National College of Ireland

ENHANCING API SECURITY WITH MIDDLEWARE-BASED SECURE API GATEWAYS

MSc Research Project
MSc in Cyber Security

Abhijith Thanippilly Soman
Student ID: 23271400

School of Computing
National College of Ireland

Supervisor: Michael Pantridge

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Abhijith Thanippilly Soman
Student ID:	23271400
Program:	MSc in Cyber Security **Year:** 2024-2025
Module:	Practicum Part 2
Supervisor:	Michael Pantridge
Submission Due Date:	14/09/2025
Project Title:	ENHANCING API SECURITY WITH MIDDLEWARE-BASED SECURE API GATEWAYS
Word Count:	6600 **Page Count** 24

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:	Abhijith Thanippilly Soman
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):	

ENHANCING API SECURITY WITH MIDDLEWARE-BASED SECURE API GATEWAYS

Abhijith Thanippilly Soman

23271400

1: Introduction

1.1 Background

Application Programming Interfaces (APIs) are integral to modern software ecosystems, facilitating seamless communication between various systems and services. However, their widespread adoption has also made them prime targets for cyberattacks, leading to significant security challenges. Traditional security measures such as static rule-based firewalls, signature-based intrusion detection systems (IDS), and Web Application Firewalls (WAFs) often fall short in addressing the dynamic and sophisticated nature of API threats. The conventional security protocols are usually not adequate in balancing with the dynamic and complex nature of API attacks. According to reports by Salt Security (2022), 95% of organisations have reported API security events over the last 12 months, and malicious API traffic is up to 681%, as compared to an overall API traffic increase of 321% (Salt Security, 2022). This underscores the escalating risks associated with API vulnerabilities. Middleware solutions, such as API gateways, centralise authentication, authorisation, and traffic management of API requests, thereby enhancing API security by enforcing consistent policies and monitoring across distributed systems.

1.2 Problem Statement

APIs play a crucial role in the latest software, accommodating system communication. Nevertheless, APIs have become popular targets for cyberattacks due to their widespread use. Traditional security mechanisms often struggle to protect against emerging API threat types. Aharon et al. (2024) address this challenge by highlighting the difficulties in detecting API attacks, particularly due to the variability of datasets. They suggest using a few-shot detection

method, enhanced by machine learning and Transformer structures, to effectively detect new and evolving threats. The current study filled these gaps by incorporating machine learning techniques in middleware-based API gateways to implement real-time anomaly detection and dynamic threat mitigation to enhance the security of APIs.

1.3 Research Aim, Objectives, and Questions

1.3.1 Research Aim

The research aims to develop a middleware-based secure API gateway integrated with machine learning to enhance real-time detection and mitigation of API threats.

1.3.2 Research Objectives

- To evaluate the effectiveness of machine learning algorithms in detecting API anomalies in real-time.
- To assess the security challenges faced by APIs and the potential of middleware-based gateways in addressing them.
- To investigate how machine learning integration can improve API security beyond static rule-based approaches.
- To analyse the performance of the proposed middleware solution using evaluation metrics and suggest ways to detect anomalies.

1.3.3 Research Questions

- How can middleware-based secure API gateways be enhanced with machine learning to detect and mitigate anomalous API behaviours effectively?
- What machine learning algorithms are most suitable for real-time anomaly detection in API traffic?
- How can the integration of machine learning into API gateways improve the overall security posture of APIs?

1.4 Research Contribution

This study also adds value through the creation of a secure and middleware-based API gateway programmed with threat mitigation and anomaly detection models using machine learning. It can further the security of APIs with its concept of tackling the shortcomings of the classic

methods of security solutions by providing a scalable and changing option to secure APIs against emerging cyber threats.

1.5 Research Assumptions

The research uses the assumption that data sets on which machine learning models are trained truly represent how API traffic looks in the real world. It further assumes that the selected machine learning algorithms will be able to detect real-time anomalies based on historical API traffic data or access patterns.

1.6 Research Limitation

The study is constrained using publicly accessible Kaggle data set that might not cover all the real API traffic cases. Besides, the research concentrates on machine learning algorithms, limiting the exploration of other potential techniques.

2: Related Work

2.1 Introduction

The literature review explores existing research on API security, focusing on middleware solutions, machine learning for anomaly detection, and the integration of real-time threat mitigation. It examines the effectiveness of current methods, identifies gaps, and provides a foundation for applying machine learning-driven middleware to enhance API security.

2.2 Impact of middleware on API Security

The middleware solutions are essential to the improvement of API security services to control flows, implement policies related to API requests, and have real-time threat identification. The study by Modi, Chourasia, and Pandey (2022) attempts to plan and execute a model using the RESTful API to detect and eliminate security vulnerabilities such as unauthorised access, data breaches, injection attacks, and authentication weaknesses. They apply a top-down method, which is a systematic approach to discover the API security vulnerabilities as well as the mitigation strategies. They integrate a middleware layer for API traffic monitoring, but the shortcoming is the lack of validation with real-world datasets. Though it comes with a strong framework, it does not comprehensively deal with dynamic or changing threats. The project will combine middleware with machine learning models to maintain the adaptability and scalability of the security solution.

Conversely, Behbehani, Rajarajan, Komninos, and Al-Begain (2022) aim at identifying open banking API security threats through Bayesian attack graphs. This research employs the use of a Bayesian network to simulate attack paths, to define vulnerabilities, a good method to be applied in the open banking landscape. Middleware in this context plays a crucial role in integrating and routing API requests securely between numerous services. Nevertheless, there is a limitation to the study because it primarily looks at a class of industry, the results can be less transferable to general API ecologies. The limitation will be addressed in this research by using a more universal middleware solution to different API environments, adding real-time threat detection and machine learning integration.

Similarly, Arumugam (2025) discusses the security threat in cloud computing and reveals the unsafe API. The research paper outlines the increasing complexity of online attacks against cloud systems, providing strategies for mitigating these risks. The study suggests implementing middleware in cloud environments to enforce security policies and manage API traffic, but it lacks practical application and requires validation with real-world sample data. By incorporating a machine learning-based middleware to measure and warn against anomalous examples, the present study would fill this gap and model a dynamic, practical framework that would monitor cloud-based API activities in real-time.

In contrast, Drofa (2025) discusses integrating advanced API solutions into full-stack web and mobile applications to optimise the user experience related to improving API performance, scalability, and seamless interaction between different software components. The article provides insights into API optimisation, but it lacks a deep focus on security implications, particularly in high-risk environments. Middleware in their approach is used to manage API traffic and optimise performance, but security is not a primary focus. The research fills this gap by prioritising security within the API design, ensuring that the middleware not only improves performance but also strengthens security through machine learning-driven anomaly detection.

2.3 Middleware-Based Solutions for API Security Enhancement

Middleware-based solutions provide a robust framework for securing APIs by integrating real-time threat detection, access control, and traffic management. Thus, Samper and Ferreira (2024) explore the use of remote attestation APIs to secure image sharing in messaging apps. Their research leverages remote attestation to verify the integrity of shared images, enhancing the security of messaging applications. However, their solution is narrow in scope, focusing only on image sharing within a specific app context. This research will address broader API

security concerns by integrating middleware-based solutions with machine learning to offer dynamic, real-time protection across various API scenarios.

Conversely, Karri et al. (2025) make a comparison of microservices migration as it applies to the way of improving cloud-native applications. They work on the measurement of the benefits of migrating microservices towards improved performance and scalability of cloud-native applications. Although this offers a good understanding of how to optimise the cloud-native environments, this study does not concentrate much on the middleware and its role in API security. The research is going to be an extension of the current findings, as the middleware solutions will be implemented in the cloud-native ecosystem to ensure that performance is not done at the expense of security.

Conversely, Ali et al. (2023) introduce a generic IoT middleware to smart city applications, which aims at promoting device and system communication. Although the middleware is useful in promoting integration and communication, the restrictions include the inability to ameliorate security concerns, such as IoT and API interactions. The ongoing study will take their study further by adding machine learning to the middleware to do real-time anomaly detection and threat mitigation on IoT-driven APIs to enhance security beyond the provision of communication.

Likewise, Almadani et al. (2025) address the issue of the publish/subscribe middleware-based intelligent transportation systems (ITS) and the problem of implementing the systems. Upon investigating the uses and issues of middleware in transportation, their work also fails to pursue the opportunities in the position that middleware solutions may play a direct role in improving API security. This study will respond to this issue using middleware for API security, concentrating on real-time threat discovery and countering through the different API infrastructures to improve overall security.

2.4 Impact of Machine Learning Techniques for Real-Time Anomaly Detection in API Security

API security is improved because machine learning methods allow detecting anomalies in real-time, identifying threats, and making systems more robust overall. Thirimanne et al. (2022) introduce a deep neural network-based intrusion detection system in real-time that detects any threat in the network traffic, achieving an accuracy of 95% in detecting threats. They also show that deep learning models are effective at intrusion detection, but there is no generalisation of the model to API security. Although their approach is quite promising, it is constrained by the fact that they do not use real-world API traffic as data. The implication of this limitation will

be addressed in this research, which uses machine learning models which will focus on API traffic anomalies, to extend the scope of implementation to network-based systems.

On the contrary, Dini and Saponara (2023) specifically target processes of real-time anomaly detection in automotive cybersecurity that evaluate distinct techniques to protect automotive systems against cyber threats, reporting an 87% detection rate for anomalies in automotive systems. Their results are useful, but this study is limited by the fact that it is specifically in the automotive sector, and the results cannot be applied across the board in other contexts about API security. Their approach will be generalised to API environments in the context of the current research, and middleware solutions will be used to identify and eliminate anomalies, thus increasing the security of a wide range of use cases outside the automotive system.

Equally, Xiong et al. (2025) introduce a cloud-native container anomaly detector, Deep Container, based on deep learning, with a detection accuracy of 92% in identifying cloud container-based threats. They are resilient in terms of cloud infrastructure, but they fail to reflect upon the issues of API security. The study has limitations because it only covers the part about containers; it does not cover the interaction between APIs and machine learning. The proposed research will help in filling this gap by applying the framework of Deep Container to the security of an API, using it to detect anomalies in API traffic in real-time.

In contrast, Al-Ghuwairi et al. (2023) examine the application of machine learning to intrusion detection in a cloud computing system via time-series anomalies, reporting an 89% detection rate for cloud-related security breaches. Despite the insights that their study can bring to cloud security, it is constrained by the constraints in that it focuses on cloud infrastructures that may not necessarily apply to the security of APIs. The given research will improve their methodology through the focus on APIs, employing the time-series anomaly detection to take care of the API traffic and the safety of it, with a greater preparedness towards the API-specific challenges. Although these studies offer a significant contribution to anomaly detection, this study will generalise their work by applying them to API security, by implementing machine learning and real-time monitoring of anomalies to detect and mitigate anomalies.

2.5 Theoretical Underpinnings

For this research, the Swiss Cheese Model is selected as the theoretical underpinning to illustrate how layered security measures can address vulnerabilities in API security. The Swiss Cheese Model, as outlined by Haghani et al. (2023), provides a useful framework for understanding how layered security measures can prevent failures in complex systems, such as API security. In this model, each layer of security represents a "slice of cheese," with potential

vulnerabilities seen as holes. If these holes align across layers, a security breach may occur. Applying this to API security, middleware-based solutions, machine learning, and traditional security measures form the layers. Addressing the gaps in each layer, such as real-time anomaly detection, strengthens API resilience against evolving threats, aligning with a Vision Zero target for API security.

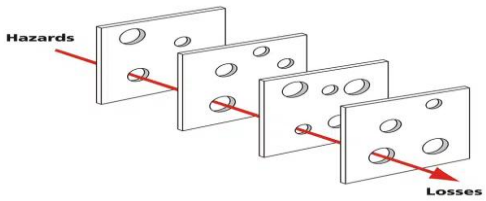


Figure 2.1: Swiss Cheese Model

(Source: Traeger, 2024)

2.6 Literature Gap

The literature review on machine learning in real-time anomaly detection in API security demonstrates some gaps. Although the current research or findings, like the one by Thirimanne et al. (2022) and Dini and Saponara (2023), provide ingredients of intrusion detection in networks and automobiles, they fail to capture the unique characteristics of API traffic and API security requirements. Also, studies such as Xiong et al. (2025) and Al-Ghuwairi et al. (2023) mostly focus on the cloud and container environment, and they do not combine machine learning approaches with the middleware-based API gateway. The proposed research will fill these gaps by using machine learning models to implement API security in detecting anomalies in real-time.

2.7 Conceptual Framework

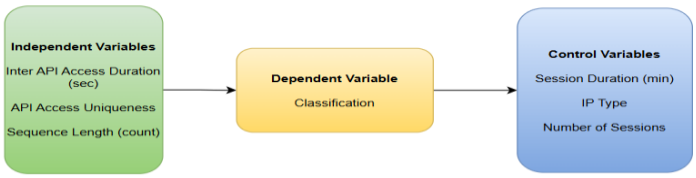


Figure 2.2: Conceptual Framework

2.8 Summary

The literature review highlights the evolving challenges in API security, emphasising the limitations of traditional methods. It discusses the potential of middleware-based solutions and machine learning techniques for real-time anomaly detection. The review identifies gaps in

existing research and positions the current study to address these shortcomings for improved API security.

3: Research Methodology

3.1 Introduction

This study intends to improve API security employing one-class SVM, hyperparameter-optimised one-class SVM, and hybrid RandomForestClassifier machine learning. It combines techniques of both unsupervised and supervised learning to detect the anomalies, and thus its methodology will be detected by using a mixture of benign and labelled data on the anomalies to maximise the performance.

3.2 Research Approach

This research has chosen the deductive research approach because it gives the researcher a chance to develop hypotheses depending on previously existing knowledge. This is because it begins with a general description or theory and evaluates it with the help of data and validates the hypothesis. Deductive reasoning works very well with studies whose aim is to prove the usefulness of the existing theory or framework (Casula, Rangarajan and Shields, 2021). The deductive research method is suitable in the given study since it entails checking the theory regarding the effectiveness of machine learning algorithms in improving API security using middleware-based gateways. It allows for the performance of structured tests of the adaptability and enhancement of known theories of API security with the aid of machine learning, which gives the suggested solution a solid basis.

3.3 Research Design

Descriptive research design has been adopted in this research because it can be used to examine more carefully and thoroughly the available phenomena relating to API security/ middleware-based secure gateways. Descriptive research is characterised by facts concerning description of the current characteristics, behaviours, and connections (Guillén-Gámez, Mayorga-Fernández and Contreras-Rosado, 2021). This framework presents a thorough picture of how API can be secured using machine learning algorithms. With its descriptive design, this study has monitored and assessed the current security threat of APIs and middleware programs. It enables collection of granular data on machine learning security threats, behaviours, and mitigation strategies, and can be of great benefit in terms of providing insights into real ways machine learning can enhance security instead of any assumptions.

3.4 Technical framework

The selected framework of the current research is the CRISP-DM (Cross-Industry Standard Process for Data Mining), as it can be defined as a structured method of data mining, which would be most suitable in integrating machine learning into API security. CRISP-DM is a popular model that can be used in predictive analytics and is applicable in solving complex issues in security using data-based solutions (Schröer, Kruse and Gómez, 2021). This framework is especially successful because it can afford iterative and dynamic procedures; thus, the data preprocessing, modelling, evaluation, and deployment phases will be at par with the research aims and objectives. CRISP-DM allows defining trends in the API traffic, the optimisation of the anomaly detection model, and improvements to the middleware solution. Its distinct steps of knowing the business, data preparation, modelling and evaluation offer a guide to using machine learning to enhance real-time security of API with the ability to adapt and revise as research is conducted.

3.5 Data Collection

The information used by this study was obtained by the API Access Behaviour Anomaly Dataset, which are obtainable on Kaggle. Such a dataset is chosen because it is relevant in the search to capture API access pattern and both benign and anomalous behaviours. It has a variety of features, i.e., inter_api_access_duration, sequence_length, and num_sessions, which play a pivotal role in analysing API security and anomaly detection (Guntur, 2021). The dataset was preprocessed to fill in any missing values, and also irrelevant columns were removed in order to analyse the critical features that deal with API access behaviour. Classification is the target variable, which has been used to transform the data into normal and anomalous behaviours, which is appropriate in the proposed machine learning models. The data has also been partitioned into training and test data so as to be able to give a proper analysis of the performance of the secure API gateway middleware integrated with machine learning. This research has been done on a solid basis given by the dataset.

3.6 Justification of Machine Learning Models

In this study, three machine learning models have been used to solve the problem of detecting anomalies in a mixed dataset: state One-Class SVM, hyperparameter-optimised One-Class SVM, and Hybrid approach with RandomForestClassifier (trained on a combined benign and labelled anomalies dataset). All models were selected due to their advantages when dealing with the characteristics of the datum, especially the ability to find anomalies in the overall unoffending set of data.

One-Class SVM

One-Class SVM model was chosen due to its capability of recognising outliers in an unsupervised way. One-Class SVM is particularly useful in anomaly detection problems, when the training data is mostly normal (benign) data as we have them in our case. It uses learning a decision function to detect an anomaly and thus determines the anomalous instances that do not fit in the learned benign (normal) data distribution. The model is optimal in cases where the core focus is to identify anomalies in the datasets in which one has positive data to work on. Training on purely benign data makes the model learn the underlying structure of normal behaviour, and thus it can effectively learn to spot abnormal behaviour that does not act in a normal way.

Hyperparameter-Tuned One-Class SVM

GridSearchCV hyperparameter tuning was used in order to improve the performance of the One-Class SVM model. The most important parameters, e.g. nu and gamma, were optimised in order to enhance the sensitivity and specificity of the model in identifying an anomaly. Tuning these parameters made the model more appropriate to optimise the detection of anomalies and restrict the false positive and false negative cases such that the overall performance of a model could detect outliers in a mixed dataset.

Hybrid Method with RandomForestClassifier

The Hybrid method, combining for training Set 70% benign data with 30% labelled anomaly data, for Test Set 30% of normal (benign) data and 80% of malicious (anomalous) data and trained using the RandomForestClassifier, offers a more robust solution by leveraging supervised learning. Unlike One-Class SVM, Random Forest can manage both benign and anomaly classes simultaneously, which allows it to better generalise when faced with both normal and abnormal instances. The Random Forest model, with its ensemble learning approach, offers high accuracy and robustness by creating multiple decision trees and aggregating their results, reducing overfitting, and improving performance on diverse data, such as our mixed dataset.

3.7 Summary

This chapter outlined the research approach, design, and framework for enhancing API security using machine learning. The deductive approach was employed to evaluate hypotheses, while descriptive design facilitated detailed analysis. CRISP-DM guided the methodology, utilising data from Kaggle for anomaly detection. Three machine learning models were justified and applied.

Chapter 4: Design Specification

4.1 Introduction

The design specification is an elaboration of the development of a threat detection system based on a middleware based on machine learning, entailing preprocessing of data, one-class support vector machine, hybrid random forest classifiers, and evaluation algorithms that would enhance security.

4.2 Design Specification of Methodological Architecture

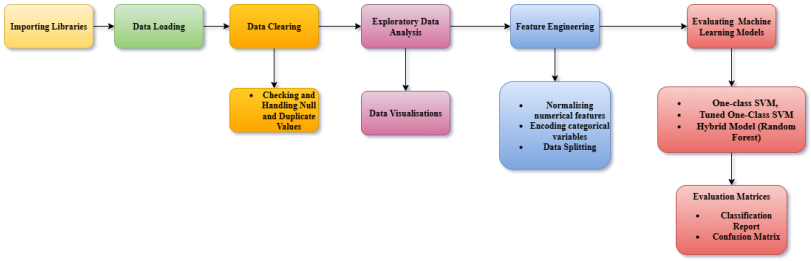


Figure 4.1: Methodological Architecture

The flowchart in Figure 4.1 summarises the most important steps in the methodology for assessing middleware-based threat detection and mitigation by employing machine learning.

- ***Importing Libraries:*** The modules that would be required in this project are pandas to manage data, numpy to manage numerical work, sklearn to manage machine learning models, matplotlib and seaborn to visualise data and warnings to block irrelevant warnings.
- ***Data Loading:*** The data loading is done through the liability of pandas by pd.read_csv(supervised_dataset.csv), and this has the data on the anomaly access behaviour. The dataset consists of such features as the duration of access, session information, and the target variable characteristic.
- ***Data Cleaning and Preparation:*** Data cleaning and preparation refer to the removal of unnecessary columns, such as Unnamed: 0, _id and then replacing missing values with dropna(). One-Hot Encoding was used to encode the categorical variables, specifically in the case, ip_type, and source. Highly correlated features num_users were dropped to eliminate redundancy and reduce unimportant features to analyse.
- ***Data Visualisations:*** The visualisations of the data were comprised of a bar plot indicating the frequency in the target variable category (normal vs. anomalous traffic). A heatmap of correlation was drawn to visualise relationships between features, indicating strongly correlated ones. These visualisations assisted in gaining insight into the dataset and features behind model training and evaluation.

- ***Data Splitting:*** In the case of the One-Class SVM, the dataset had to be partitioned such that only the benign (normal) data was used, and the model was trained to identify anomalies in normal traffic tracks. The training set was made up of the benign data, which was scaled using a standard scaler. Its own test set consisted of benign and malicious data to evaluate the consistency, and using the same test set, it evaluated the likelihood of the model to detect malicious traffic accurately, and at the same time, focus on anomaly detection.

  In the case of the Hybrid Model based on Random Forest, we partitioned the dataset between 70 % benign and 30 % malicious data to train the model. The rest of the data was divided into 30% benign and 80% anomalous data and used as a test. This split of the dataset was aimed at ensuring that the model learns the normal and abnormal traffic patterns and achieves a balanced evaluation on benign and malicious data. Then the classifier was trained, and after scaling, how much traffic is normal or anomalous. Both types of traffic were used to learn the model by using this hybrid approach, thereby enhancing generalisation and anomaly detection in the real world.
- ***Model Evaluation***: A One-Class SVM was trained on benign data to find out anomalies in API traffic. The measure was on its ability to categorise benign traffic as normal and the abnormal traffic as outliers. Its effectiveness was evaluated in terms of such metrics as precision, recall, and F1-score. The confusion matrix gave an overview of the model's strengths and weaknesses and showed that there are difficulties in reducing the false positives, which means that the model still requires improvement.

  Once hyperparameters were tuned with GridSearchCV, the performance of the One-Class SVM model was reassessed to determine whether more improvements could be achieved regarding detecting the anomalies. Some parameter changes, such as gamma, nu and kernel, resulted in improved precision, recall and false positive reduction. The ROC AUC score indicated that the model was better at classifying the different classes, hence more balanced and effective.

  The Random Forest Hybrid Method was set with the Hybrid Method, where the Random Forest classifier was trained with normal and anomalous traffic data. The comparison was done based on which the model successfully classified both types of traffic, and they were precise, recalling, and accurate. The confusion matrix represented no false positives and few false negatives, and high F1-scores.

4.3 Summary

This chapter details how to improve security with middleware security implementation that incorporates machine learning in the detection of threats. It entailed the preprocessing of data, the development of models, based on the One-Class SVM and Random Forest classifiers, the process of hyperparameter tuning, and the evaluation criteria. The hybrid method displayed better results in anomaly detection.

5: Implementation

5.1 Data Exploration and Preprocessing

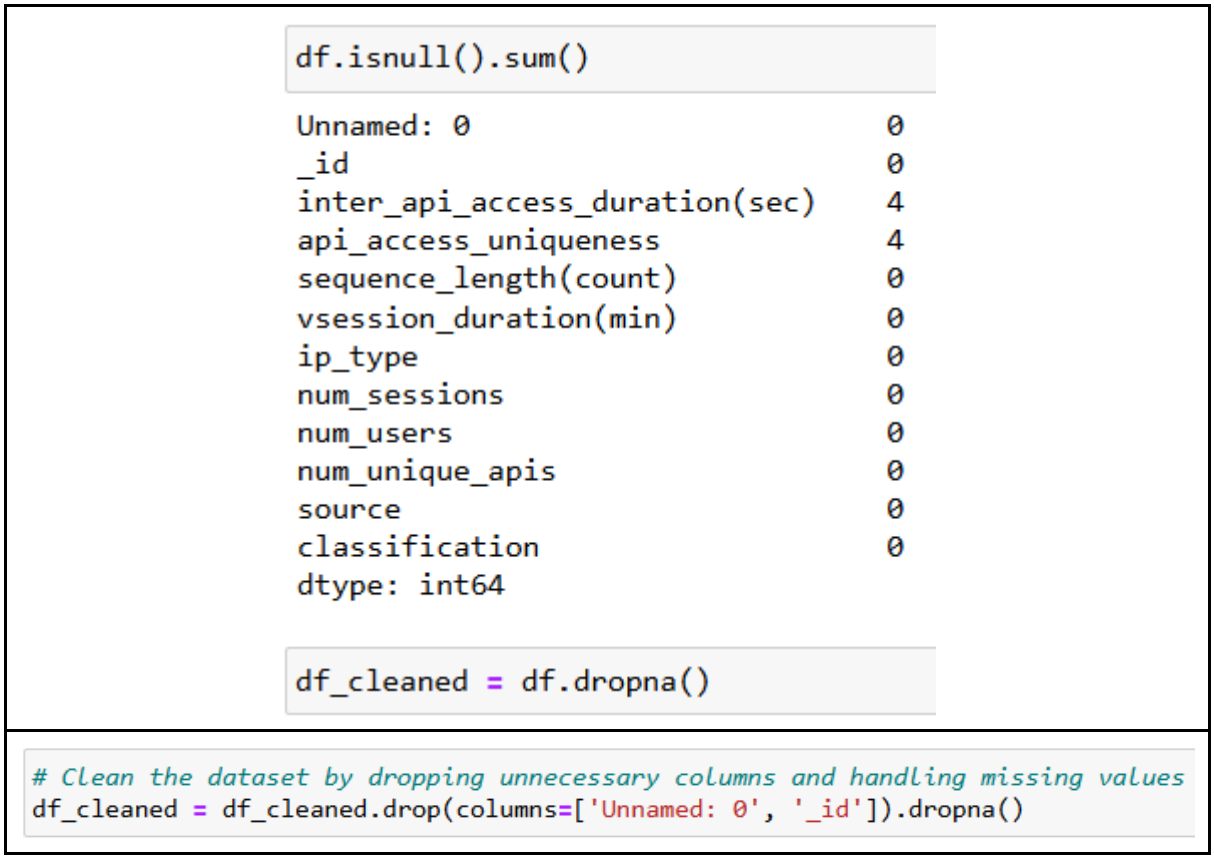
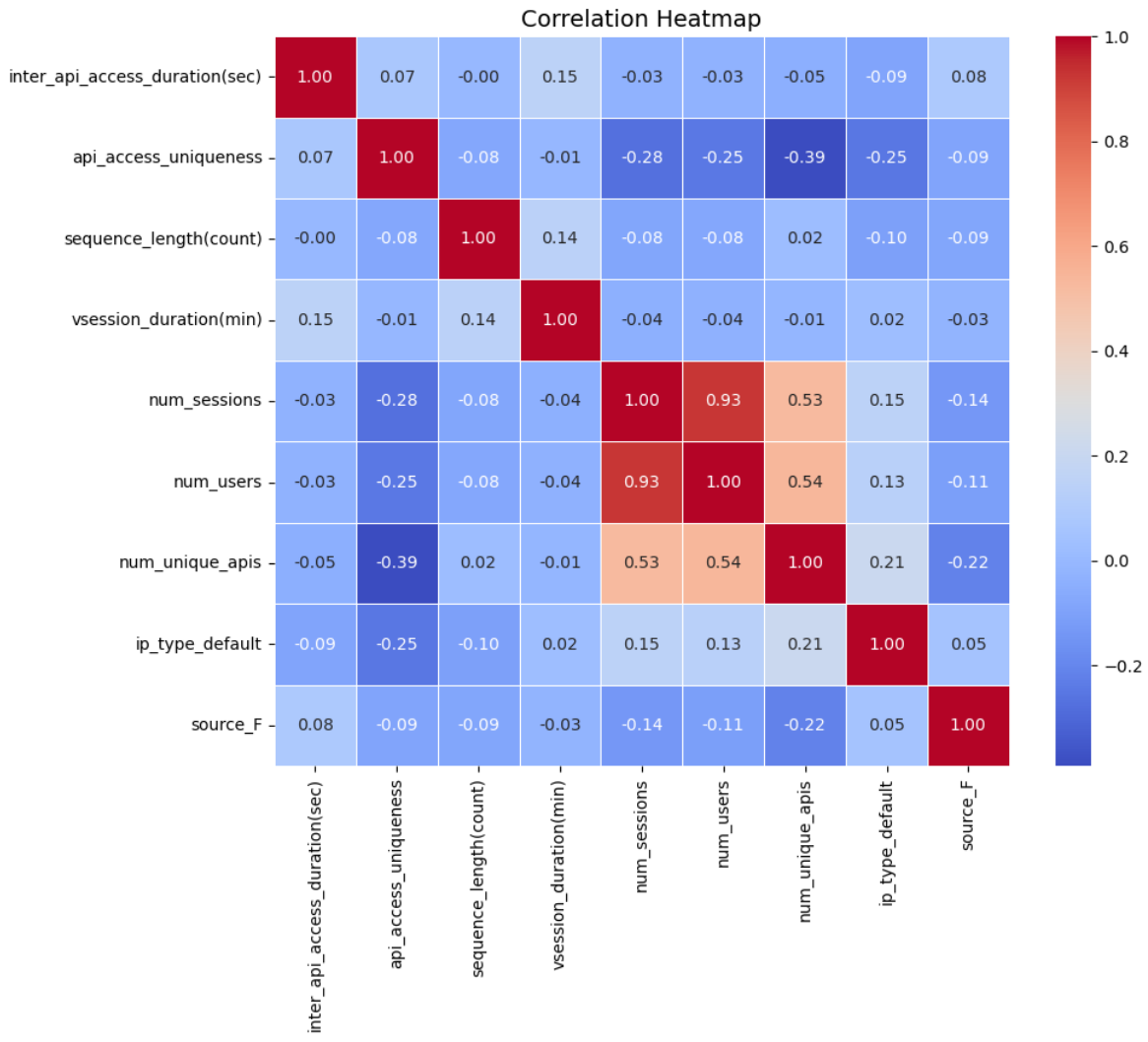


Figure 5.1: Check and drop Null values with unnecessary columns.

The first figure shows the result of df.isnull().sum(), which checks for missing values in each column of the dataset. It reveals that there are 4 missing values in the inter_api_access_duration(sec) and api_access_uniqueness columns, while all other columns, such as Unnamed: 0, _id, sequence_length(count), vsession_duration(min), and others, have no missing values. The second figure shows the code used for cleaning the dataset. It drops unnecessary columns (Unnamed: 0 and _id) using df.drop(columns=[...]) and manages

missing values by removing rows with any null values using dropna(). This ensures that only relevant data is kept for further analysis.




```
# Identify and drop highly correlated features (abs(corr) > 0.70)
# Only consider upper triangle of the correlation matrix
upper = corr.where(np.triu(np.ones(corr.shape), k=1).astype(bool))
to_drop = [col for col in upper.columns if any(upper[col].abs() > 0.70)]
print("Dropping highly correlated columns:", to_drop)

# Drop the highly correlated columns
df_encoded_reduced = df_encoded.drop(columns=to_drop)

# Display the remaining features
print("Remaining features:", df_encoded_reduced.columns.tolist())
```

```
Dropping highly correlated columns: ['num_users']
Remaining features: ['inter_api_access_duration(sec)', 'api_access_uniqueness', 'sequence_length(count)', 'vsession_duration(mi
n)', 'num_sessions', 'num_unique_apis', 'classification', 'ip_type_default', 'source_F']
```

Figure 5.2: Correlation Heatmap

The first figure shows a correlation heatmap of various features in the dataset. The colour intensity indicates the strength of the correlation between different columns, with red hues representing high positive correlations and blue hues representing negative or weak correlations. For example, num_sessions and num_users have a high correlation (0.93), suggesting that as the number of sessions increases, the number of users does as well.

The second figure shows the Python code used to identify and drop features with high correlations (greater than 0.70). The code calculates the upper triangle of the correlation matrix and identifies features with a high correlation. In this case, the num_users column is dropped, as it is highly correlated with num_sessions. The importance of this step lies in reducing multicollinearity. A problem is that high correlations among features will result in model performance that is distorted, leading to overfitting. Removing such features ensures that the model is more stable and generalises better to unseen data.

```
# Perform One-Hot Encoding on categorical columns (ip_type, source)
df_encoded = pd.get_dummies(df_cleaned, columns=['ip_type', 'source'], drop_first=True)
```

```
# Split the data into training (benign only) and testing (complete dataset)
X_train = df_encoded[df_encoded['classification'] == 'normal'].drop(columns=['classification'])
y_train = df_encoded[df_encoded['classification'] == 'normal']['classification']

X_test = df_encoded.drop(columns=['classification'])
y_test = df_encoded['classification']
```

```
# Scale the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Figure 5.3: Data Preprocessing for one-class SVM

The figures depict essential preprocessing steps for preparing data for machine learning:

- One-Hot Encoding: The first image uses pd.get_dummies() to transform categorical variables (ip_type, source) into binary columns, with drop_first=True to avoid multicollinearity.
- Data Splitting: The second image shows how the dataset is divided into training and testing sets. Training data (X_train, y_train) is filtered to include only "normal" classifications, while testing data (X_test, y_test) contains all classes for model evaluation.
- Feature Scaling: The third image illustrates the use of StandardScaler to normalise the features in the dataset, ensuring they all have a similar scale.

These steps are crucial for ensuring that machine learning models perform optimally. One-hot encoding converts categorical data into a format that algorithms can process, while splitting the data into training and testing sets prevents overfitting. Feature scaling standardises the data, making it easier for models to learn patterns and converge more effectively.

5.2 Middleware Implementation and Architectural Design

To operationalise the threat detection models, a middleware-based API gateway was developed using the Flask web framework in Python. This gateway serves as a practical implementation of the research, acting as a proxy that intercepts API requests, analyses them for anomalous behaviour using the trained machine learning models, and then either forwards the request or flags it as a potential threat.

The architecture of the Flask application is designed for modularity and scalability. Upon initialisation, the application pre-loads the trained and saved machine learning models the tuned One-Class SVM and the Hybrid RandomForestClassifier along with the corresponding data scaler object. This prevents the overhead of loading these assets for every incoming request, ensuring low-latency processing. A dedicated preprocessing module was created to mirror the exact steps used during model training, including one-hot encoding of categorical variables (ip_type, source), handling of numerical features, and feature scaling.

1. User Interface (UI) Endpoint:

An HTML-based user interface was developed to allow for manual, ad-hoc testing of API traffic data. This endpoint, accessible via a web browser, presents a form where a user can input the feature values for a single API request, such as inter_api_access_duration, sequence_length, and num_sessions. The form also includes a dropdown menu to select which model (Tuned One-Class SVM or Hybrid) should be used for the prediction. Upon submission, the form data is sent to the Flask backend, where it undergoes the full preprocessing pipeline before being passed to the selected model. The prediction result "Normal" or "Anomalous" is then rendered back to the user on the same page.

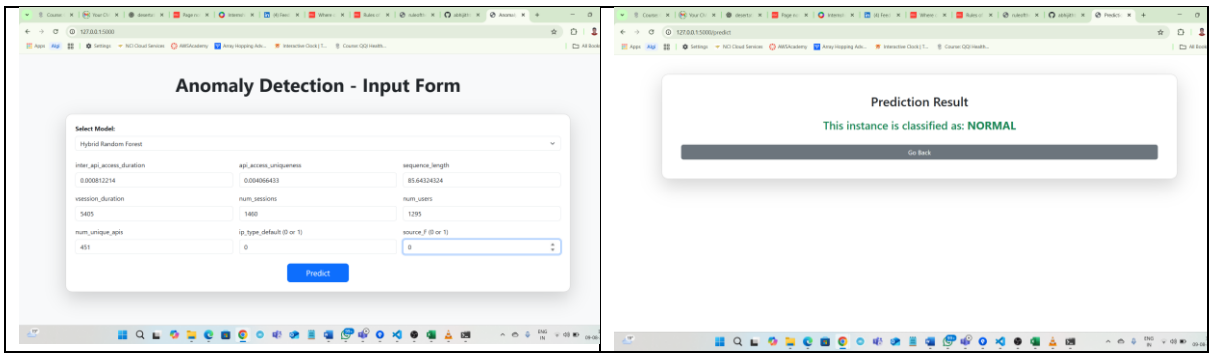


Figure 5.4: Normal Prediction

2. JSON API Endpoint:

For integration with other applications or for batch testing, a dedicated API endpoint (/predict_api) was created. This endpoint accepts POST requests with a JSON payload containing the feature data for one or more API requests. This allows for programmatic interaction, simulating how another service would use the gateway for real-time validation. The endpoint processes the JSON data, applies the preprocessing and prediction logic using the Hybrid model by default (due to its superior performance), and returns a JSON response containing the prediction for each input record. Tools like Postman or command-line scripts can be used to send batch requests to this endpoint, enabling efficient performance testing and validation of the model against a large set of test data.

5.3 Evaluation Insights and Model Superiority

The results presented in Chapter 6 clearly indicate that the Hybrid RandomForestClassifier model significantly outperformed the Tuned One-Class SVM. The primary reason for this superiority lies in the fundamental difference in their learning approaches. The One-Class SVM is an unsupervised model trained exclusively on benign data. Its task is to learn the boundary of "normal" behaviour and classify anything outside this boundary as an anomaly. While effective, this approach is inherently prone to a higher rate of false positives because any benign but novel or slightly unusual traffic pattern can be incorrectly flagged as malicious. It essentially identifies what is *not normal* rather than what is definitively *malicious*.

In contrast, the Hybrid model is a supervised classifier trained on a mixed dataset containing labelled examples of both benign and malicious traffic. This allows the model to learn the specific, distinguishing patterns and characteristics that differentiate a genuine threat from normal activity.

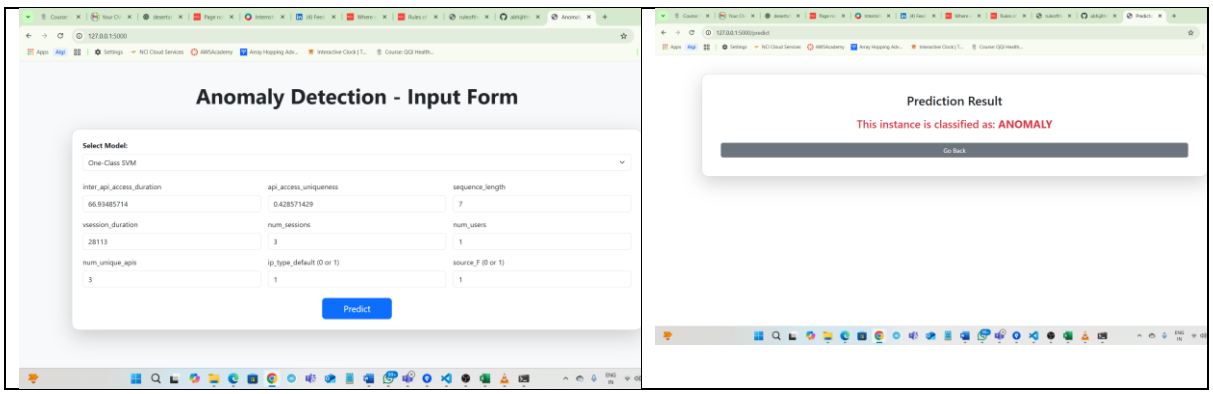


Figure 5.5: Anormal Prediction

5.4 Novelty of the Methodological Approach

The novelty of this research does not solely rest on the final, high-performing Hybrid model, but also on the methodological journey taken to arrive there. The initial implementation and tuning of a One-Class SVM model represent a significant and innovative aspect of the project. In many real-world cybersecurity scenarios, organisations possess vast amounts of data on normal user behaviour but have a very limited, or poorly labelled, dataset of actual attacks.

By first focusing on an unsupervised approach, this research directly addresses this common "data-imbalance" problem. It explores the practical limits of what can be achieved in API security when one must rely solely on defining normalcy. The process of building, tuning, and evaluating the One-Class SVM provides a valuable baseline and highlights the inherent challenges of unsupervised anomaly detection, particularly the struggle with false positives.

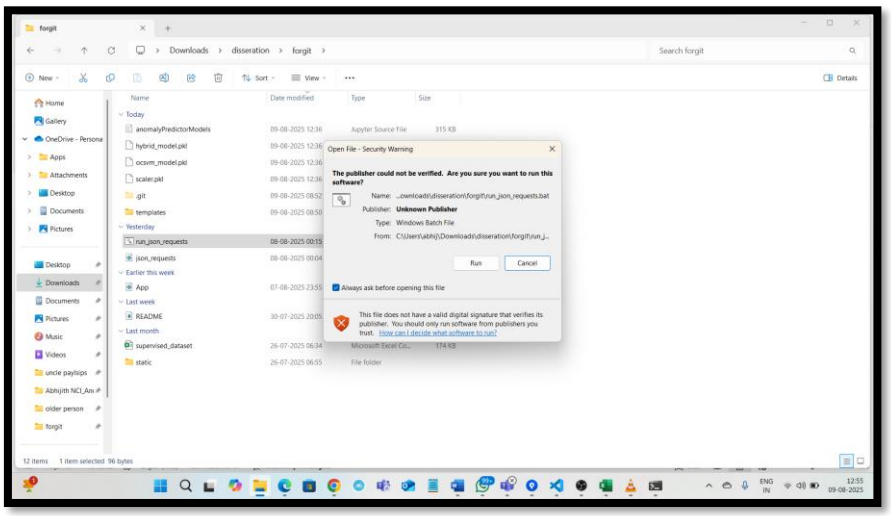


Figure 5.6: Running Batch File

5.5 Implementation Challenges and Resolutions

High Initial False Positives:

The initial, untuned One-Class SVM model produced an unacceptably high number of false positives, flagging a significant portion of benign traffic as anomalous.

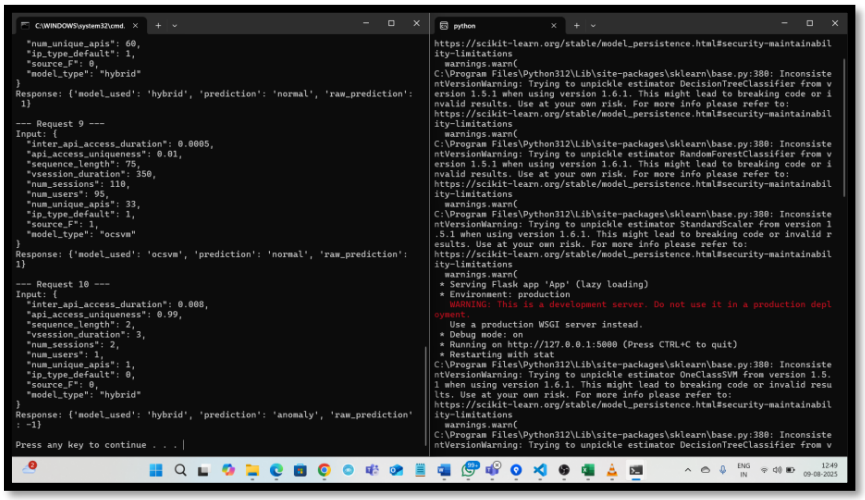


Figure 5.7: Batch File Execution Output I

Resolution: This was addressed through extensive hyperparameter tuning using GridSearchCV. By systematically searching for the optimal values for nu (the expected fraction of outliers) and gamma, we were able to tighten the decision boundary, significantly reducing false positives and improving the model's overall F1-score, as evidenced by the results in Chapter 6.

Model and Preprocessing Integration:

A significant technical hurdle was ensuring that the data preprocessing pipeline within the live Flask application was a perfect replica of the pipeline used to train the models. Minor discrepancies in feature order, scaling methods, or encoding logic led to cryptic errors and inaccurate predictions.

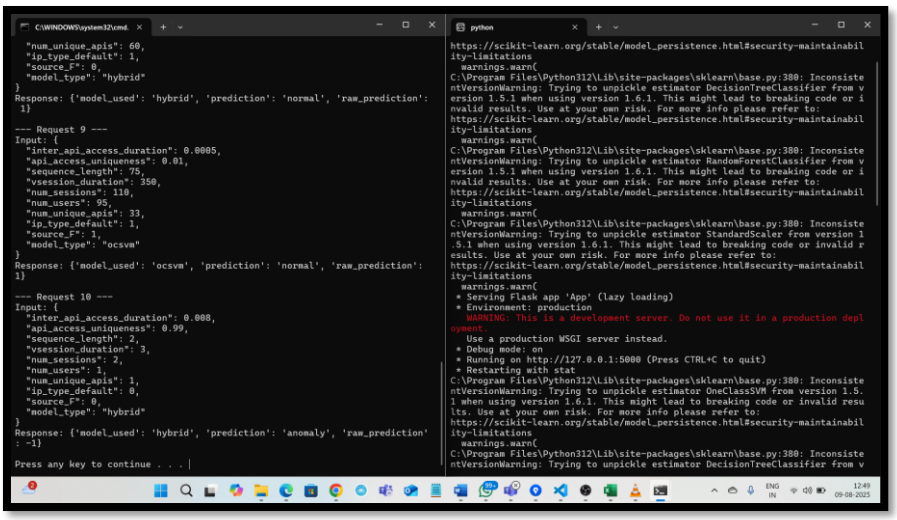


Figure 5.8: Batch File Execution Output II

Resolution: To resolve this, the preprocessing logic was encapsulated into a single, reusable Python class. This class was imported and used in both the initial model training script and the

final Flask deployment script. This ensured that any data point, whether from the original dataset or a new incoming API request, would undergo the exact same transformation, guaranteeing consistency and reliability.

Performance of the One-Class SVM:

Despite tuning, the One-Class SVM still struggled to achieve the desired balance of precision and recall. It became clear that its unsupervised nature was a fundamental limitation for this specific problem where clear threat patterns existed.

Resolution: This challenge directly motivated the development of the Hybrid model. Recognising the ceiling of the unsupervised approach, we pivoted to a supervised method that could leverage the labelled malicious data. This strategic shift from trying to perfect an ill-suited model to adopting a more appropriate one was key to the project's ultimate success.

6: Evaluation

Classification Report:

	precision	recall	f1-score	support
0	1.00	0.95	0.97	1106
1	0.91	1.00	0.95	589
accuracy			0.97	1695
macro avg	0.96	0.97	0.96	1695
weighted avg	0.97	0.97	0.97	1695

Classification Report (After Hyperparameter Tuning):

	precision	recall	f1-score	support
0	1.00	0.98	0.99	1106
1	0.96	1.00	0.98	589
accuracy			0.98	1695
macro avg	0.98	0.99	0.98	1695
weighted avg	0.99	0.98	0.98	1695

Classification Report (Hybrid Model - RF Classifier):

	precision	recall	f1-score	support
normal	1.00	1.00	1.00	332
outlier	1.00	1.00	1.00	471
accuracy			1.00	803
macro avg	1.00	1.00	1.00	803
weighted avg	1.00	1.00	1.00	803

Figure 6.1: One-class SVM, Hyper Tune One-class SVM, and Hybrid Model Classification report

The classification reports showcase the performance metrics of three different models, the One-class SVM model, the One-class SVM model after hyperparameter tuning, and the hybrid model using a random forest (RF) classifier.

The One-class SVM model performance shows that for Class 0 (normal), the model achieved high precision (1.00), recall (0.95), and F1-score (0.97), indicating strong detection with minimal false positives, but a slight recall loss. For Class 1 (outlier), precision (0.91), recall (1.00), and F1-score (0.95) are solid, but the slightly lower precision suggests room for improvement. The overall accuracy of 0.97 indicates superior performance, though hyperparameter tuning could enhance precision, especially for outlier detection.

After hyperparameter tuning, the performance improved significantly. Class 0's precision (1.00), recall (0.98), and F1-score (0.99) saw notable gains, reducing false negatives without compromising precision. For Class 1, precision (0.96), recall (1.00), and F1-score (0.98) showed better precision and maintained perfect recall. Overall, the accuracy increased to 0.98, with both macro and weighted averages improving, reflecting better consistency across both classes. The post-hyperparameter tuning results are significantly better due to improved precision and recall for both classes, reducing false negatives and enhancing model consistency, leading to higher accuracy and balanced performance across classes.

The hybrid formulation based on the RF classification approach had a good score in terms of perfect classification of the two classes, where precision, recall, and F1-scores were 1.00, and such results reflect a rich performance. The accuracy was 1.00, indicating better detection due to normal and outlier cases compared to the previous models. The hybrid model outperforms others since it designs both classes correctly, without faulty classification, and the precision, recall, as well as F1-Scores are perfect. It means that it has maximised its performance in terms of the normal and outlier classes. The enhancements observed following hyperparameter optimisation also imply the significant role of model optimisation to fit the data distribution.

7: Conclusion

7.1 Conclusion

The work that had been done also effectively improved the safety of API through middleware and machine learning to detect anomalies in real-time. The results show that machine learning models, especially the hybrid method, can be of immense help in the detection and mitigation of API threats. This study will bridge the gap in the current API security practices where dynamic and adaptive solutions are used.

7.2 Strengths and Weaknesses

This is a strong point of the research; the implementation of machine learning with the integration of middleware to run security of the API is an innovative approach. It is good at real-time anomaly detection and mitigation of threats. A drawback is, however, the fact that it uses a publicly available dataset, which does not accurately reflect the complexity of API traffic in the real world, which impacts generalisation.

7.2 Future recommendation

It is recommended to further validate the proposed middleware-based API security solution using diverse, real-world datasets. Future implementations should also focus on optimising machine learning models for increased scalability and performance in high-traffic API environments. Additionally, exploring advanced anomaly detection methods could enhance the robustness of the proposed solution.

References

Aharon, U., Marbel, R., Dubin, R., Dvir, A. and Hajaj, C. (2024). Few-Shot API Attack Detection: Overcoming Data Scarcity with GAN-Inspired Learning. *arXiv (Cornell University)*. doi:https://doi.org/10.48550/arxiv.2405.11258.

Al-Ghuwairi, A.-R., Sharrab, Y., Al-Fraihat, D., AlElaimat, M., Alsarhan, A. and Algarni, A. (2023). Intrusion detection in cloud computing based on time series anomalies utilizing machine learning. *Journal of Cloud Computing*, 12(1). doi:https://doi.org/10.1186/s13677-023-00491-x.

Ali, Z., Mahmood, A., Khatoon, S., Alhakami, W., Ullah, S.S., Iqbal, J., and Hussain, S. (2023). A Generic Internet of Things (IoT) Middleware for Smart City Applications. *Sustainability*, [online] 15(1), p.743. doi:https://doi.org/10.3390/su15010743.

Arumugam, K.J. (2025). Behind the Cloud: Uncovering Critical Security Threats. [online] doi:https://doi.org/10.2139/ssrn.5160686.

Behbehani, D., Rajarajan, M., Komninos, N. and Al-Begain, K. (2022). Detecting Open Banking API Security Threats Using Bayesian Attack Graphs. *2022 14th International Conference on Computational Intelligence and Communication Networks (CICN)*. doi:https://doi.org/10.1109/cicn56167.2022.10008365.

Casula, M., Rangarajan, N. and Shields, P. (2021). The Potential of Working Hypotheses for Deductive Exploratory Research. *Quality & Quantity*, 55(1), pp.1703–1725. doi:https://doi.org/10.1007/s11135-020-01072-9.

Dini, P., and Saponara, S. (2023). Design and Experimental Assessment of Real-Time Anomaly Detection Techniques for Automotive Cybersecurity. *Sensors*, 23(22), pp.9231–9231. doi:https://doi.org/10.3390/s23229231.

Drofa, D. (2025). *Integrating Advanced API Solutions into Full-Stack Web and Mobile Applications to Optimise User Experience*. [online] Doi.org. Available at: https://doi.org/10.47191/ijcsrr/V8-i5-16 [Accessed 17 Jul. 2025].

Guillén-Gámez, F.D., Mayorga-Fernández, M.J. and Contreras-Rosado, J.A. (2021). Incidence of Gender in the Digital Competence of Higher Education Teachers in Research Work: Analysis with Descriptive and Comparative Methods. *Education Sciences*, 11(3), p.98. doi:https://doi.org/10.3390/educsci11030098.

Guntur, R. (2021). *API security: Access behavior anomaly dataset*. [online] Kaggle.com. Available at: https://www.kaggle.com/datasets/tangodelta/api-access-behaviour-anomaly-dataset.

Haghani, M., Coughlan, M., Crabb, B., Dierickx, A., Feliciani, C., Gelder, R. van , Geoerg, P., Hocaoglu, N., Laws, S., Lovreglio, R., Miles, Z., Nicolas, A., O'Toole, W., Schaap, S., Semmens, T., Shahhoseini, Z., Spaaij, R., Tatrai, A., Webster, J. and Wilson, A. (2023). A roadmap for the future of crowd safety research and practice: Introducing the Swiss Cheese

Model of Crowd Safety and the imperative of a Vision Zero target. *Safety Science*, [online] 168(168), pp.106292–106292. doi:https://doi.org/10.1016/j.ssci.2023.106292.

Karri, S.B., Penugonda, C.M., Karanam, S., Tajammul, M., Rayankula, S. and Vankadara, P. (2025). Enhancing Cloud-Native Applications: A Comparative Study of Java-To-Go Micro Services Migration. *International Transactions on Electrical Engineering and Computer Science*, 4(1), pp.1–12. doi:https://doi.org/10.62760/iteecs.4.1.2025.127.

Modi, B., Chourasia, U. and Pandey, R. (2022). Design and implementation of RESTFUL API based model for vulnerability detection and mitigation. *IOP Conference Series: Materials Science and Engineering*, 1228(1), p.012010. doi:https://doi.org/10.1088/1757-899x/1228/1/012010.

Salt Security (2022). *API Security | Salt Security*. [online] Salt Security. Available at: https://salt.security/ [Accessed 17 Jul. 2025].

Samper, J., and Ferreira, B. (2024). *Leveraging remote attestation APIs for secure image sharing in messaging apps*. [online] Cryptology ePrint Archive. Available at: https://ia.cr/2024/2057 [Accessed 17 Jul. 2025].

Schröer, C., Kruse, F., and Gómez, J.M. (2021). A Systematic Literature Review on Applying CRISP-DM Process Model. *Procedia Computer Science*, [online] 181(1), pp.526–534. Available at: https://www.sciencedirect.com/science/article/pii/S1877050921002416.

Thirimanne, S.P., Jayawardana, L., Yasakethu, L., Liyanaarachchi, P. and Hewage, C. (2022). Deep Neural Network Based Real-Time Intrusion Detection System. *SN Computer Science*, 3(2). doi:https://doi.org/10.1007/s42979-022-01031-1.

Traeger, S. (2024). *Ultimate Guide to Swiss Cheese Model and Its Applications*. [online] reliability.com. Available at: https://reliability.com/resources/guide-to-swiss-cheese-model-with-examples/.

Xiong, K., Wu, Z., and Jia, X. (2025). DeepContainer: A Deep Learning-based Framework for Real-time Anomaly Detection in Cloud-Native Container Environments. *Journal of Advanced Computing Systems*, 5(1), pp.1–17. doi:https://doi.org/10.69987/jacs.2025.50101.