learning approaches

Hybrid approaches combine multiple models, leveraging the strengths of different architectures. CNN-LSTM models, in particular, are effective in analysing both spatial features (via convolutional layers) and temporal dependencies (via recurrent layers).

For example, CNN-LSTM models have been trained on IoT-23 datasets to detect malware, achieving accuracies of over 96 % [12]. Other studies demonstrate the utility of 1D CNN-LSTM networks in detecting a wide range of attacks in Bot-IoT datasets, achieving F1 scores above 99 % [13].

These models are particularly useful in environments where traffic behaviour evolves over time. However, the increased accuracy comes at the cost of higher computational requirements. Deep models require extensive hardware resources, and their training and inference can be time-consuming [14].

2.2. Standard network profiles and their role

Standard network profiles describe expected traffic behaviour and serve as references for detecting abnormal events. Profiles are typically derived from historical records, behavioural modelling, or time-series data [15]. Their utility depends on the environment and detection goals.

2.3. Profile attributes and structures

Common attributes used in profiling include the source and destination IP addresses, port numbers, protocols, timestamps, and flow durations. These metrics provide the context for traffic behaviour and are essential in evaluating deviations from the norm.

Historical Profiles: Constructed from long-term data, historical profiles capture aggregate network behaviour based on past traffic records. Unlike dynamic behavioural profiles, which are continuously updated as devices interact in real time [16], historical profiles focus on long-term trends and recurring patterns without real-time adaptations. They typically use time-series analysis to represent global network behaviour. Although static, they can be enhanced with machine learning to detect deviations, compensating for their limited adaptability.

Behavioural Profiles: These profiles model normal network behaviour dynamically by continuously analysing real-time traffic and updating the model with the new data. They capture device-specific patterns, including temporal sequences, IP-spatial interactions, and application usage [17]. This enables accurate detection of anomalies even in dynamic environments. However, they require significant computational resources and large amounts of data to maintain accuracy.

Statistical Profiles: Relying on statistical measures, such as mean, median, and variance, statistical profiles define expected traffic behaviour in simple terms. They are easy to implement and resource-efficient but may struggle to adapt to rapid changes [18].

Each profiling method has its trade-offs. Historical and statistical models are more efficient, but less adaptive. Behavioural profiles offer better real-time accuracy, but require more complex infrastructure. Integrating these with machine learning can significantly enhance anomaly detection capabilities in diverse network environments.

3. Materials and methods

In complex corporate networks, various types of data are transmitted, ranging from standard user communication to traffic between components of developed applications. To increase accuracy and reliability, our solution is designed to address both containerised microservice and traditional enterprise network traffic. Consequently, our research utilised two distinct datasets: the first captures internal communication within a Docker-based application environment, while the second includes packet data from a standard computer network.

3.1. Dataset from docker network

The selection of an appropriate dataset is critical for evaluating anomaly detection models in containerised environments. While several public network security datasets exist, they often prove to be insufficient due to a lack of the specific communication patterns found in modern web applications, such as the interactions between front-end, back-end, cache, and database. Furthermore, many public datasets lack the granular labelling required to precisely distinguish between "normal" behaviour and "anomalous" behaviour.

However, our motivation was not merely to produce a static dataset, but to validate a reproducible methodology and technological pipeline. We aim to provide a framework that organisations can adapt to their own specific environments. And while the individual traffic profiles may vary by company, the underlying anomaly detection process remains consistent. This focus on a generalised procedure for monitoring and securing containerised applications provides more long-term value than a fixed, external dataset.

3.1.1. Environment simulation

A Docker-based networking infrastructure was implemented to create a realistic environment that mirrors the diversity of traffic of a real-world corporate application, incorporating already mentioned interactions between backend services, database systems, and caching layers. In order to simulate the behaviour of a realistic web application, a Magento [19] e-commerce deployment was configured as the testbed platform.

The architecture schematically consists of an entry point (Nginx web server) connected to a Docker bridge network. The following five primary components were deployed as containers within this network:

- **Frontend:** Nginx and PHP-FPM web server.
- **Opensearch:** Used as the application's search engine database.

Label	Numeric representation	Description
benign	1	Normal communication
mysql	2	SQL communication in plain text
redis	3	Redis communication in plain text
unsecure	4	Sensitive user data sent in plain text
injection	5	Injection attack
xss	6	XSS attack
ddos	7	DOS attack
portscan	8	Portscan attack

Table 1. Labels used for the packets.

- **RabbitMQ:** Handling asynchronous communication and message processing.
- **MySQL/MariaDB:** The primary relational database.
- **Redis:** Utilised for session management and caching.

This controlled setup allowed us to establish a precise baseline, as we could strictly define the transition from normal operational traffic to injected anomalies, ensuring the high-quality labelling necessary for training our model.

3.1.2. Data gathering

Packet-level network data were systematically captured using Wireshark [20] producing raw `.pcap` files that were subsequently converted to CSV format for training. Approximately 6 GB of network traffic was collected over multiple operational days to ensure temporal diversity. The traffic categories include:

- **Legitimate traffic**: Simulated user sessions including authentication sequences, product catalogue interactions, and purchase workflows.
- **Vulnerable behaviour**: Deliberately insecure configurations, such as MySQL or Redis communications and sensitive user data transmitted in plain text.
- **Attack vectors**: Orchestrated penetration scenarios including SQL injection, Cross Site Scripting (XSS), port scanning, and DOS attacks.

The raw captures were processed into two complementary formats, as suggested in [21]:

- Flow-based representations: Derived statistical features characterising connection patterns and protocol behaviours.
- Packet-level records: Preserved raw payloads with categorical annotations indicating attack classifications.

3.1.3. Data labelling

The gathered data were further divided into several groups and labelled. The Table 1 contains the data on used labels.

3.1.4. Dimensionality reduction

The complexity of the Docker-based application traffic required an advanced approach to data dimensionality. The raw dataset consisted of 315 attributes, 256 of which represented the byte frequency distribution of the packet payloads to capture deep structural patterns of malicious traffic. To manage this large number of features, we used an LSTM Autoencoder. It compressed the input data in a non-linear way into a smaller, 40-dimensional representation. This compressed representation (bottleneck layer) was then used for the subsequent detection and classification tasks.

3.2. Dataset from standard network

To train and validate the model, we selected the UNSW-NB15 dataset, a well-established benchmark for intrusion detection research. This dataset includes both legitimate and malicious network flows generated in a controlled environment. It offers detailed flow-level metadata along with labels indicating the presence or absence of an attack.

Key characteristics of UNSW-NB15 that made it suitable for our use case include:

- **Realistic traffic patterns:** The dataset simulates modern attack scenarios in diverse traffic conditions.
- **Timestamps:** Time-stamped entries make it possible to construct ordered flow sequences, which are essential for sequential modelling.
- **Labelled data:** Although our model is trained in an unsupervised fashion, labelled entries are invaluable for post-training evaluation.
- **CSV format:** The dataset's structure in CSV files simplifies the integration with data manipulation tools such as Pandas.

We combined all the provided CSV files into one unified dataset for preprocessing and analysis. Though the model does not rely on the labels during training, these annotations were preserved for later performance metrics calculation.

3.2.1. Data preprocessing

Effective preprocessing is essential to improve the performance and stability of the model. Raw network

traffic data is often noisy, redundant, and unstandardized. To ensure consistent input quality, the following preprocessing steps were performed:

- **Data import:** All records were loaded using the Pandas library, which offers a flexible structure for tabular data analyse.
- **Filtering attributes:** Only the subset of features considered relevant for anomaly detection were selected. However, the non-selected features were kept in order to preserve flexibility.
- **Cleaning:** Missing or undefined values were addressed using imputation. Numeric fields were filled with medians to reduce outlier sensitivity, while categorical ones were imputed using the mode or a placeholder value such as "unknown".
- **Deduplication:** Identical records were removed to reduce redundancy and training time.
- **Temporal feature engineering:** Two new attributes were computed: `delta_time` (the time between consecutive flows) and `hour_of_day` (used to model daily patterns).
- **Categorical encoding:** Non-numeric attributes such as protocol type were transformed via label encoding for compatibility with numerical models.

3.2.2. Feature selection and dimensionality

To ensure that the model remains computationally efficient while maintaining high sensitivity to threats, we performed a targeted feature selection process during the preprocessing stage to refine the model's input, rather than using the full raw attribute set. This approach was chosen to eliminate redundant or non-informative attributes that increased memory overhead without contributing to predictive accuracy, resulting in a finalised set of 18 relevant features: srcip, Spkts, Dpkts, Sload, Dload, tcprtt, synack, ackdat, dsport, proto, state, sttl, dttl, sbytes, dbytes, and dur. In addition to these standard network parameters, we specifically engineered two temporal attributes, `delta_time` and `hour_of_day`, to better capture the nuances of container-based traffic. The inclusion of `delta_time` allows the model to analyse the precise intervals between the incoming packets, while `hour_of_day` provides crucial context for detecting anomalies, as malicious activities in these environments often occur during off-peak hours when standard administrative traffic is lower. This reduction in dimensionality allows the model to ignore noisy or redundant data, leading to faster training and more robust detection.

3.2.3. Standardisation and normalisation

Two distinct scaling strategies were employed to improve model convergence and eliminate scale-based bias:

- **Standardisation:** Applied to features to normalise them to zero mean and unit variance. This transformation helps to balance input features that differ in range or units.
- **Normalisation (MinMaxScaler):** Used to rescale feature values into the $[0, 1]$ interval. This step prevents high-magnitude features from dominating during training.

Normalisation was performed using strictly statistics only from the training set only. This prevents data leakage, which happens when information from the test set unintentionally affects the model's training process.

3.2.4. Sequence generation

For sequential learning, we constructed fixed-length windows of flow sequences using a sliding window mechanism. Each sequence consists of 10 temporally ordered flows. The window was shifted by one flow each time, producing overlapping sequences. This method maximises the number of training samples and helps the model to capture local temporal dependencies.

3.3. Implementation of application data analysis

The first part of our system focuses on the analysis of the application data transmitted in a Docker network. The AI pipeline was divided into two key stages: anomaly detection and attack classification, each using specialised models tailored for the task. The resulting topology of the model is shown in the Figure 1.

A dedicated user interface was intentionally excluded from this work, as the solution was designed to integrate into existing monitoring infrastructures (e.g. SIEM systems such as Splunk [22] or Elastic Security).

3.3.1. Anomaly detection

An anomaly is defined as network traffic that deviates from normal behaviour. To detect such irregularities, an LSTM autoencoder was implemented in PyTorch, and trained exclusively on benign flow-level data. The model used 315 input features, including statistical attributes and a byte-frequency distribution (across 256 payload characters).

Anomalies were identified based on reconstruction error, with a predefined threshold in the configuration file. This approach serves as an efficient initial filter, separating normal traffic from suspicious activity without heavy computational overhead. By flagging deviations early, the system optimises security monitoring while reducing false positives. Key advantages:

- Lightweight detection for high-traffic networks,
- automated anomaly scoring via reconstruction error,
- configurable sensitivity through adjustable thresholds.

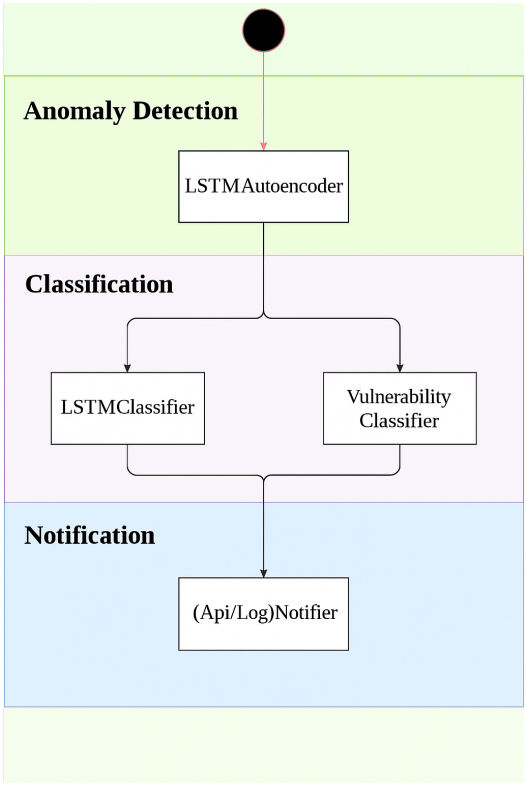

Figure 1. Pipeline topology.

3.3.2. Classification

This module represents the second stage of the analysis, focusing on classifying the type of attack or vulnerability present in the detected anomalous network traffic. After identifying abnormal traffic, it analyses its characteristics and behaviour in order to categorise it into a specific threat type.

The primary goal is to assign precise labels to each anomaly, enabling downstream modules to provide users with clear, actionable insights about the detected security issues. Two parallel classifiers were trained:

- LSTMClassifier for flow-level attack classification. The distribution of the training dataset is shown in Table 2.

  It processed sequences of packet vectors (each sequence = a communication session), and output one-hot encoded labels for attacks. Zero-padding and truncation techniques were used to handle variable sequence lengths.

- Vulnerability Classifier based on TF-IDF vectorisation of packet payloads, followed by KNN classification. The distribution of the training dataset is shown in Table 3.

  Custom preprocessing included token sanitisation and removing noise characters before vectorisation. StandardScaler was applied post-vectorisation for normalisation.

3.3.3. User notification

Existing monitoring tools such as Splunk and Sentry have GUIs, however, this solution prioritises flexibility and integration. Rather than using a GUI, it uses an API interface for seamless compatibility with existing systems. This makes deployment easier, avoids workflow disruptions, and ensures scalability across different environments.

Number of rows in the test set	Label
852	BENIGN
312	SQL_INJ
2 587	XSS
1 383	DOS
12 500	PORTSCAN

Table 2. Data distribution in the LSTMClassifier test set.

Number of rows in the test set	Label
19 272	PLAIN_SQL
12 174	PLAIN_REDIS
5 795	PLAIN_HTML

Table 3. Data distribution in the vulnerability classifier test set.

The application supports file logging as well. Both notification methods are configurable via a `config.cfg` file. At least one notification method must be enabled for the system to operate.

- File Logging:
  - ▷ Writes security alerts to a specified log file path.
  - ▷ Formats notifications with attack details (port, error score, attack type).
- API Notification:
  - ▷ Sends structured JSON alerts via POST requests.
  - ▷ Includes comprehensive attack metadata (Source/destination IPs and ports, Threat classification, Anomaly score).

3.4. Implementation of standard network data flow analysis

The second part of our system focuses on a comprehensive anomaly detection system for standard network traffic. It outlines the entire architecture, from data processing and model training, to deployment and user interaction. The implementation consists of several interconnected components that work together to process, analyse, detect, and visualise anomalous patterns in real-time network data. These components include a machine learning model based on a hybrid CNN-LSTM autoencoder, a backend server developed using Flask, a PostgreSQL relational database for persistent storage, and a React-based frontend for user interaction.

3.4.1. System objective

The primary goal of this part the system is to detect anomalies in network behaviour by identifying deviations from the learned patterns of standard network activities. This task is performed using a model trained on historical, dominantly normal data. Instead of relying on manually labelled examples of attacks, the system assumes an unsupervised learning relying on autoencoder-based reconstruction to flag unusual behaviours. This is particularly suitable for real-world deployments, where labelled attack data is scarce or non-existent.

By combining CNNs for spatial feature extraction and LSTM networks for capturing temporal dependencies, the model can process sequential traffic flows and distinguish normal from abnormal activity. The system operates in a near-real-time monitoring loop, where new data are continuously evaluated against the model and results are provided through a responsive, web-based interface.

3.4.2. Model implementation

The anomaly detection model is designed as a deep autoencoder consisting of convolutional and recurrent layers:

- **Input Layer:** Accepts sequences of shape $(10, N)$, where N is the number of pre-processed features.
- **Convolutional Layers:** Three 1D convolutional layers with filter sizes of 1, 3, and 5 are applied in parallel to capture different spatial patterns across flows.
- **Concatenation:** Outputs from all CNN paths are concatenated to produce a rich, combined representation.
- **Dropout Layer:** Randomly deactivates neurons during training to prevent overfitting and improve generalisation.
- **LSTM Encoder:** Processes the concatenated representation to summarise the entire sequence into a latent vector.
- **RepeatVector:** Duplicates the latent vector across time steps to prepare it for decoding.
- **LSTM Decoder:** Attempts to reconstruct the original sequence from the latent vector.
- **Output Layer:** A TimeDistributed Dense layer outputs the reconstructed sequence.

The training is conducted using the Mean Squared Error (MSE) loss function, which measures the difference between the original and reconstructed sequences. Anomalies are detected by computing the reconstruction error at an inference time. Samples with an error exceeding a predefined threshold are classified as anomalies. The Adam optimiser is used due to its ability to handle sparse gradients and variable learning rates.

3.4.3. Configuration file

All system parameters, including data paths, selected feature lists, model thresholds, and visualisation settings, are maintained in a centralised JSON configuration file. This allows flexible adjustments without modifying the source code. It supports deployment across environments by abstracting the settings from the implementation logic.

3.4.4. Flask API server

The Flask-based backend exposes a set of RESTful endpoints that bridge the frontend and the database. It provides a real-time access to network traffic analytics and alerts.

Key endpoints include:

- `/top-ips`: Returns IPs sorted by anomaly frequency.
- `/delete-all-alerts`: Clears all alerts from the database.
- `/ip-details`: Returns detailed metadata about a selected flow.

All responses are serialised in a JSON format, ensuring compatibility with frontend libraries and other consumers.

3.4.5. Database

A PostgreSQL database serves as the layer for storing processed flows and anomaly alerts. It includes two main tables:

- **network_traffic**: Contains the full record of the evaluated traffic, including extracted features and anomaly scores.
- **alerts**: Stores alerts triggered by the model, which are immediately displayed on the frontend.

The schema is optimised for fast read/write performance and historical querying, which is essential for retrospective analysis and model retraining.

3.4.6. User interface

The frontend is implemented as a dynamic, single-page application using React. It provides a responsive and intuitive interface for security monitoring. Features include:

- **Top IPs Table:** Lists IP addresses ranked by anomaly count.
- **Pie Chart:** Shows the ratio of normal to anomalous flows.
- **Recent Alerts Feed:** Displays the detected anomalies with relevant metadata. Users can view details or clear alerts.

Data are fetched using HTTP requests and automatically updates to reflect the latest detections. This provides operators with real-time visibility and actionable insights into network security. The final GUI of the system is shown in Figure 2.

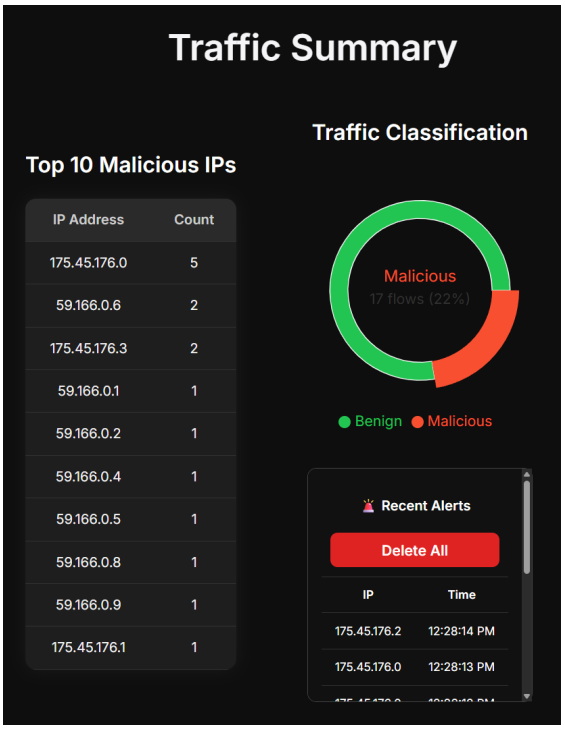

Figure 2. Web interface displaying real-time anomaly detection dashboard.

Together, these components form a modular and extensible system capable of operating in diverse network environments, providing transparency, automation, and user engagement.

4. Results

As our system provides a data analysis of anomalies and attack/vulnerability detection for application data and standard network traffic, the evaluation of the system is divided into two subsection.

4.1. Evaluation of model training progress for application data

Different metrics were used to measure the performance of each of the three trained models. The training history was saved and later visualised using Python.

4.1.1. LSTM autoencoder

To evaluate this model, the MSELoss metric was used. The R^2 (R-squared) value was also measured, but the loss carried a greater significance. The results, shown in Figure 3, were sufficiently good for the distinction between normal and malicious packets, and therefore the training was terminated.

The LSTM autoencoder converged rapidly (8 epochs, final loss = 0.0089, $R^2 = 0.76$) with a 0.150 separation between normal (0.008) and malicious (0.123) packet reconstruction losses.

4.1.2. LSTM classifier

The CrossEntropyLoss metric was used to evaluate this model. The results are shown in Figure 4.

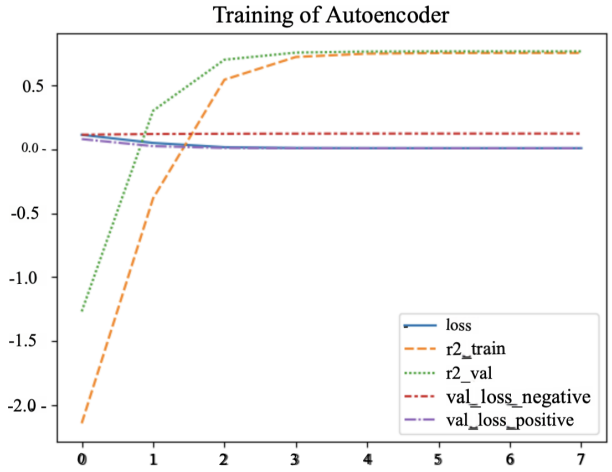

Figure 3. LSTM autoencoder training: `train_loss` (legitimate), `val_loss_positive` (validation), `val_loss_negative` (malicious).

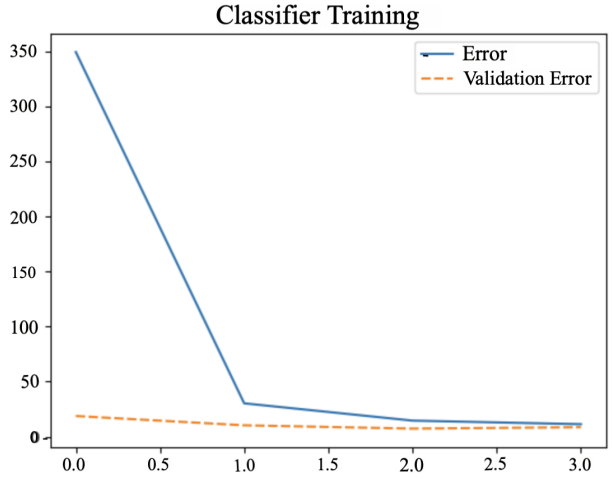

Figure 4. Training history of the LSTM classifier.

A total of 4 training epochs were performed, with a final training loss of 11.37 and a validation loss of 8.65.

4.1.3. Vulnerability classifier

The metrics used to evaluate this model were precision, recall, and F1-score. The resulting metric values were 0.93 for F1-score, 0.93 for precision, and 0.93 for recall.

4.2. Real world testing of application data analysis

The system was evaluated under three scenarios: normal traffic, vulnerable communications (unencrypted payloads), and simulated attacks.

4.2.1. Normal traffic

Three 10-minute test captures were performed in an encrypted attack-free environment. Testing statistics were collected from Splunk, which enabled alert notification categorisation. Various anomaly detection thresholds were applied. An ideal model should generate minimal or zero alerts under these conditions.

- **Low anomaly threshold**:
  - ▷ Total flows: 1 136,
  - ▷ Alerts triggered: 65,
  - ▷ Breakdown: 58 % attacks, 8 % vulnerabilities, 3 % unclassified suspicious traffic,
  - ▷ Error rate: **5 %**.
- **Medium anomaly threshold**:
  - ▷ Total flows: 1 156,
  - ▷ Alerts triggered: 54,
  - ▷ Breakdown: 83 % attacks, 1 % vulnerabilities, 14 % unclassified suspicious traffic,
  - ▷ Error rate: **4.6 %**.
- **High anomaly threshold**:
  - ▷ Total flows: 1 156,
  - ▷ Alerts triggered: 12,
  - ▷ Breakdown: 91 % attacks, 8 % unclassified suspicious traffic,
  - ▷ Error rate: **1 %**.

4.2.2. Vulnerable communications

Testing in an unencrypted environment involved short 200-flow captures containing plaintext SQL/Redis queries (no real attacks) to evaluate the vulnerability detection accuracy and false positives (misclassified benign traffic).

- **Low anomaly threshold** (sensitive):
  - ▷ Total flows: 200,
  - ▷ Alerts triggered: 72,
  - ▷ Classification: 82 % attacks, 18 % vulnerabilities.
- **Medium anomaly threshold** (balanced):
  - ▷ Total flows: 200,
  - ▷ Alerts triggered: 54,
  - ▷ Classification: 72 % attacks, 28 % vulnerabilities.
- **High anomaly threshold** (strict):
  - ▷ Total flows: 200,
  - ▷ Alerts triggered: 12,
  - ▷ Classification: 84 % attacks, 15 % vulnerabilities.

4.2.3. Simulated attacks

To test the accuracy of attack detection, short-duration tests were conducted in a vulnerable environment. Each test included:

- 3 XSS attack vectors,
- 3 SQL injection attempts,
- Sequential scanning of 50 network ports.
- **High anomaly threshold** (strict filtering):
  - ▷ Total detected attacks: 12.
  - ▷ Attack distribution:
    - – PORTSCAN: 3,
    - – DOS: 3,
    - – XSS: 2,
    - – SQL injection: 2.
- **Low/Medium anomaly thresholds** (permissive filtering):
  - ▷ Total detected attacks: 10.
  - ▷ Attack distribution:
    - – PORTSCAN: 3,
    - – SQL injection: 1,
    - – XSS: 1,
    - – DOS: 5.

4.3. Evaluation of the anomaly detection model for standard user traffic

The objective of the evaluation is to assess the model's capability to distinguish anomalous network flows from normal ones. We used a controlled offline environment and multiple quantitative metrics to evaluate the performance. This section outlines the evaluation methodology, metric definitions, result interpretation, and confusion matrix analysis.

4.3.1. Evaluation methodology

All experiments were conducted using the UNSW-NB15 dataset in an offline testing scenario. The model was trained exclusively on normal data, with a small subset of anomalous samples prepared for the evaluation. We used fixed-length sequences of 10 flows as the input, each sequence representing a continuous time window of traffic. The CNN-LSTM autoencoder reconstructed these sequences, and the reconstruction error was used as the decision criterion.

The assumption is that normal sequences will exhibit low reconstruction error, while anomalous ones, due to unfamiliar patterns, will result in higher reconstruction errors. A reconstruction threshold was selected by analysing the error distribution on a validation set, balancing sensitivity and specificity.

4.3.2. Evaluation results

We conducted experiments with varying model parameters and reconstruction thresholds. The goal was to find an optimal balance between precision and recall.

Summary of results:

- Accuracy: 79 %–91 %.
- Precision: 70 %–91 %.
- Recall: 80 %–90 %.
- F1-score: 82 %–91 %.

Although a lower threshold increased recall, it also led to more false positives. Conversely, a higher threshold reduced false alarms, but increased false negatives. The results also demonstrated the impact of the threshold on both classes individually.

Table 4 shows an excellent recall for anomalies (98 %) – almost all malicious flows were correctly identified. Precision for anomalies was 82 %, meaning 18 % of the flagged flows were false positives. Normal traffic was also well-identified, with a precision of 97 % and a recall of 79 %.

Class	Precision	Recall	F1-score	Count
Normal (0)	0.97	0.79	0.87	18 883
Anomaly (1)	0.82	0.98	0.89	18 883
Macro Avg.	0.90	0.88	0.88	-

Table 4. Class-wise classification metrics.

4.3.3. Confusion matrix

To further visualise the performance of the model, a confusion matrix is provided in Figure 5.

The rows represent the true labels and the columns represent the predicted labels. This matrix supports our findings that the model is highly sensitive to anomalies, although it occasionally mislabels normal flows.

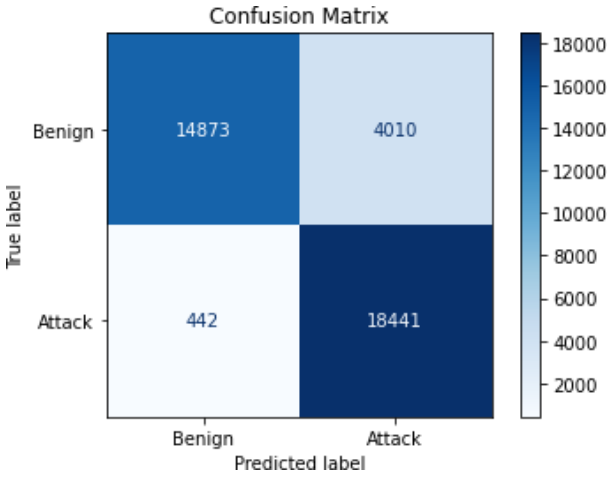

Figure 5. Confusion matrix for anomaly detection model.

5. Discussion

There are few considerations that should be discussed.

5.1. Recommendations for application data analysis system

The tool should be integrated via its REST API interface. The key implementation requirements include:

(1.) **Notification system**:

- Real-time alerts delivered as JSON-formatted POST requests,
- Configurable endpoint URL (user-provided).

(2.) **Receiver requirements**:

- Guaranteed server availability,
- Proper HTTP status code handling (200/201 for successful delivery),
- Queue management for high-availability scenarios.

(3.) **Message format example**:

```
{
  "dst_ip": "239.255.255.250",
  "dst_port": 1900,
  "name": "DDOS",
  "payload": "...",
  "score": 0.14269817886212338,
  "src_ip": "192.168.32.1",
  "src_port": 35882,
  "type": "attack"
}
```

5.2. Summary of standard data analysis

Our model is optimised for high sensitivity, which is desirable in security applications where missing an anomaly is riskier than triggering a false alarms. Although precision could be improved, the ability to detect nearly all anomalies provides robust protection. These results confirm that an unsupervised CNN-LSTM autoencoder is effective in detecting anomalies, especially when trained on high-quality normal data.

The evaluation also revealed the model's sensitivity to the reconstruction threshold. Adjusting this value has a substantial impact on its performance. In practice, threshold tuning should be done iteratively using validation data.

Despite being trained only on normal flows, the model demonstrated generalisation to unknown anomalies. This adaptability is crucial for real-world deployment, where new threats can significantly differ from the training data.

6. Conclusion

This research presents an AI-powered network security tool that detects vulnerabilities and malicious activity by analysing network traffic patterns, designed as an easily implementable solution.

In the first step, the system was trained and tested using a synthetic environment with virtual servers simulating e-commerce traffic, the only limitation being protocol diversity and multi-user scenarios. Currently capable of identifying four attack types (Portscan, SQL injection, XSS and DoS) and three vulnerability classes (SQL, Redis and sensitive data transmitted in cleartext), the tool demonstrated promising detection accuracy while maintaining computational efficiency.

Future work should expand the detection capabilities to additional threat types, improve multi-user scenario handling, and extend support for various network topologies beyond the current configuration.

The second step began with the design of a concept solution, followed by an implementation of an unsupervised anomaly detection model using a hybrid

CNN-LSTM autoencoder architecture. The model was trained on a standard profile of normal network behaviour. It leverages reconstruction error as a decision metric: low error implies normal traffic, while high error signals potential anomalies. This model successfully learned to identify patterns characteristic of normal activity and could effectively flag deviations. Experimental evaluation confirmed the effectiveness of the proposed solution. The model demonstrated high anomaly detection capability across various testing conditions, achieving a solid performance in terms of precision, recall, and F1-score. Its ability to generalise from normal training data to previously unseen anomalies was a key finding. Despite the positive results, certain limitations remain. The model was trained and tested on a specific dataset with a predefined set of features, which may limit generalisation to other traffic profiles without retraining or reconfiguration. Enhancing the performance and integrating real-time streaming support would be necessary for production use.

Future improvements could include the integration of IP reputation scoring to enrich the anomaly context and better identify malicious actors. Moreover, the system could be extended to classify specific types of attacks, adding more actionable insights.

References

[1] E. F. Agyemang. Anomaly detection using unsupervised machine learning algorithms: A simulation study. *Scientific African* **26**:e02386, 2024. https://doi.org/10.1016/j.sciaf.2024.e02386

[2] M. Usama, J. Qadir, A. Raza, et al. Unsupervised machine learning for networking: Techniques, applications and research challenges. *IEEE Access* **7**:65579–65615, 2019. https://doi.org/10.1109/ACCESS.2019.2916648

[3] H. S. Emadi, S. M. Mazinani. A novel anomaly detection algorithm using DBSCAN and SVM in wireless sensor networks. *Wireless Personal Communications* **98**(2):2025–2035, 2018. https://doi.org/10.1007/s11277-017-4961-1

[4] R. Aryani, M. Nasrun, C. Setianingsih, M. A. Murti. Clustering data in power management system using k-means clustering algorithm. In *2019 IEEE Asia Pacific Conference on Wireless and Mobile (APWiMob)*, pp. 164–170. 2019. https://doi.org/10.1109/APWiMob48441.2019.8964143

[5] M. A. Alsoufi, M. M. Siraj, F. A. Ghaleb, et al. An anomaly intrusion detection systems in IoT based on autoencoder: A review. In F. Saeed, F. Mohammed, Y. Fazea (eds.), *Advances in Intelligent Computing Techniques and Applications*, pp. 224–239. Springer Nature Switzerland, Cham, 2024. https://doi.org/10.1007/978-3-031-59707-7_20

[6] M. Catillo, A. Pecchia, U. Villano. Autolog: Anomaly detection by deep autoencoding of system logs. *Expert Systems with Applications* **191**:116263, 2022. https://doi.org/10.1016/j.eswa.2021.116263

[7] I. Al-Turaiki, N. Altwaijry. A convolutional neural network for improved anomaly-based network intrusion detection. *Big Data* **9**(3):233–252, 2021. https://doi.org/10.1089/big.2020.0263

[8] Y. Mirsky, T. Doitshman, Y. Elovici, A. Shabtai. Kitsune: An ensemble of autoencoders for online network intrusion detection, 2018. arXiv:1802.09089. https://doi.org/10.48550/arXiv.1802.09089

[9] P. Blažek, J. Hajný. Identifikace anomálií v datové komunikaci pomocí entropie. *Elektrorevue* **18**(4):119–123, 2016.

[10] E. Sabeti, S. Oh, P. X. K. Song, A. O. Hero. A pattern dictionary method for anomaly detection. *Entropy* **24**(8):1095, 2022. https://doi.org/10.3390/e24081095

[11] M. Ozkan-Okay, R. Samet, Ö. Aslan, et al. A novel feature selection approach to classify intrusion attacks in network communications. *Applied Sciences* **13**(19):11067, 2023. https://doi.org/10.3390/app131911067

[12] A. A. A. M. H. A. Zubaidi. IoT malware detection based on anomaly traffic identification using CNN-LSTM. In *2024 16th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*, pp. 1–6. 2024. https://doi.org/10.1109/ECAI61503.2024.10607471

[13] H. Benaddi, M. Jouhari, K. Ibrahimi, et al. Improvement of anomaly detection system in the IoT networks using CNN-LSTM approach. In *GLOBECOM 2023 – 2023 IEEE Global Communications Conference*, pp. 3771–3776. 2023. https://doi.org/10.1109/GLOBECOM54140.2023.10437475

[14] I. Goodfellow, Y. Bengio, A. Courville. *Deep learning*. MIT Press, Cambridge, MA, USA, 2016. [2025-05-19]. http://www.deeplearningbook.org

[15] J. Yi, S. Zhang, L. Tan, Y. Tian. A network traffic abnormal detection method: Sketch-based profile evolution. *Applied Sciences* **13**(16):9087, 2023. https://doi.org/10.3390/app13169087

[16] S. P, K. Palaniappan, B. Duraipandi, U. M. Balasubramanian. Dynamic behavioral profiling for anomaly detection in software-defined IoT networks: A machine learning approach. *Peer-to-Peer Networking and Applications* **17**(4):2450–2469, 2024. https://doi.org/10.1007/s12083-024-01694-y

[17] K. Xu, Y. Wan, G. Xue, F. Wang. Multidimensional behavioral profiling of internet-of-things in edge networks. In *Proceedings of the International Symposium on Quality of Service*, IWQoS '19. Association for Computing Machinery, New York, NY, USA, 2019. https://doi.org/10.1145/3326285.3329072

[18] M. S. Gill, D. Lindskog, P. Zavarsky. Profiling network traffic behavior for the purpose of anomaly-based intrusion detection. In *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications / 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pp. 885–890. 2018. https://doi.org/10.1109/TrustCom/BigDataSE.2018.00127

[19] Adobe. Adobe Commerce (Magento) B2B- und B2C-Unternehmenslösungen – Adobe – Adobe Deutschland. [2025-05-19]. https://business.adobe.com/uk/products/commerce.html

[20] Wireshark. Wireshark – Go deep. [2025-05-19]. https://www.wireshark.org

[21] K. Wang, S. J. Stolfo. Anomalous payload-based network intrusion detection. In E. Jonsson, A. Valdes, M. Almgren (eds.), *Recent Advances in Intrusion Detection*, pp. 203–222. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. https://doi.org/10.1007/978-3-540-30143-1_11

[22] Splunk. Splunk – Data platform for security, observability & AI. [2025-05-19]. https://www.splunk.com