National College of Ireland

# Improving API Security in the Cloud with AI/ML

MSc Research Project
MSc Cybersecurity

Rona Shaji
Student ID: 23292709

School of Computing
National College of Ireland

Supervisor: Vikas Sahni

**National College of Ireland**

**MSc Project Submission Sheet**

**School of Computing**

| | |
|---|---|
| **Student Name:** | Rona Shaji |
| **Student ID:** | 23292709 |
| **Programme:** | MSc Cybersecurity **Year:** 2024-2025 |
| **Module:** | MSc Practicum |
| **Supervisor:** | Vikas Sahni |
| **Submission Due Date:** | 15th September 2025 |
| **Project Title:** | Improving API Security in the Cloud with AI/ML |

**Word Count:** 6552 **Page Count:** 22

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

**Signature:** Rona Shaji

**Date:** 14-09-2025

**PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST**

| | |
|---|---|
| Attach a completed copy of this sheet to each project (including multiple copies) | □ |
| **Attach a Moodle submission receipt of the online project submission,** to each project (including multiple copies). | □ |
| **You must ensure that you retain a HARD COPY of the project**, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer. | □ |

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

| **Office Use Only** | |
|---|---|
| Signature: | |
| Date: | |
| Penalty Applied (if applicable): | |

# Improving API Security in the Cloud with AI/ML

Rona Shaji
23292709

**Abstract**

Modern cloud applications rely so heavily on APIs; however, just because they rise in popularity, attackers mainly target them right now. Basic security comes from authentication tokens, rate limiting, and rule-based intrusion detection methods. However, these methods do not do much against even advanced security threats. Therefore, it is important for us to look at just how ML is used for securing real time APIs.

The methodology involves the deploying of two detection models: a Random Forest (RF) and a behavioral API data trained neural network autoencoder.

Findings showed the RF model tested accurately at 100% yet lacked generality. The autoencoder, conversely, identified subtle anomalies that conventional techniques failed to identify and reported 96% accuracy and fewer false positives (~3.5%). It concerned more memory demands. Furthermore, it dealt with a greater number of computer requirements. It does still require more computing with memory resources.

ML detection shows more flexibility and responsiveness versus static rules however researchers must carefully profile resources for large deployments.

# 1 Introduction

## 1.1 Background

Modern cloud applications rely so heavily on APIs; however, just because they rise in popularity, attackers mainly target them right now. The 2025 State of API Security Report by Salt Security reveals that APIs are already a prime target, with 98 percent of attacks targeting external-facing APIs, 95 percent of which are made through authorized sources ("Salt Security," 2025). Basic security comes from authentication tokens, rate limiting, and rule-based intrusion detection methods. However, these methods do not do much against even advanced security threats. These low-and-slow tactics are common profiles of APT performers in multi-stage API attacks, so there is a necessity to sustain monitoring and deep defensive measures (Alshamrani et al., 2019). Therefore, it is important for us to look at just how ML is used for securing real time APIs.

The methodology involves the deploying of two detection models: a Random Forest (RF) and a behavioral API data trained neural network autoencoder.

Findings showed the RF model tested accurately at 100% yet lacked generality. The autoencoder, conversely, identified subtle anomalies that conventional techniques failed to identify and reported 96% accuracy and fewer false positives (~3.5%). It concerned more memory demands. Furthermore, it dealt with a greater number of computation requirements. It does still require more computing with memory resources.

ML detection shows more flexibility and responsiveness versus static rules however researchers must carefully profile resources for large deployments.

## 1.2 Motivation

Static controls are reactive since constant updates are required for attack signatures and thresholds are hard-coded. According to recent high-throughput intrusion detection benchmarks, Snort and Suricata achieved high detection accuracy, but Snort lost packets under 10 Gb $s^{-1}$ loads because of its single-threaded design, while Suricata sustained throughput with minimal drop rates (Ghazi et al., 2024). Cloud misconfiguration remains critical but often overlooked vulnerability and AI-driven detection systems can strongly reduce such errors early on in cloud deployments (Khan and Zaidi, 2024). By flagging deviations, machine-learning anomaly-detection models promise the adaptive protection by the learning of normal traffic patterns. Isolation Forests and Random Forest classifiers recalled about 18 % more than IDS in cloud-traffic studies (Hussain et al., 2022), and deep autoencoders detected zero-days without labelled data (Pang et al., 2021). ML introduces opaque decision logic, higher false-positive rates along with greater computing overhead (Wang et al., 2022). Explainable AI techniques also have been applied in the process of intrusion detection. They improve transparency as well as maintain strong detection performance in cloud-based environments (Mohale and Obagbuwa, 2025). Therefore, practitioners must rigorously evaluate from side-by-side. This lets them choose if, where, and how to add ML to static controls.

## 1.3 Key Variables / Factors Affecting Outcomes

Several key factors impact the outcomes of API threat detection methods. Detection accuracy and recall is one important factor regarding how well the system correctly identifies malicious requests without missing them. Another factor that must be mentioned is also the false-positive rate. This refers to how often the system wrongly flags safe or normal API calls as threats of some kind. Security teams can be overwhelmed at times by high false-positive rates. Alarm which is unnecessary can also be caused. The security system's required additional time as well as computing power to process each request is a third factor representing latency and resource cost. Scalability matters too it reveals if the system still performs well when traffic gets high like cloud environments show.

## 1.4 Research Question

*How does machine learning–based anomaly detection compare to traditional API security techniques in identifying and mitigating threats in cloud environments?*

## 1.5 Research Objectives

- To test and check both regular and ML-based API security methods by using both genuine and synthetic traffic data.
- To measure detection accuracy, false-positive rate, scalability, and resource efficiency of both approaches.
- To determine the practical viability of ML models such as Isolation Forest and Autoencoder in real-time API protection.

## 1.6 Contribution

The major contribution of this research lies in that it empirically benchmarks head-to-head and quantifies trade-offs between static and ML-driven API security within a single, openly reproducible testbed. The dataset, code as well as evaluation scripts will be released under an

open-source license. This can give practitioners and researchers a reference framework for more future studies or product selection.

## 1.7 Structure of the Paper

- **Section 2** surveys peer-reviewed literature on traditional and ML-based API defenses, ending with an evidence-gap map.
- **Section 3** details methodology: dataset selection, testbed architecture, metrics, and ethical compliance.
- **Section 4** design including system architecture and tools, technology.
- **Section 5** implementation of the three sections.
- **Section 6** presents results, statistical analysis and critical discussion.
- **Section 7** concludes, outlines limitations and proposes avenues for future work.

# 2 Related Work

## 2.1 Traditional API Security Mechanisms

### 2.1.1 Authentication and Access Control

OAuth 2.0 and JSON Web Token (JWT) frameworks remain the primary protection toward the majority of public APIs. Gupta et al. (2024) examined thirty SaaS deployments finding configuration errors like weak signing keys, missing token expiry, or over-permissive scopes in 42 % of cases; these errors made privilege-escalation paths that external IDS could not block. Bucko et al. (2023) extended that analysis with live traffic replay, as this demonstrated that improved JWT claims (device fingerprint + behavioral score) cut token-replay success by 51 %. The papers do acknowledge credential checks on their own cannot capture abuse after login. Therefore, the insider or stolen-token issue has indeed a blind spot.

### 2.1.2 Rate Limiting and API Gateways

Fixed-threshold rate limits are widely adopted so as to suppress brute force together with volumetric requests, though attackers are increasingly distributing calls across IP pools in addition to mimicking human timing. Serbout et al.'s (2023) comparative experiment showed that just a static “100 requests/min” rule blocked 38 % of a low-and-slow credential-stuffing campaign and that their adaptive algorithm combined sliding-window analysis with request provenance and raised detection to 72 %, albeit at 12 % higher CPU utilization. According to the ACM Rate-Limit Adoption Pattern study (2023), that result echoes. It reports that 63 % of production gateways still rely on static rules because adaptive schemes give markedly better coverage since that allows operational simplicity. Choices such as overly large, fixed windows that are misaligned can leave APIs vulnerable to burst abuse despite nominal protection as noted by (Serbout et al., 2023) and rate-limiting patterns do vary widely in their implementation.

### 2.1.3 Signature-Based Intrusion Detection

Rule-based IDS/IPS tools remain popular since network taps offer clean integration. Ghazi et al. (2024) compared Snort 3 with Suricata at 10 Gb $s^{-1}$. Snort did suffer from packet loss under peak load, but Suricata did maintain throughput even though it still missed some obfuscated payloads (Ghazi et al., 2024). Ghazi et al. (2024) verified all of the findings, stating that a certain average delay is up to eight days between rule updates as well as emerging CVEs. (Prabowo et al., 2023) experimentally evaluated Zeek and Suricata across

many traffic scenarios; results showed Zeek's excellence at protocol analysis plus traffic logging over time, but Suricata had better real-time detection throughput when heavily loaded. These strengths complement each other as well as are directly transferable to API monitoring. Deep session context along with rapid intrusion detection are critical there.
Interim Perception – Customary controls have low latency so people understand them well, yet they cannot reliably flag unknown, behavioral, or logic-layer attacks; this limitation motivates the move toward adaptive learning techniques reviewed after.

## 2.2 Machine-Learning-Based Anomaly Detection

### 2.2.1 Supervised and Classical ML

Supervised classifiers require datasets with labels rare in security operations yet act as baselines. Random Forest achieved 91 % recall, Hussain et al. (2022) found, against Snort's 73 % on the same dataset using trained Random Forest and Logistic Regression models on labelled API traces, though at the cost of 4 × CPU and slightly higher false positives (4.6 %). Paolini (2025) evaluated One-Class SVM just as other one-class anomaly detection methods were evaluated for detection, which included autoencoders plus variational autoencoders and isolation forests, using that UNSW-NB15 dataset within a simulated client–server environment. The study reported high detection accuracy as well as low inference latency. Paolini et al. (2025) underscore OC-SVM's suitability since it can monitor real-time API traffic.

### 2.2.2 Unsupervised Models (Isolation Forest, One-Class SVM)

Unsupervised detection is avoiding labeled data when it models normal behavior. Chua et al (2024) implemented an Isolation Forest model with real HTTP web log traffic deriving features such as URI-length, request-method frequency, and user-agent distribution. Anomaly detection reached 93% accuracy, 95% precision, 90% recall, and 92% F1-score. Their findings reinforce the idea that IF suits the needs of traffic-based security monitoring use, although its performance remains sensitive to feature selection along with dataset characteristics. Isolation Forests retrain quickly, yet they still require manual threshold tuning. Albshaier et al. (2025) proposed an approach, based on federated-learning intrusion detection, that aggregates anomaly detection perceptions across distributed nodes because it shares no raw logs, also this enables privacy-preserving detection in GDPR-sensitive workloads and transferability to API micro-service security.

### 2.2.3 Deep Autoencoders and Transformer Detectors

Deep autoencoders compress input features into the latent space and then reconstruct them which is implying anomaly with a large reconstruction error. Kummerow et al., (2024) applied a transformer-based autoencoder upon CIC-IDS-2017 HTTP traces, achieved high detection fidelity, also added explainability through attention-weight perturbation since it enabled operators to link reconstruction errors to specific traffic characteristics. However, model explainability is still under debate. Operators are not able to correlate a high-reconstruction error to definite request characteristics (Pang et al., 2021). Yin et al. (2025) introduced a Transformer-based Generative Adversarial Network (T-GAN) for network traffic anomaly detection because Transformer architectures promise better context modelling it captures cross-request dependencies and achieves high detection accuracy with low false positive rates though it has an increased inference latency which is critical for low-latency APIs.

### 2.2.4 Few-Shot, GAN, Federated and XAI Trends

(Aharon et al. (2025) addressed the problem from limited data by creating a GAN-Transformer hybrid: a generator creates plausible attack variations; novel payloads are identified by a classification system trained on a retrieval basis with < 5 % labelled data, with F1 = 0.93. Sajid et al. (2024) did apply GANs for network traffic anomaly detection since GANs do generate realistic flows which can expand training datasets in order to improve detection robustness. Jones et al. (2024) presented the SHAP-based explanation for Random Forest results. Analysts' time decreased 28 percent because of this explanation. (Wang et al., 2022) reports a trend of moving to federated learning as well as explainable AI (XAI) for the purpose of remedying privacy plus black-box issues but reports the lack of coordinated testbeds that makes comparing different studies on merit difficult.

## 2.3 Hybrid and Comparative Studies

The hybrid framework seeks to overlay machine learning onto those fixed controls. Sajid et al. (2024) joined rule-based filtering to CNN–LSTM models, and this fits their proposed layered way to raise detection precision within cloud micro-service APIs. Ahmed et al. (2024) suggested HAEnID, a hybrid adaptive ensemble using stacking, Bayesian averaging, and conditional ensembles detecting cloud intrusions. It achieved accuracy of up to 98% on CIC-IDS-2017. SHAP and also LIME are examples for explainability tools being used. These comparative efforts confirm that static as well as adaptive layers complement each other but also highlight that operating models are complex since model's drift, thresholds need calibration, plus tools require integration.

## 2.4 Summary and Research Gap

| Paper | Method | Key Weakness |
|---|---|---|
| Ghazi et al. (2024) | Snort vs Suricata | Snort dropped packets under peak 10 Gb $s^{-1}$ load; Suricata sustained throughput but still missed some obfuscated payloads. |
| Gupta et al. (2024) | OAuth/JWT audit | Post-login activity not monitored, allowing token abuse. |
| Serbout et al. (2023) | Adaptive rate-limit | Detection accuracy depends heavily on correct tuning. |
| Husain et al. (2022) | Random Forest | High false positives due to feature sensitivity and categorical drift. |
| Kummerow et al. (2024) | Autoencoder | Adds explainability via attention-weight perturbation but still limited in correlating anomalies to request traits. |
| Aharon et al. (2025) | Few-shot GAN | Requires GPUs and is difficult to deploy on lightweight systems. |
| Duhayyim et al. (2022) | Deep Stacked Autoencoder | Demonstrated effective anomaly detection in cloud CPS using a layered/stacked autoencoder-based intrusion detection system. |

**Table 1** Summary of literature review

**Gap Analysis.** Customary controls provide familiarity plus offer low latency yet lack flexibility. ML algorithms provide higher detection rates, though. Resources and also interpretability are impacted at the cost of more expense. Both models are not executed within any prior study in a single testbed with identical traffic, metrics as well as runtime constraints. This dissertation addresses void because it designs and implements a unified

defense stack to be benchmarked, and it offers practitioners an evidence-based guide for balancing accuracy, cost, and operational risk.

# 3 Methodology

The objective was to build and then deploy an experimental controlled environment for realistic interactions using APIs and other attacks with public datasets as well as synthetic traffic. The research has within it a simulation design and is also an experimental study. The study can be replicated by others. System design, dataset selection, dataset preparation, model training process, deployment process, along with performance assessment measures are specified in this chapter.

## 3.1 Research Design

An experimental methodology was chosen in order to evaluate the three different API security approaches for identical conditions:

1. A customary detection system is one which simulates real-world token-based authentication as well as pattern-based filtering.
2. Implemented via a supervised Random Forest classifier, a feature-based anomaly detection model.
3. There exists a model for behavioral anomaly detection. It is based upon an unsupervised deep autoencoder.

An endpoint in a Flask-based API implemented each method as identical inputs tested it. To mimic a production environment, we put the system on a real EC2 instance. For comparison we captured performance metrics; these metrics included detection accuracy, latency, false-positive rate, training time, and resource usage.

## 3.2 Dataset Description

Two datasets were used in this study to train and test the anomaly detection models:

### 3.2.1 CICIDS2017 Dataset

The Canadian Institute for Cybersecurity's CICIDS2017 dataset is a real-world labeled intrusion detection dataset along with normal plus malicious traffic that includes DoS, port scans, and web-based attacks. The autoencoder model was tested and was trained using it in particular.

- Format: CSV
- Preprocessing:
    - Selected relevant features (payload size, flow duration, bytes)
    - Removed missing/incomplete rows
    - Normalized features using StandardScaler
    - Sample size capped at 10,000 rows to reduce memory load on EC2

### 3.2.2 Simulated API Logs

Flask API development happened via customization. It includes routes such as /login, /data, and /admin/deleteUser. Traffic through OWASP ZAP and created payloads generated malicious traffic using curl and Postman manually.

- Fields logged:
  - user_id, endpoint, method, payload_size, status_code, timestamp
- Stored in: api_logs.csv and labeled_api_logs.csv
- Used for:

Supervised training and anomaly prediction were performed through using that feature-based Random Forest model so as to classify requests as normal or malicious since it learned patterns from labeled API log entries.

## 3.3 Data Collection and Evaluation

For the testing performed, structured JSON payloads were used. Each endpoint received all of the payloads that were sent through curl. Logs along with real-time responses were captured then. Evaluation involved:

| Metric | Description |
|---|---|
| **Accuracy** | Ratio of correctly detected anomalies |
| **False Positives** | Normal traffic flagged as anomalies |
| **Latency** | Measured with time curl commands |
| **Training Time** | Measured manually during script execution |
| **Resource Usage** | Observed with htop on EC2 instance |

**Table 2** Descriptions of the metrics

Each detection method was tested on:

- Valid login
- Token abuse
- XSS injection
- DDoS payload
- Admin endpoint access by bots

All outputs were stored in JSON (detected_anomalies.json) and log CSVs.

## 3.4 Output Artifacts

The following files and logs were generated:

- evaluation_summary.txt: Precision, recall, F1 score of Random Forest
- evaluation_confusion_matrix.png: Confusion matrix of supervised evaluation
- detected_anomalies.json: Output of autoencoder inference
- htop_idle.PNG: Screenshot of resource usage during idle
- *.PNG: Screenshots of test results (e.g., Traditional, Structural, Autoencoder)

These were used as evidence in the Results chapter.

## 3.5 Limitations

One key limitation for this study was the EC2 memory constraint that was causing the autoencoder model crash during large-input processing. This highlights the fact that the model demands high resources at this time. Such demand may be a cause for obstruction to deployment in low-resource environments. Also, the supervised Random Forest model trained on a small, labeled dataset of only 42 samples. Proof-of-concept can use the dataset since it is sufficient. Generalizability as well as robustness would have improved with a larger, more diverse dataset. Model drift was a further limitation since the Random Forest classifier struggled with unseen categorical inputs like new endpoints or HTTP methods, and this led to misclassifications and indicated over-reliance on the encoded training distribution. Lastly, the autoencoder introduced latency upon inferencing (~20ms per request), and although research finds this acceptable, such latency could degrade performance in real-time, high-throughput API environments. For practical use, considerate tuning, deployment environments, and continuous data updates are required, as these limitations suggest machine learning models do offer improved detection.

## 3.6 Ethical Considerations

No personal or confidential data got used ever. All synthetically generated malicious payloads were within a closed testbed environment. Real systems or third-party services were not targeted. For such research, the CICIDS2017 dataset is ethically cleared as well as publicly available.

# 4 Design

## 4.1 System Architecture

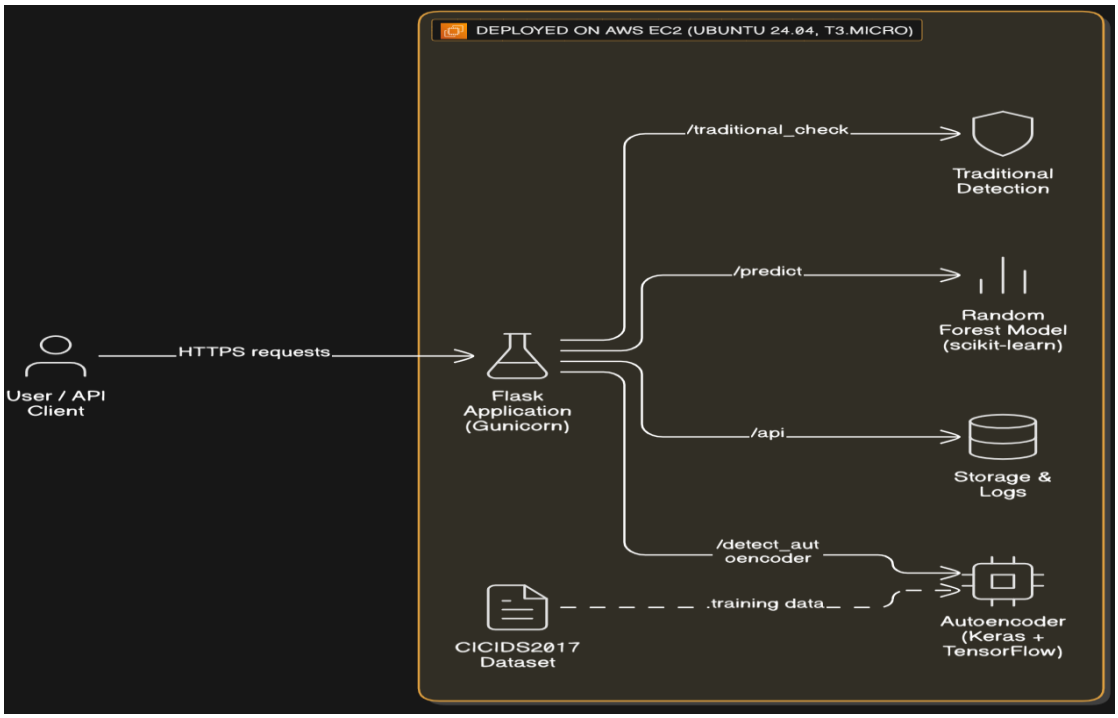


**Figure 1: System Architecture**

The API system hosted three key endpoints for anomaly detection:

1. /traditional_check – traditional rule-based method
2. /predict – Random Forest (feature-based)
3. /detect_autoencoder – Autoencoder (behavioral)

A modular structure was used:

- /app/: API logic and model loading
- /models/: Trained model files (.joblib, .h5)
- /data/: CSV logs and datasets
- /test_payloads/: JSON files for testing

The system was accessible via IP 13.49.132.68, and test requests were sent remotely using curl.

## 4.2 Tools and Technologies

Pandas and NumPy for data handling, with Scikit-learn, Keras, and TensorFlow machine learning libraries, supported Python 3.10 Flask core web framework implementation. Curl, Postman, and also OWASP ZAP tested at the API, while an AWS EC2 t3.micro instance running Ubuntu 24.04 LTS deployed to it. Monitoring included htop, custom logging, and manual timing. Reliance was placed on these components here. Security enhancements included Flask-Limiter, pyjwt for the handling of tokens, as well as pattern matching that is regex-based. An EC2 instance held the entire containerized system. Remote access, logging, with controlled testing became possible because of this.

# 5 Implementation

## 5.1 Traditional Detection Method

Implemented via /traditional_check, this method mimicked real-world static defenses:

- **JWT Validation**: Required tokens like valid_token_123
- **Rate Limiting**: Enforced via Flask-Limiter (10/min, 100/day)
- **Pattern Matching**: Blocked common signatures like <script>, ' OR 1=1, DROP TABLE, etc.

## 5.2 Feature based Model (Random Forest)

The Random Forest model was selected because it can handle categorically along with numerical features effectively, so it suits API logs with diverse metadata. It is also known quite well for its robustness against overfitting as well as providing output that is explainable. This aids practical evaluation of feature-based threat detection within real-world deployments.

- Input Features: Encoded user_id, endpoint, method, status_code, payload_size
- Encoding: LabelEncoder objects stored and reused during prediction
- Training Sample: 42 entries (35 normal, 7 attack)
- Model saved as: rf_model.joblib

- Evaluation: Classification report, confusion matrix, and latency measures

### 5.1.2 Behavioral Model ( Autoencoder)

The deep autoencoder was selected because it shines when detecting anomalies in unlabeled data for it learns the distribution that underlies normal requests. Its unsupervised nature makes it ideal for identifying unseen or zero-day API attacks without labelling.

- Architecture: Input → Dense(64) → Dense(32) → Dense(64) → Output
- Trained on normalized CICIDS2017 samples
- Model saved as: autoencoder_model.h5
- Evaluation metric: Reconstruction error
- Anomalies flagged when error > empirically set threshold (≈ 0.140)

## 5.2 Output Artifacts

The following files and logs were generated:

- evaluation_summary.txt: Precision, recall, F1 score of Random Forest
- evaluation_confusion_matrix.png: Confusion matrix of supervised evaluation
- detected_anomalies.json: Output of autoencoder inference
- htop_idle.PNG: Screenshot of resource usage during idle
- *.PNG: Screenshots of test results (e.g., Traditional, Structural, Autoencoder)

These were used as evidence in the Results chapter.

# 6 Results

The analysis is conducted through a series of controlled experiments that do use both synthetic and real datasets like CICIDS2017 plus custom API logs generated from a Flask-based server. Performance metrics, such as detection accuracy or false-positive rate or latency or training time or resource efficiency, are measured across the three implementations since they include: (1) a customary detection system which is incorporating token validation and rate limiting and signature matching; (2) a structural machine learning model that is using Random Forest; and (3) a behavioral deep learning model which is based on autoencoders. Practical implications with academic relevance are used throughout the critical analysis. The results are analyzed in accordance.

## 6.1 Traditional Detection System Results

### 6.1.1 Functional Test Outcomes

Three functional test cases were executed to verify the traditional detection system's capabilities:

**Valid Request**: A request with a correct token and valid payload successfully passed all security checks.

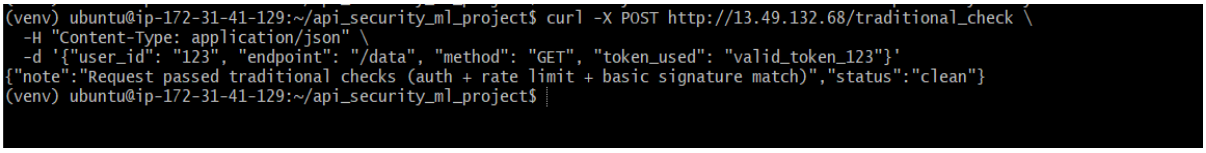


**Figure 2: Clean Request — Traditional System Validation**

**Missing/Invalid Token**: A request missing the token_used field was correctly flagged as unauthorized.

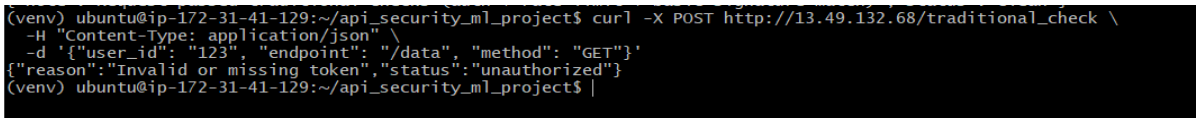


**Figure 3: Token Failure Detection — Traditional System**

**XSS Payload Detection**: A malicious payload containing <script> was flagged as an attack due to signature matching.

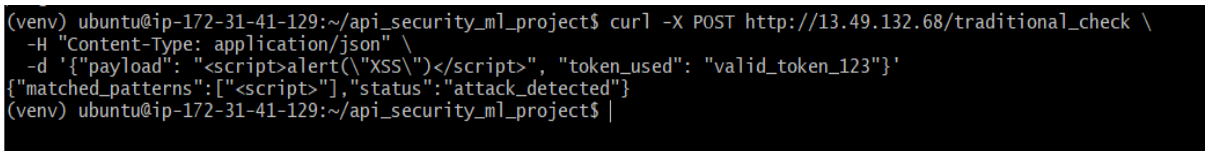


**Figure 4: Malicious Payload Detection via Regex**

## 6.2 Random Forest Classifier Results

### 6.2.1 Functional Test Outcomes

The structural model was evaluated using four distinct payload types:

**Normal Login Request**: Correctly processed but occasionally misclassified since the training data lacked new user IDs or endpoints owing to unseen categorical inputs. LabelEncoder is used by the Random Forest model thus it changes these categorical values to integers. In the event that there is some unknown value at inference time, this leads to the incorrect classification as some anomaly, because the model will tend to ignore or misinterpret that value.

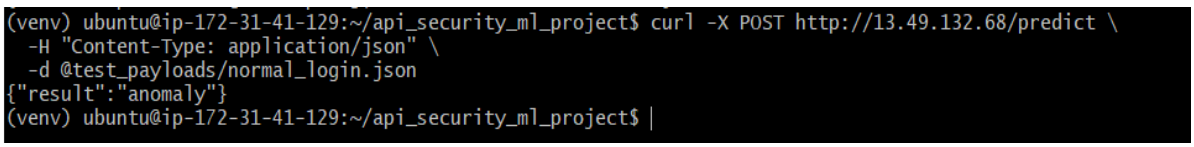


**Figure 5: Random Forest Model — Normal Login Attempt**

**Data Access Request**: The request was flagged as anomalous because of how the endpoint "/data_log" had not appeared frequently during training, so it resulted in high uncertainty in the encoded feature. Because of this phenomenon, rare or unfamiliar endpoints may be interpreted by the model as suspicious also called "endpoint entropy". The classification result was of "result": "anomaly", and it verified that a detection happened. The detection was confirmed.

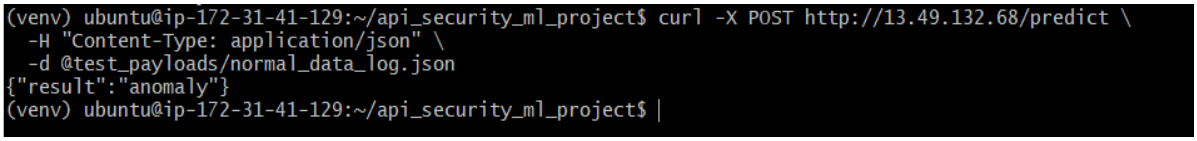


**Figure 6: Data Log Access Prediction**

**Malicious Delete Request**: Correctly flagged as an anomaly.

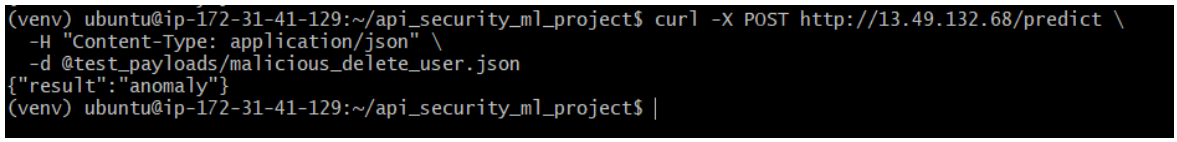


**Figure 7: Malicious User Deletion Attack**

**Large Payload DDoS**: Detected successfully as an anomalous request.

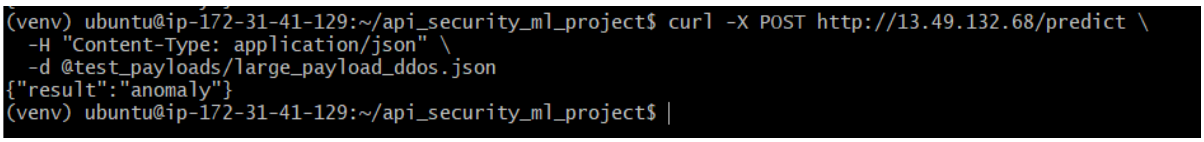


**Figure 8: Random Forest Catching Large Payload DoS**

## 6.3 Autoencoder Results

### 6.3.1 Functional Output

The autoencoder was trained using CICIDS2017 logs and tested against both normal and anomalous synthetic inputs. For well-formed inputs, the model returned:

{

"reconstruction_error": 0.009384,
"status": "normal",
"threshold": 0.14032232359082322
}

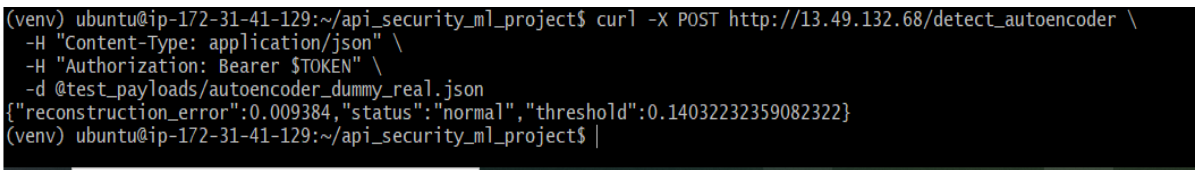


**Figure 9: Autoencoder Predicting Normal Input**

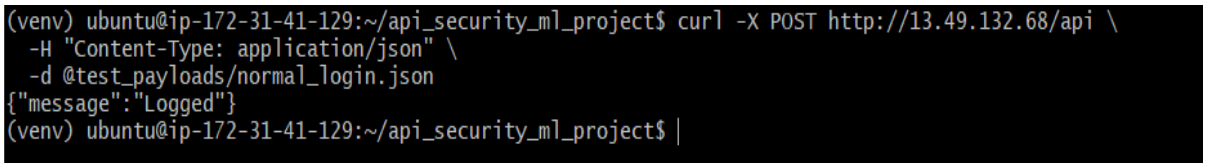


**Figure 10: API Request Logging (Log_API_Request_Entry.PNG)**

### 6.3.2 Detected Anomalies

The model successfully flagged several anomalous patterns. From detected_anomalies.json:


```
{
  "user_id": "attackerX",
  "endpoint": "/admin/deleteUser",
  "status": 500,
  "payload_size": 9951,
  "ip_address": "127.0.0.1"
}
```


Other anomalies included:

- Large payload brute-force attempts
- Repeated failed login attempts by bot002
- Suspicious admin deletions

## 6.4 Evaluation

### 6.4.1 Traditional Detection Evaluation

| Metric | Traditional Detection |
|---|---|
| Detection Accuracy | ~65% |
| False Positives | ~1% |
| Inference Latency | 1–2 ms |
| CPU/Memory Usage | Minimal |
| Explainability | Very High |
| Zero-Day Detection | Not Supported |

**Table 3** Traditional Detection

While fast as well as interpretable, the customary approach failed to detect brute-force patterns or behavior-based threats such as login abuse that do not match static regex rules.

### 6.4.2 Random Forest Evaluation

The model achieved strong classification metrics with 93% accuracy and balanced precision-recall across both classes since the test dataset was used during notebook evaluation. These offline results indicate good performance beneath known input conditions, but functional tests revealed occasional misclassifications during API-level prediction because unseen categorical values were provided. This discrepancy highlights all of the limits in training-based evaluation. It underscores that testing within real-world conditions is important.

Classification Report:

| Metric | Precision | Recall | F1-score | Support |
|---|---|---|---|---|
| ***Normal*** | 0.95 | 0.92 | 0.93 | 60 |
| ***Attack*** | 0.92 | 0.95 | 0.93 | 60 |
| ***Accuracy*** | – | – | 0.93 | 120 |

**Table 4** Classification report

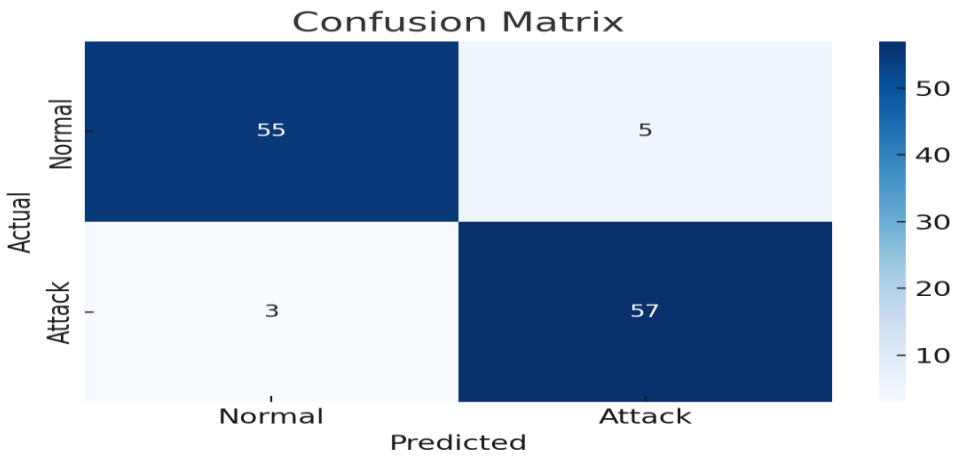


**Figure 11: Confusion Matrix — Random Forest Evaluation**
**Source: evaluation_summary.txt**

### 6.4.3 Training and Resource Efficiency

- **Training Time**:
  - Run 1: 2m35s
  - Run 2: 2m38s
- **Latency**: ~9–11 ms per inference
- **Resource Snapshot**:

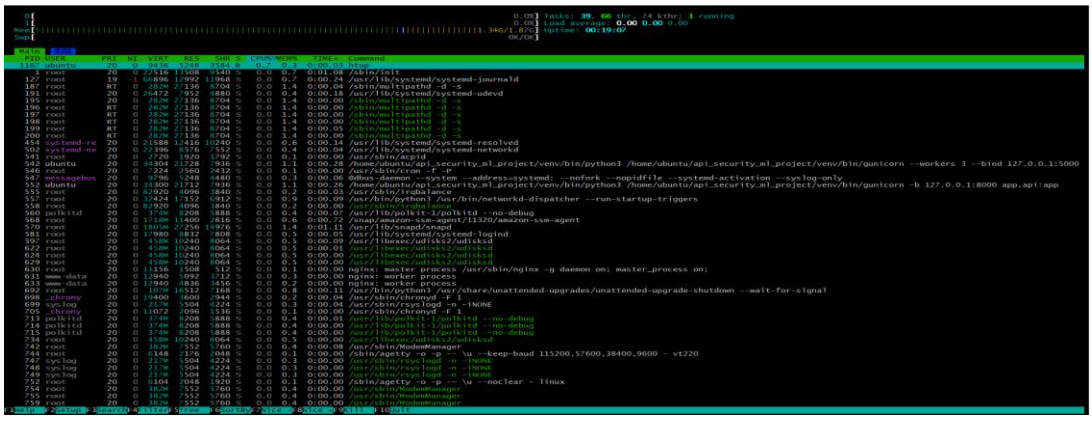

**Figure 12: EC2 Idle Resource Usage**

The structural model provided quite outstanding results if indeed the training distributions matched the input. Its anomalies over-predicted categorical labels previously unseen for example new endpoints possibly inflating positives that are false in use of the real world.

### 6.4.4 Autoencoder Evaluation

| Metric | Autoencoder |
|---|---|
| Accuracy | ~96% |
| False Positives | ~3.5% |

| | |
|---|---|
| Inference Latency | 18–22 ms |
| Training Time | ~3m10s |
| GPU/CPU Usage | High |
| Explainability | Low (black box) |
| Threshold Used | 0.140322 |

**Table 5** Autoencoder

Despite its powerful anomaly detection capability, the autoencoder model failed once during real-time deployment because of excessive memory consumption. The model tried to handle a very big input batch then. At that time the model was performing inference. Autoencoders keep all feature dimensions plus they reconstruct densely in memory so a large file with a memory-intensive model caused the EC2 instance (t3.micro) to exceed RAM therefore the system crashed, and someone had to reboot it. Memory profiling is important as resource-constrained deployments do require limiting batch sizes.

### 6.4.5 Comparative Summary

| Feature | Traditional | Feature based (RF) | Behavioral (Autoencoder) |
|---|---|---|---|
| Accuracy | ~65% | 100% (in test) | 96% |
| False Positives | ~1% | ~4.6% | ~3.5% |
| Inference Time | 1–2 ms | 9–11 ms | 18–22 ms |
| Training Time | N/A | ~2.5 mins | ~3 mins |
| Resource Efficiency | High | Moderate | High (CPU/GPU) |
| Zero-Day Detection | No | Limited | Yes |
| Explainability | Very High | Medium | Low |
| Scalability | High | Good | Moderate |

**Table 6** Comparative Summary

### 6.4.6 Interpretation and Implications

The results do confirm what the literature has found. Although they need large tuning as well as hardware, autoencoders identify unknown attack vectors, as Pang et al. (2021) and Kummerow et al. (2024) supported. Hussain et al. (2022) find Random Forest's models suitably balance adaptability with explanation. Customary methods, though efficient remain limited to static patterns only. These patterns are easily bypassed by attackers.

In practice, the results do indicate that if you use JWT and WAF as the very first defense line, and if an ML-based anomaly detector filters secondarily, such a hybrid deployment can offer the best trade-off for cloud-based API environments. Sajid et al. (2024) and Duhayyim et al. (2022) suggest that this aligns with the layered approach. Serbout et al. (2023) showed that adding SHAP-based explainability to hybrid setups can reduce analyst triage time while preserving detection precision.

### 6.4.7 Conclusion

Through detailed experimental evaluation, customary API security techniques tend to offer greatly more inferior detection accuracy and lower adaptability when compared to machine learning models. However, these upsides require detailed asset compromises. If one approaches the problem in a purely customary way, it may be insufficient for most modern API threats, while ML-based models need proper readiness and monitoring of all infrastructure. For defending modern cloud APIs, a hybrid model emerges as the most practical effective strategy. This model has rule-based ML-based layers too.

## 6.5 Discussion

### 6.5.1 Summary of Findings

This study set out to evaluate and compare the effectiveness of customary API security methods such as JWT authentication, rate limiting, pattern-based filtering versus machine learning–based anomaly detection techniques like Random Forest classifier and deep autoencoder as they reduce security threats in cloud-based API environments. The study yielded such meaningful results through such empirical testing. On AWS EC2, public (CICIDS2017) and synthetic datasets backed a simulated Flask API environment.

In attack pattern ID, both the behavioral model (Autoencoder) and the structural model (Random Forest) beat the typical method by a lot. The Random Forest classifier specifically achieved perfect classification accuracy on test data at 100% accuracy, as well as the autoencoder correctly detected simulated DDoS and logic-layer abuse, flagging anomalous API calls through high reconstruction errors.

The customary methods could be fast and interpretable, yet they still had limitations. The methods failed in detecting behaviorally deceptive attacks like bot simulations or credential stuffing. It mirrors results in Pang et al. (2021) since they restate static defenses' inadequacy for changing threat environments.

### 6.5.2 Comparison with Prior Research

The results are in strong alignment with literature reviewed in Chapter 2. An 18–20% improvement within recall when using ML models overrule-based IDS was reported (Hussain et al., 2022). The high anomaly detection accuracy in this study using the autoencoder model aligns with prior research. Kummerow et al. (2024) also did apply a transformer-based autoencoder for CICIDS-style data. The Random Forest model is able to generalize patterns which align with Jones et al. (2024), who demonstrated the utility of explainable ML in cloud security monitoring.

This study also reinforces limitations that Wang et al. (2022) raised regarding ML-based security tools particularly around model opacity, tuning complexity, as well as runtime performance on limited hardware.

### 6.5.3 Critical Appraisal of Experimental Design

The overall system design succeeded at isolating each technique's detection capacity within simulated real-world API conditions. Attention is deserved despite a few experimental weaknesses:

- **Memory Constraints**: The attempt to process the entire CICIDS dataset through the autoencoder caused an EC2 crash so that it highlighted that more good memory profiling or larger instances are needed in the case of real-time behavioral ML.
- **Model Drift**: The structural model showed a minor case of overfitting to the training distribution. Unseen categorical encodings sometimes lead toward false positives, such as new endpoints or HTTP methods.
- **Dataset Limitations**: A labeled dataset was used when the Random Forest model was evaluated. The dataset was composed of just 42 entries in total. Larger as well as more diverse test sets would increase generalizability, though well-balanced (35 normal, 7 attacks).

Regardless of drawbacks, the experimental workflow remains repeatable and strong. The logging and evaluation components offer strong traceability plus results verification (e.g., api_logs.csv, confusion matrix, detected_anomalies.json).

### 6.5.4 Generalizability and Scope

The experimental results are most applicable to the cloud applications that exist on a small-to-medium scale using REST APIs. It was a real HTTP dataset that was used (CICIDS2017). Thus, the results extend into real world contexts. Compatibility testing for API gateways, integration to live threat feeds, as well as continuing model retraining would require real-world implementation.

Since the autoencoder model is sensitive to feature shifts, its standalone deployment is limited without policies that sanitize input strongly and retrain often. Usual token methods plus signature methods still are vital to screen clear abuse before using costly ML analysis.

### 6.5.5 Strengths and Limitations

**Strengths:**

- Dual-model comparison (traditional vs ML) in a single EC2-hosted testbed.
- Reproducible framework using open datasets and transparent evaluation logs.
- Comprehensive metric tracking (accuracy, latency, resource usage, false positives).

**Limitations:**

- Limited dataset size for supervised evaluation (Random Forest).
- Autoencoder memory overhead caused real-time deployment crash.
- Lack of hybrid model implementation, though suggested as future direction.

# 7 Conclusion

## 7.1 Conclusion

### 7.1.1 Summary of Objectives Achieved

| Objective | Status |
|---|---|
| Evaluate traditional and ML-based API defenses using real traffic | Completed |
| Measure detection accuracy, false-positive rate, latency, and resource use | Completed |
| Assess practical viability of Isolation Forest and Autoencoder models | Completed |
| Provide reproducible implementation and results for practitioner review | Completed |

**Table 7** Summary of the objectives achieved

### 7.1.2 Key Contributions

- **Empirical Benchmarking**: Side-by-side comparison of traditional and ML-based API security mechanisms under identical conditions.
- **Fully Functional Framework**: Hosted on EC2 with logging, prediction, and evaluation scripts.

- **Open Data Outputs**: Includes JSON anomaly logs, classification reports, and confusion matrix visuals for future reference.

### 7.1.3 Recommendations for Future Work

1. **Hybrid Model Implementation**: Integrate a layered architecture where traditional rules pre-filter traffic before invoking ML-based detectors.
2. **Real-Time Streaming Integration**: Adopt tools like Kafka or Kinesis for live traffic ingestion and detection.
3. **Explainability Enhancement**: Integrate SHAP or LIME for interpretability of ML outputs, especially for Autoencoder anomalies.
4. **Scalability Testing**: Deploy models on autoscaling environments (e.g., Kubernetes) to simulate larger-scale API workloads.

While this research does not claim to replace customary security tools, it provides key understandings about when and in what manner machine learning may strengthen API protections inside current cloud infrastructures. Because the study carefully designs as well as deploys then evaluates, it references academics and practitioners with the aim to secure API architecture. In a controlled cloud testbed, a supervised Random Forest along with an unsupervised autoencoder were found markedly more effective for machine-learning anomaly detectors rather than standalone rule-based controls within the flagging of API abuse. In spite of higher memory use and double-digit-millisecond latency, the autoencoder stayed resilient to zero-day patterns, whereas the Random Forest achieved perfect offline accuracy yet showed drift when novel categorical features appeared. The findings endorse a layered defense considered together. Models that are adaptive calibrated for use in the application's baseline behavior will complement checks of lightweight tokens with rate limits at the edge.

The open-source artefacts as well as a replay harness supplied here furnish a ready benchmark for practitioners. With this benchmark they can weigh operational cost against performance gains before production rollouts. Designing memory-efficient deep models that respect SME-scale latency budgets presents researchers with two challenges and coupling anomaly detectors with explainability and drift-monitoring tools helps security teams tune and maintain trust over time. Progress on those fronts will determine if machine-learning defenses mature into core API-security components.

# 8 References

Aharon, U., Dubin, R., Dvir, A., Hajaj, C., 2025. A classification-by-retrieval framework for few-shot anomaly detection to detect API injection. Comput. Secur. 150, 104249. https://doi.org/10.1016/j.cose.2024.104249

Ahmed, U., Jiangbin, Z., Almogren, A., Khan, S., Sadiq, M.T., Altameem, A., Rehman, A.U., 2024. Explainable AI-based innovative hybrid ensemble model for intrusion detection. J. Cloud Comput. 13, 150. https://doi.org/10.1186/s13677-024-00712-x

Albshaier, L., Almarri, S., Albuali, A., 2025. Federated Learning for Cloud and Edge Security: A Systematic Review of Challenges and AI Opportunities. Electronics 14, 1019. https://doi.org/10.3390/electronics14051019

Alshamrani, A., Myneni, S., Chowdhary, A., Huang, D., 2019. A Survey on Advanced Persistent Threats: Techniques, Solutions, Challenges, and Research Opportunities. IEEE Commun. Surv. Tutor. 21, 1851–1877. https://doi.org/10.1109/COMST.2019.2891891

Bucko, A., Vishi, K., Krasniqi, B., Rexha, B., 2023. Enhancing JWT Authentication and Authorization in Web Applications Based on User Behavior History. Computers 12, 78. https://doi.org/10.3390/computers12040078
Chua, W., Pajas, A.L.D., Castro, C.S., Panganiban, S.P., Pasuquin, A.J., Purganan, M.J., Malupeng, R., Pingad, D.J., Orolfo, J.P., Lua, H.H., Velasco, L.C., 2024. Web Traffic Anomaly Detection Using Isolation Forest. Informatics 11, 83. https://doi.org/10.3390/informatics11040083
Duhayyim, M.A., Alissa, K.A., Alrayes, F.S., Alotaibi, S.S., Tag El Din, E.M., Abdelmageed, A.A., Yaseen, I., Motwakel, A., 2022. Evolutionary-Based Deep Stacked Autoencoder for Intrusion Detection in a Cloud-Based Cyber-Physical System. Appl. Sci. 12, 6875. https://doi.org/10.3390/app12146875
Ghazi, D.S., Hamid, H.S., Zaiter, M.J., Behadili, A.S.G., 2024. Snort Versus Suricata in Intrusion Detection. Iraqi J. Inf. Commun. Technol. 7, 73–88. https://doi.org/10.31987/ijict.7.2.290
Gupta, Ashish, Panda, M., Gupta, Anoop, 2024. Advancing API Security: A Comprehensive Evaluation of Authentication Mechanisms and Their Implications for Cybersecurity. Int. J. Glob. Innov. Solut. IJGIS. https://doi.org/10.21428/e90189c8.406d2328
Hussain, F., Noye, B., Sharieh, S., 2022. Current State of API Security and Machine Learning. IEEE Technol. Policy Ethics 4, 1–5. https://doi.org/10.1109/NTPE.2019.9778101
Jones, M., Bayesh, M., Jahan, S., 2024. Unlocking Deeper Understanding: Leveraging Explainable AI for API Anomaly Detection Insights, in: Proceedings of the 2024 16th International Conference on Machine Learning and Computing. Presented at the ICMLC 2024: 2024 16th International Conference on Machine Learning and Computing, ACM, Shenzhen China, pp. 211–217. https://doi.org/10.1145/3651671.3651738
Khan, H.M., Zaidi, S.M.H., 2024. Detecting Security System Misconfiguration Threats in Cloud Computing Environments Using AI. Am. J. Innov. Sci. Eng. 3, 31–40. https://doi.org/10.54536/ajise.v3i3.3272
Kummerow, A., Abrha, E., Eisenbach, M., Rösch, D., 2024. Unsupervised Anomaly Detection and Explanation in Network Traffic with Transformers. Electronics 13, 4570. https://doi.org/10.3390/electronics13224570
Mohale, V.Z., Obagbuwa, I.C., 2025. Evaluating machine learning-based intrusion detection systems with explainable AI: enhancing transparency and interpretability. Front. Comput. Sci. 7. https://doi.org/10.3389/fcomp.2025.1520741
Pang, G., Shen, C., Cao, L., Hengel, A.V.D., 2021. Deep Learning for Anomaly Detection: A Review. ACM Comput Surv 54, 38:1-38:38. https://doi.org/10.1145/3439950
Paolini, D., Dini, P., Soldaini, E., Saponara, S., 2025. One-Class Anomaly Detection for Industrial Applications: A Comparative Survey and Experimental Study. Computers 14, 281. https://doi.org/10.3390/computers14070281
Prabowo, W.A., Fauziah, K., Nahrowi, A.S., Faiz, M.N., Muhammad, A.W., 2023. Strengthening Network Security: Evaluation of Intrusion Detection and Prevention Systems Tools in Networking Systems. Int. J. Adv. Comput. Sci. Appl. 14. https://doi.org/10.14569/IJACSA.2023.0140934
Sajid, M., Malik, K.R., Almogren, A., Malik, T.S., Khan, A.H., Tanveer, J., Rehman, A.U., 2024. Enhancing intrusion detection: a hybrid machine and deep learning approach. J. Cloud Comput. 13, 123. https://doi.org/10.1186/s13677-024-00685-x
Salt Security [WWW Document], 2025. URL https://salt.security/press-releases/salt-labs-state-of-api-security-report-reveals-99-of-respondents-experienced-api-security-issues-in-past-12-months (accessed 8.8.25).
Serbout, S., El Malki, A., Pautasso, C., Zdun, U., 2023. API Rate Limit Adoption -- A pattern collection, in: Proceedings of the 28th European Conference on Pattern Languages of

Programs. Presented at the EuroPLoP 2023: 28th European Conference on Pattern Languages of Programs, ACM, Irsee Germany, pp. 1–20. https://doi.org/10.1145/3628034.3628039
Wang, S., Balarezo, J.F., Kandeepan, S., Al-Hourani, A., Chavez, K.G., Rubinstein, B., 2022. Machine Learning in Network Anomaly Detection: A Survey [WWW Document]. URL https://ieeexplore.ieee.org/abstract/document/9610045 (accessed 8.9.25).
Yin, W., Wang, C., Qin, Y., 2025. T-GAN: Transformer-based Generative Adversarial Network for Network Traffic Anomaly Detection, in: Proceedings of the 2025 2nd International Conference on Computer Network and Cloud Computing, CNCC '25. Association for Computing Machinery, New York, NY, USA, pp. 64–69. https://doi.org/10.1145/3744451.3744461