National College of Ireland

Streamlining Personalised Vulnerability Analysis and Anomaly Detection through API requests with a Random Forest Model

MSc Research Project
MSc in Cybersecurity

Athipan Thiyada Sridharan
Student ID: x22135405

School of Computing
National College of Ireland

Supervisor: Apurva Vangujar

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Athipan Thiyada Sridharan
Student ID:	x22135405
Programme:	MSc in Cybersecurity
Year:	2023
Module:	MSc Research Project
Supervisor:	Apurva Vangujar
Submission Due Date:	14/12/2023
Project Title:	Streamlining Personalised Vulnerability Analysis and Anomaly Detection through API requests with a Random Forest Model
Word Count:	4773
Page Count:	33

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:	
Date:	14th December 2023

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):	

Streamlining Personalised Vulnerability Analysis and Anomaly Detection through API requests with a Random Forest Model

Athipan Thiyada Sridharan
x22135405

Abstract

While cyber threats are rapidly evolving, it is crucial to have effective anomaly detection systems to monitor our server. This research introduces a machine learning-based detector designed to identify anomalies and vulnerability analysis in web applications. By analyzing historical data of known vulnerabilities and monitoring real-time traffic, the system distinguishes between normal and malicious activities, ensuring a strong security defence. Integrating this model into a web application allows for an automated, immediate response to potential security threats. The study's findings suggest that applying machine learning algorithms can significantly decrease the likelihood of overlooking security breaches, thus strengthening web applications against evolving cyber-attacks.

Keywords: Anomaly Detection, Web Application Security, Machine Learning, Scikit learn , pandas, Random Forest, Continuous Learning, Python, ReactJS, API, Request Filter, Logging.

1 Introduction

This thesis tackles the crucial challenge of enhancing web application security, specifically focusing on anomaly detection to monitor the server thereby safeguarding it from cyber-attacks. We begin by identifying significant gaps in current security measures and drawing attention to recent data breaches and cyber-attacks.

The study by Ponemon Institute (2023) found that organizations that use anomaly detection are 50% less likely to suffer a data breach than those that don't. The average cost of a data breach in 2023 is $3.66 million. This means that organizations that use anomaly detection can save an average of $1.83 million per year on data breaches.

The anomaly detection is becoming a must-have layer in any domain, which is the high motivation for publishing this paper.

The heart of our research is a machine learning-based method that uses Random Forest to analyze complex incoming traffic and its patterns. This approach proves particularly effective in distinguishing malicious requests from normal API requests, a critical aspect of web application security.

The report is organized into several sections. It commences with the Research Question, followed by the clear objective of the study. The next part contains the Literature

Review. This provides the necessary background and existing studies made in the relevant field. The Methodology section explains much more about the proposed solution and how it will work in real time.

Subsequently, we define the Design Requirements for our proposed solution, ensuring it meets the necessary criteria. The Implementation chapter details the practical application of our Random Forest model. We then conduct an Evaluation to assess the model's effectiveness in detecting threats, followed by a Discussion where we analyze and interpret our findings. The thesis concludes with a section reflecting on the contributions of our research to web application security, enabling future scope to make more advancements in the field of anomaly detection.

2 Research Questions

- How can machine learning assist in finding anomalies in the HTTP requests?
- How can vulnerability analysis be made more effective by collecting data impacted by the requests?
- How to reduce false positives of the machine learning model that is used in the vulnerability Analysis?

3 Reseach Objective

The main objective of the research is to enhance web application security by employing a machine learning-based request filtering system, with a specific focus on training the Random Forest model. A model trained with the malicious and non-malicious log files will be integrated with the request filter, which is set to evaluate each incoming request to the server, determining whether the input is malicious or not.

The Malicious dataset is obtained from past incidents, or by conducting penetration testing for the chosen application. Most application will record their past incidents in the bug bounty tracker.

The prediction made by the model is written as a separate log file which is set to further evaluation, where the corrections are again made in case of false positive cases, and again the Random Forest model will be built with the updated dataset. The feedback-driven system enhances the efficiency of the model thereby reducing the false positive cases.

Refer to fig. 1 for the overview of the project.

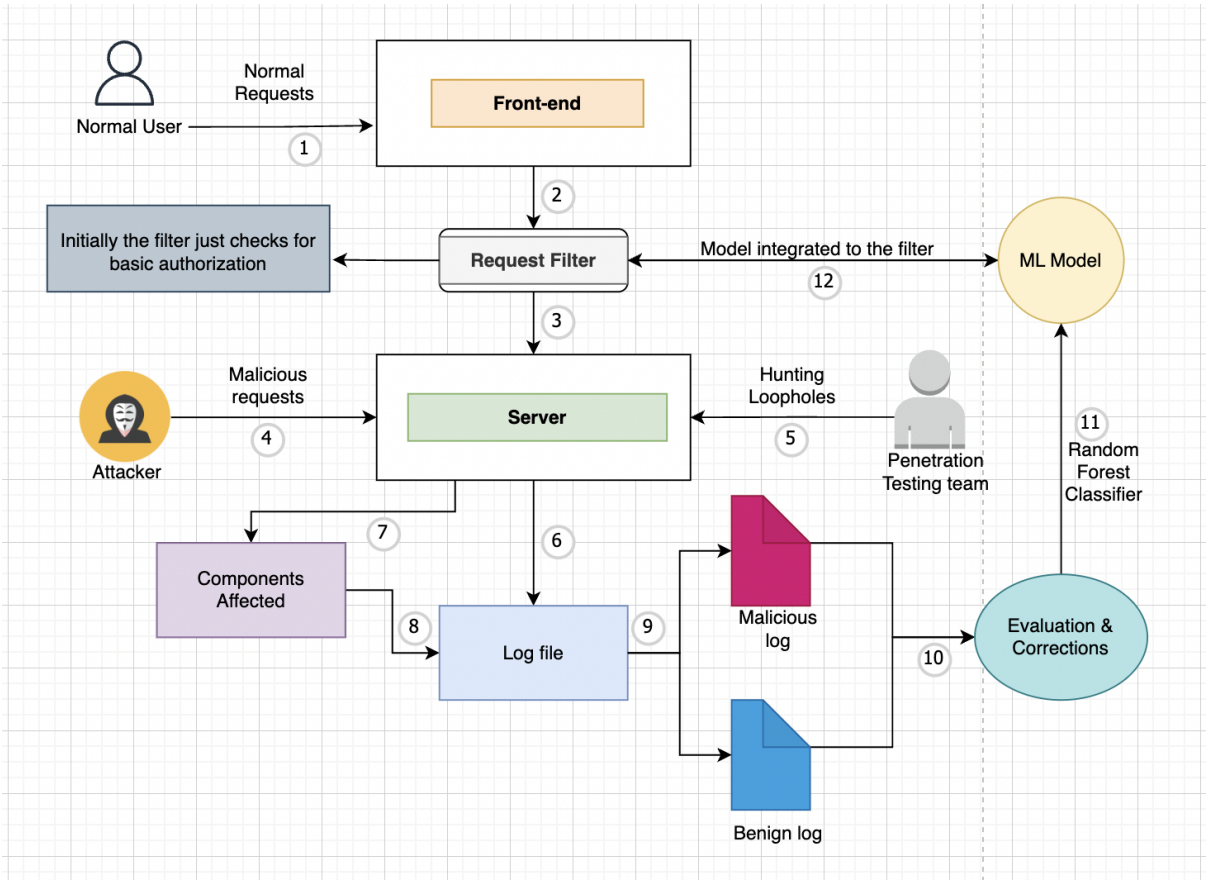

Figure 1: The Project overview

4 Related Work

In this section, we explore various significant studies in anomaly detection within web applications, examining their research outcomes and how they compare with our project.

4.1 ITES: Integrated Testing and Evaluation System for Software Vulnerability Detection Methods (2020)

It was developed by Zhang et al. (2020). ITES evaluates software vulnerability detection methods by integrating the testing framework into the application. It automates the collection of vulnerability cases, forming an extensive vulnerability cases library. The system assesses fourteen different detection methods on both Windows and Linux platforms. ITES introduces a range of evaluation metrics like accuracy and resource cost. Since their paper focuses on vulnerability detection with their integrated libraries, the concept of integration and finding vulnerability is on similar ground to our paper. Whereas our main focus is on evaluating a specific anomaly detection methodology with a streamlined development methodology.

4.2 Survey of Automatic Software Vulnerability Detection, Program Repair, and Defect Prediction (2020)

It was conducted by Shen and Chen (2020). this survey is about automated technologies in software vulnerability detection, program repair, and defect prediction. The research focuses on the impact of deep learning on software security. It evaluates various automated techniques summarizes their features and discusses challenges and future research directions in software security. Their challenges and research outcomes of deep learning help us to overcome the false positive cases by adopting the continuous learning method.

4.3 MONILOG: An Automated Log-based Anomaly Detection System for Cloud Computing Infrastructures (2021)

Developed by Vervaet (2021). MoniLog is an innovative anomaly detection system mainly for cloud computing infrastructures. This research utilizes log data to detect anomalies in large-scale environments in real time. Automates the anomaly detection process to manage increasing log data volumes. Structures log streams and classifies anomalies based on criticality. Our paper utilizes the log files, that way, this paper helped us to understand the handling of the log files and find the anomalies in the log files. In contrast to their project, our paper proposes a different way of anomaly detection with the integration of the machine learning models and adopting a continuous learning method by streamlining the web development specifically for this anomaly detection.

4.4 Analysis of Software Development Process in Respect to Anomaly Detection (2018)

Zavarzin and Afanaseva (2019) explored anomaly detection in Agile software development. Introduced a novel method using k-means clustering and fuzzy transform (F-transform) for anomaly detection. Capable of identifying anomalies in both stationary and non-stationary time series. Allows for iterative refinement for enhanced detection

precision. Their approach is distinct from our project's focus on datasets obtained from past incidents.

4.5 Automated Software Vulnerability Detection with Machine Learning (2018)

Harer et al. (2018) focused on using machine learning for automated software vulnerability detection. Compiled a large dataset of open-source functions analyzed with static analyzers. Compared source-based models with build process artefact models, preferring the former. Achieved notable performance metrics using deep learning and tree-based models. Although using machine learning, their approach and technologies differ from our project.

4.6 Self-Attention based Automated Vulnerability Detection with Effective Data Representation (2021)

Conducted by Wu et al. (2021), focusing on using the Transformer model for vulnerability detection. Emphasized extracting Code Slices for precise vulnerability pattern representation. Introduced an effective data representation method to enhance semantic information retention. Demonstrated Transformer's superiority over RNN models in vulnerability detection tasks.

4.7 Machine Learning for Anomaly Detection: A Systematic Review (2021)

Conducted by Nassif et al. (2021), this paper presents a review of machine learning models for anomaly detection, analyzing a wide range of articles from 2000-2020. It covers various applications of anomaly detection, highlighting the versatility of machine learning in this domain. The paper discusses the prevalence and effectiveness of unsupervised machine learning techniques in anomaly detection. Insights from this study are most important in understanding the trends and effectiveness of different ML models in anomaly detection, which is crucial for our project's focus on machine learning-based anomaly detection in web applications.

4.8 Mathematical Validation of Proposed Machine Learning Classifier for Heterogeneous Traffic and Anomaly Detection (2020)

Guezzaz et al. (2021) focused on developing a machine learning classifier for anomaly detection in network traffic. It proposes a novel classifier model and highlights the importance of preprocessing and optimization in machine learning algorithms. The paper makes a significant contribution to the field of IDS, providing modern traffic analysis and anomaly detection. The approaches and methodologies discussed in this study are valuable for our project, especially in terms of model development and validation techniques.

4.9 Survey of Intrusion Detection Systems: Techniques, Datasets, and Challenges (2019)

This paper was submitted by Khraisat et al. (2019). This survey offers an in-depth review of intrusion detection systems, both signature-based and anomaly-based. It provides a detailed taxonomy of IDS techniques and evaluates their effectiveness through various research works. The paper discusses the challenges in intrusion detection and the evasion techniques used by attackers, providing a structured overview of the field. This survey is particularly relevant to our project as it gives insights into the evolution of intrusion detection systems and the importance of continuous development in this area.

4.10 AutoLog: A Log Sequence Synthesis Framework for Anomaly Detection (2023)

It is developed by Huo et al. (2023). AUTOLOG addresses the need for log datasets in anomaly detection. The framework uses program analysis to actively generate log sequences, a novel approach in the field. AUTOLOG demonstrates a major improvement in log event generation and offers customizable parameters for various scenarios. The methodologies and innovations presented in AUTOLOG are crucial to our project, particularly in terms of log data analysis and handling for anomaly detection.

4.11 An Automated Closed-Loop Framework to Enforce Security Policies from Anomaly Detection (2023)

This study is made by Henriques et al. (2022). They propose an automated process for integrating anomaly detection with policy enforcement in IT systems. It presents a novel methodology for enforcing high-level security policies into specific, actionable rules based on anomaly detection results. The framework aligns with modern network management and security automation principles, making it a substantial contribution to the information security field. The insights from this paper are valuable for our project, particularly in understanding the application of anomaly detection results in digital security scenarios.

4.12 Anomaly Detection, Analysis, and Prediction Techniques in IoT Environment: A Systematic Literature Review (2019)

This research by Fahim and Sillitti (2019) investigates anomaly detection in IoT environments. Their findings are valuable for my project, as they offer insights into detecting unusual patterns in complex systems. This helps in understanding the handling of large datasets and the application of machine learning, which is parallel to web application environments.

4.13 A Comparative Study of Machine Learning Algorithms for the Detection of Vulnerable Python Libraries (2022)

The study by Pérez-Vilarelle et al. (2022) compares machine learning models for identifying vulnerabilities in Python libraries. This paper gives the overview of the ML algorithms used in finding vulnerabilities in the Python libraries, which is significant for this research,

as it aids in understanding how different machine learning algorithms can be employed for security purposes in web applications, especially for detecting vulnerabilities.

4.14 The Rise of Software Vulnerability: Taxonomy of Software Vulnerabilities Detection and Machine Learning Approaches (2021)

Hanif et al. (2021) offered a comprehensive taxonomy of software vulnerability detection, emphasizing machine learning approaches. This paper is particularly relevant to this work, as it provides a broad overview of how machine learning, including supervised and deep learning techniques, can be utilized for detecting software vulnerabilities, aligning closely with our project's aim.

Research Niche

This study focuses on developing a specialized anomaly detector for web applications using machine learning algorithms with the dataset obtained from past incidents enhanced with feedback-based continuous learning method, which is a novel approach towards anomaly detection. The insights from these papers are critical in shaping the methodologies and approaches for this research, particularly in terms of data handling, algorithm selection, and understanding the landscape of software vulnerabilities and anomaly detection.

5 Methodology

This section explains the methodology employed in our study, focusing on the use of impacted data and previously collected anomaly logs from a web application. These data are crucial in training our machine learning model, which is directly integrated into the application.

5.1 Collection of Previously Affected Vulnerabilities or Anomalies

The methodology begins by assembling a list of vulnerabilities previously impacting the web application, along with necessary details for reproducing the issue. This can be done by collecting the dataset from the bug bounty programs conducted, vulnerabilities recorded in the bug tracker and also conducting a special penetration testing to collect the logs from the incidents.

For instance, consider a scenario where the product was previously vulnerable to SQL injection. We collect detailed request specifications and root causes of such attacks, storing them in a structured format as illustrated in Figure 17.

The step involves gathering data from various sources that significantly influence the vulnerabilities and making them into a single dataset. These sources might include:

1. API Requests:
    - Request path
    - Request body

- Request status
- Cookies in the request
- Content type of the request
- Request Header

2. API Response Parameters:

- Response body
- Response status
- Cookies in the response
- Content type of the response

For demonstration, we have selected these parameters for the study, but our proposed solution is not restricted to any particular source.

Any source that is affected by the request can be collected. For example, additional data sources such as Database Audits, response time, CPU Utilization file changes, and permission changes can also be included based on specific vulnerabilities.

5.2 Data Representation

At this stage, we possess a dataset of various vulnerabilities, stored similarly to the impact data logged from the server. This dataset is saved as 'Malicious.json'.

Parallelly, we maintain a 'Benign.json' file containing normal data logs devoid of any malicious content, representing non-threatening scenarios.

Refer to Fig. 2 for the data collection method.

5.3 Model Development

Before model training, the dataset requires cleaning and preparation, involving:

- Removal of irrelevant elements
- Selection of columns critical for training, determined by the presence of malicious payloads
- Adding or removing the data to make the dataset reliable and to improve the efficiency of the model.
- Use the scrapy framework to assemble and flatten the necessary data before training the model.

5.4 Model Selection and Training

The core of our project is the machine learning model:

- We choose the Random Forest algorithm, as it is suitable for anomaly detection.
- Both 'Benign.json' and 'Malicious.json' datasets are labelled appropriately.
- The model is trained and fine-tuned to optimize its detection efficiency in this step, after which it can be capable of predicting the incoming traffic.

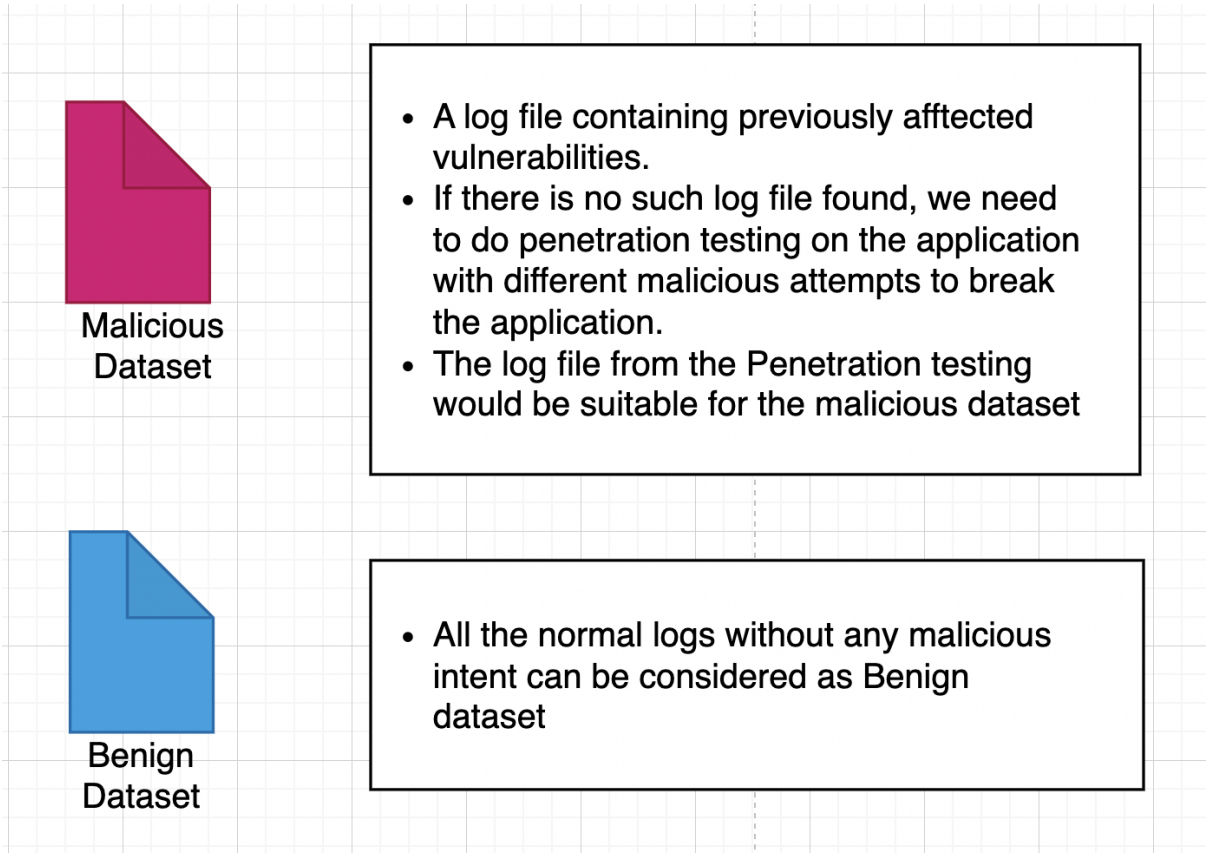

Figure 2: Dataset collection Overview

5.5 Integration for Real-Time Detection

- Following training, the model can be downloaded as a joblib and it can be integrated into the web application.
- It needs to be integrated and on the filter of any request to the server, the traffic needs to be redirected to the model to make the prediction.
- By this, it analyzes API traffic in real-time, either within the web server or as middleware.
- It acts as a monitoring mechanism for instant anomaly detection and alerts.
- In case, if the traffic is found vulnerable, it will be saved into the malicious log and if not, it will be saved in the normal log.

5.6 Continuous Model Improvement

Acknowledging the ever-evolving nature of web threats, we ensure:

- Regular updates of the dataset with new API logs.
- Periodic retraining of the model to stay abreast of new threat patterns.
- Implementation of a feedback loop for continuous model refinement.

5.7 Development of the Anomaly Detection System

The final step is to develop a comprehensive anomaly detection system that:

- Monitors API traffic to detect anomalies in real time.
- Produces detailed reports on identified anomalies.
- Seamlessly integrates with existing security frameworks to enhance the application's defence mechanism.

6 System and Software Infrastructure

Our project utilizes a MacBook Air with an M1 chip, 8GB RAM, and a 256GB SSD, chosen for its efficient computing power, important for our advanced vulnerability scanning solution.

6.1 Implementation Framework

Python 3.x: For developing the anomaly detector and implementing it with a web application, we used Python 3. Python offers a wide range of library ecosystems, ensuring access to the latest features and system compatibility.

Core Machine Learning and Data Management Libraries:

- *Scikit-learn:* Central to our model development, it provides tools for data mining and analysis.
- *Pandas:* Key for data manipulation, crucial in dataset preparation and preprocessing.
- *NumPy:* Used for advanced mathematical computations and handling large data arrays.

Data Acquisition and Management Tools:

- *Requests:* Facilitates HTTP interactions for API data extraction.
- *BeautifulSoup/Scrapy:* Essential for web scraping, augmenting our dataset with web-derived data. We used this to flatten the request data and to assemble into single dataset.
- *SQLAlchemy:* A ORM tool, that simplifies database interactions. We used this on the web application that we created for evaluating the model.

7 Implementation Overview

This section elaborates on the implementation part of the anomaly detector proposed. The proposed solution can be implemented in different and more effective ways. For demonstration, We chose the request parameters of a librarian application for training and evaluation of the model. The application was developed for this research with basic

librarian features to handle the requests and responses, and with the proper development standards, which involve Authentication, Authorization, access control, Session management, CSRF handling, Logging, Headers and cookies handling.

This ensures that the application is secure by default and it requires a proper attack vector to break the application.

The application was built by using the ReactJS as front end and Python as a back end. Refer Fig. 3 for the default application view.

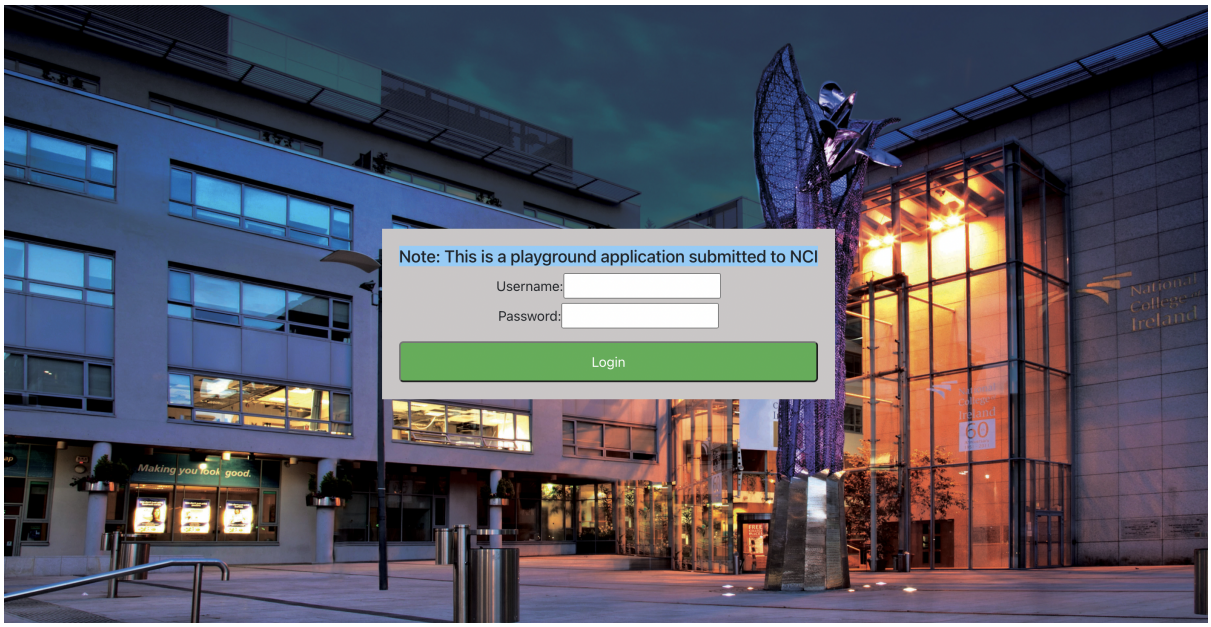

Figure 3: Login page of the test Application

The development of our anomaly detection system on the chosen application required a step-by-step process where many conflicts and struggles were faced. Multiple alternative approaches were researched, resulting in a solution that's both effective and innovative. Below are the processes involved in the implementation.

7.1 Preparing the dataset:

- A perfect logging method was implemented on the server side to collect the server logs of the application. Refer fig. 4 for the implementation of the logging functions, implemented on the application.

  The application was designed with two main roles,

  - Admin Login
  - User Login

  Refer to Fig. 5 and 6 for the preview of two portals.

  The main features of the application were,

  - Ordering the book

- Returning the book
- Adding new users
- Adding new books to the library
- Removing the book from the library

The penetration testing was conducted on the application using various well-known vulnerabilities such as SQL Injection, XSS, command injection, Path Traversal attack etc.

The impact data for both the normal requests and malicious requests were collected during each request from the front end, which included API request parameters, cookies, headers, payload and response.

Python package named Logging was used to collect all the details and they were written to a new logfile called impactdata.json. Refer to Fig. 7 for the log generated by the server.

This JSON file contains the logging of each API call in the JSON array format.

```python
@app.route('/api/logout', methods=['POST'])
def logout():
    request_details = {
        "req": {
            "api": request.path,
            "authorization": session.get('user_id'),
            "request_data": get_request_data(),
            "cookies": request.cookies,
            "headers": {key: value for key, value in request.headers.items()}
        }
    }
    # Clear the session
    session.clear()
    response = make_response(
        jsonify({"message": "Logged out successfully"}), 200)

    response.delete_cookie('session')
    response.delete_cookie('anti-csrf')

    print("session cleared")

    try:
        if os.path.exists(log_file_path) and os.path.getsize(log_file_path) > 0:
            with open(log_file_path, 'r') as log_file:
                log_data = json.load(log_file)
        else:
            log_data = []
    except json.JSONDecodeError:
        log_data = []


    request_details['req']['response'] = "Logged out successfully"

    log_data.append(request_details)
        # Write the updated log data back to the file
    with open(log_file_path, 'w') as log_file:
        json.dump(log_data, log_file, indent=4)

    return response, 200
```

Figure 4: Code involved in logging the data

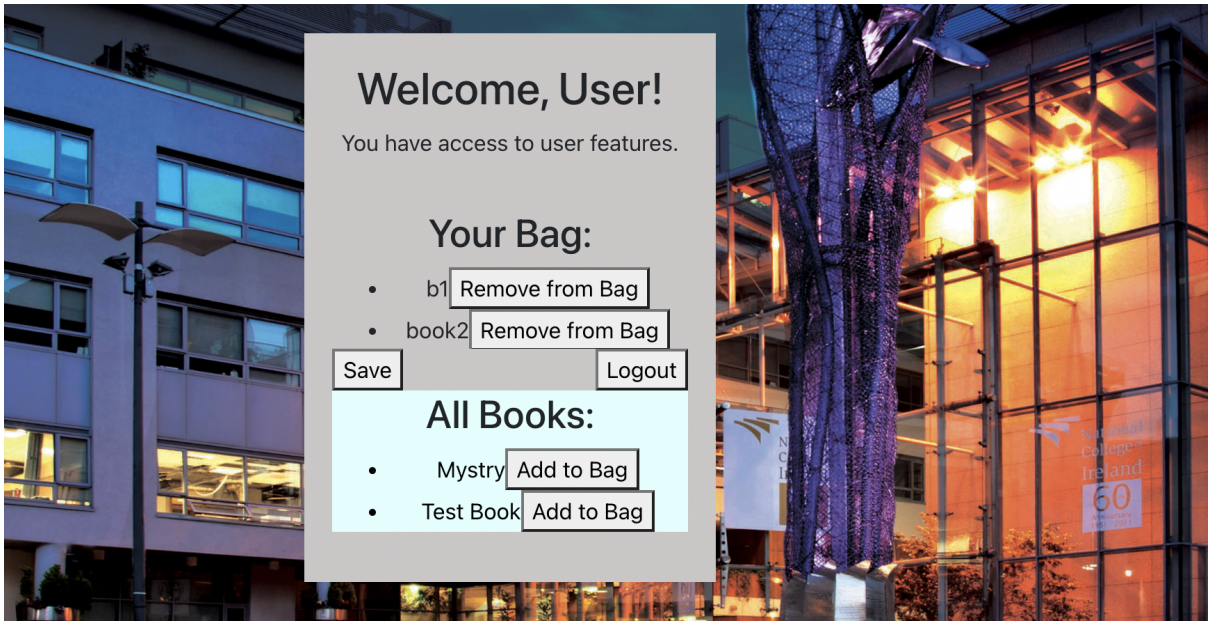

Figure 5: User Portal

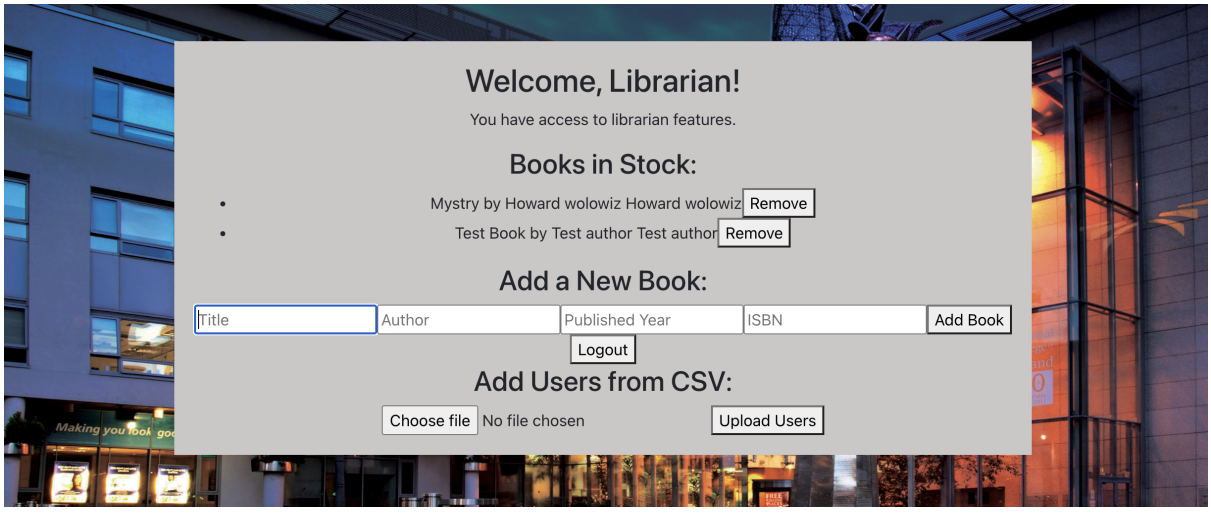

Figure 6: Admin Portal

```
{
    "req": {
        "api": "/api/getAllBooks",
        "authorization": null,
        "request_data": {},
        "cookies": {
            "session": "e0d3f0e6-bfaa-476f-bbfe-087e239e7e1c",
            "JsessionToken": "lib",
            "anti-csrf": "PJ69eKlVy_va-uf5VEsTjQ"
        },
        "headers": {
            "Host": "localhost:5000",
            "Connection": "keep-alive",
            "Sec-Ch-Ua": "\"Google Chrome\";v=\"119\", \"Chromium\";v=\"119\", \"Not?A_B
            "Sec-Ch-Ua-Mobile": "?0",
            "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/5
            "Sec-Ch-Ua-Platform": "\"macOS\"",
            "Accept": "*/*",
            "Origin": "http://localhost:3000",
            "Sec-Fetch-Site": "same-site",
            "Sec-Fetch-Mode": "cors",
            "Sec-Fetch-Dest": "empty",
            "Referer": "http://localhost:3000/",
            "Accept-Encoding": "gzip, deflate, br",
            "Accept-Language": "en-US,en-GB;q=0.9,en;q=0.8",
            "Cookie": "session=e0d3f0e6-bfaa-476f-bbfe-087e239e7e1c; JsessionToken=lib; a
        },
        "response": {
            "books": [
                {
                    "title": "Mystry",
                    "author": "Howard wolowiz",
                    "id": 2342
                }
            ]
        }
    }
}
```


Figure 7: An example of collected Data

- **Malicious Dataset:** This dataset comprised logs indicative of malicious activities and attacks. These were meticulously collected from our server's historical traffic data from the penetration testing conducted on the application. Usually, companies maintain the record of incidents on their trackers, and bug bounty programs data etc. Such a record can also be a perfect fit for the malicious dataset.

  Our dataset contains some of the malicious payloads such as XSS, SQL injection, Code injection, Command injection, Malicious headers, IDOR and many other malicious payloads that were used to break the application. The impact data of such requests were collected as Malicious.json. Refer to Fig. 8 for the malicious dataset.


```
{
  "req": {
    "api": "/api/addBook",
    "authorization": null,
    "request_data": {
      "title": "<xss onbeforescriptexecute=alert(1)><script>1</script>",
      "author": "saff",
      "published_year": "3453453",
      "isbn": "36546"
    },
    "cookies": {
      "session": "e0d3f0e6-bfaa-476f-bbfe-087e239e7e1c",
      "JsessionToken": "lib",
      "anti-csrf": "oG-iKBHHtpHzCqHMILxWlA"
    },
    "headers": {
      "Host": "localhost:5000",
      "Connection": "keep-alive",
      "Content-Length": "124",
      "Sec-Ch-Ua": "\"Google Chrome\";v=\"119\", \"Chromium\";v=\"119\", \"Not?A_Brand\";
      "Sec-Ch-Ua-Platform": "\"macOS\"",
      "Sec-Ch-Ua-Mobile": "?0",
      "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (
      "Content-Type": "application/json",
      "Accept": "*/*",
      "Origin": "http://localhost:3000",
      "Sec-Fetch-Site": "same-site",
      "Sec-Fetch-Mode": "cors",
      "Sec-Fetch-Dest": "empty",
      "Referer": "http://localhost:3000/",
      "Accept-Encoding": "gzip, deflate, br",
      "Accept-Language": "en-US,en-GB;q=0.9,en;q=0.8",
      "Cookie": "session=e0d3f0e6-bfaa-476f-bbfe-087e239e7e1c; JsessionToken=lib; anti-cs
    },
    "prediction": 0.25,
    "response": "Malicious request detected"
  }
}
```


Figure 8: An example of Malicious dataset

- **Benign Dataset:** It contains normal, non-malicious server logs. This dataset represented typical, everyday traffic, free from any security threats or vulnerabilities.

Normal testing of developed features was enough to collect this dataset. Refer to Fig. 9 for the benign dataset collected.

Both datasets were carefully labelled to distinguish between malicious and benign activities. The labelling process was crucial in training our machine learning model to recognize and differentiate between regular traffic and potential threats.


```
{
    "req": {
        "api": "/api/addBook",
        "authorization": null,
        "request_data": {
            "title": "Harry Potter",
            "author": "J.K. Rowling",
            "published_year": "2009",
            "isbn": "23454"
        },
        "cookies": {
            "session": "e0d3f0e6-bfaa-476f-bbfe-087e239e7e1c",
            "JsessionToken": "lib",
            "anti-csrf": "oG-iKBHHtpHzCqHMILxWlA"
        },
        "headers": {
            "Host": "localhost:5000",
            "Connection": "keep-alive",
            "Content-Length": "71",
            "Sec-Ch-Ua": "\"Google Chrome\";v=\"119\", \"Chromium\";v=\"119\", \"Not?A_Brand\";v=\"24\"",
            "Sec-Ch-Ua-Platform": "\"macOS\"",
            "Sec-Ch-Ua-Mobile": "?0",
            "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/119.0.0.0 Safari/537.36",
            "Content-Type": "application/json",
            "Accept": "*/*",
            "Origin": "http://localhost:3000",
            "Sec-Fetch-Site": "same-site",
            "Sec-Fetch-Mode": "cors",
            "Sec-Fetch-Dest": "empty",
            "Referer": "http://localhost:3000/",
            "Accept-Encoding": "gzip, deflate, br",
            "Accept-Language": "en-US,en-GB;q=0.9,en;q=0.8",
            "Cookie": "session=e0d3f0e6-bfaa-476f-bbfe-087e239e7e1c; JsessionToken=lib; anti-csrf=oG-iKBHHtpHzCqHMILxWlA"
        },
        "Prediction": 0.66,
        "response": {
            "message": "Book added successfully!!!"
        }
    }
},
```


Figure 9: An example of Benign Dataset

7.2 Model Development and Training

Upon preparing our datasets, we proceeded to develop the machine learning model. The model was trained using both the Malicious.json and benign.json datasets, ensuring it learned from a wide spectrum of data - from typical server requests to sophisticated attack vectors.

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score

normal_data = pd.read_json('./ml/normal.json') # loading the dataset
xss_data = pd.read_json('./ml/Malicious.json') # loading the dataset

# LEts add a label to each dataset
normal_data['label'] = 0  # Non-malicious
xss_data['label'] = 1     # Malicious
```

Figure 10: Code for loading the dataset

Fig. 10 shows the code involved in loading the dataset.

7.3 Data Loading and Preprocessing

We began by loading our datasets: 'normal-data' (non-malicious traffic) and 'xss-data' (malicious traffic), using Python's Pandas library. These datasets were in JSON format, making them ideal for processing with Python. Each dataset was labelled with binary values, '0' for normal data and '1' for XSS (Cross-Site Scripting) data, enabling the model to distinguish between benign and malicious requests. Refer to Fig. **??** to find the code involved in the preparation of the dataset in accordance with this step.

```
# Feature extraction
tfidf_vectorizer = TfidfVectorizer(max_features=1000)
X_tfidf = tfidf_vectorizer.fit_transform(combined_data['processed_request_data'])
y = combined_data['label']

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X_tfidf, y, test_size=0.2, random_state=42)

# Train the RandomForestClassifier
classifier = RandomForestClassifier(n_estimators=100, random_state=42)
classifier.fit(X_train, y_train)
```

Figure 11: Preparing the dataset

7.4 Feature Extraction with TF-IDF

The crux of our data processing involved transforming the requested data into a format suitable for machine learning analysis. We employed the TF-IDF Vectorizer from Scikit-

learn, a tool that converts text data into numerical vectors, emphasizing the importance of each term in the context of the document. This step was crucial in capturing the nuanced differences between normal and malicious requests. Refer to fig. 12 for the steps involved while transforming the dataset using TF-IDF Vectorizer.

7.5 Training and Test Split

The combined dataset was split into a training set (80 Percent) and a test set (20 Percent), ensuring a robust evaluation of the model's performance on unseen data. Scikit-learn's train-test-split function facilitated this division.

```
# Predict on the test set
y_pred = classifier.predict(X_test)

# Evaluate the predictions
accuracy = accuracy_score(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Display the results
print('Accuracy:', accuracy)
print('Classification Report:', class_report)
```

```
Accuracy: 0.5
Classification Report:               precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           1       0.75      0.60      0.67         5

    accuracy                           0.50         6
   macro avg       0.38      0.30      0.33         6
weighted avg       0.62      0.50      0.56         6
```

Figure 12: Training the dataset

7.6 Random Forest Classifier: Model Training

We chose the RandomForestClassifier for its efficacy in handling complex datasets and its inherent ability to perform well with minimal tuning. The classifier was trained on the training set, learning to discern patterns indicative of various attacks.

After training the model, the model joblib and vectorizer joblib files of the trained model are downloaded and imported to the web application. Refer Fig. 13 for the code involved in integrating the model with the application.

Whenever the new request hits the server, it passes through the security filter, where the incoming data are collected and evaluated by the joblib that we imported before.

The prediction variable holds the evaluation result, where a value close to '0' represents the involvement of malicious activity in the request and '1' represents the input is safe. Refer to fig. 14 to find the code involved in the prediction of the input traffic.

In case any malicious activity was detected, the details were logged into a separate log file called Malicious-log.

When the application was in training mode, we collected as many as logs possible from our logging system implemented on the application. These logs were then periodically evaluated and the false positive cases were identified. This step can be either done by

```
16  app = Flask(__name__)
17  # Enable CORS for all routes and allow credentials
18  CORS(app, supports_credentials=True, resources={
19      r"/api/*": {"origins": "http://localhost:3000"}})
20
21  model = load('model.joblib')
22  vectorizer = load('vectorizer.joblib')
23
```

Figure 13: importing and defining the joblibs to the application

manual means or by writing scripts to identify the requests that are not meeting the requirements.

For the demonstration, we have created a script file to find the anomalies with the hardcoded criteria.

Refer fig. 15, this script the is-anomalous-book function checks if a book has extra attributes.

is-anomalous-login checks if the /api/login request contains any parameters other than username and password.

With more such hardcoded criteria, the classification of benign cases and malicious cases will become easy in the initial stages, which lead to more efficiency in the collection benign and malicious logs.

After the collection of enough logs during the test run, we again trained the malicious logs and benign logs into the model, and the updated joblibs were replaced. The updated log file showed more efficiency in terms of identifying the anomalies, thereby decreasing the chances of false positive cases. This can be explained with the help of the fig. 16

In this stage, the model is successfully integrated with the request filter, thus monitoring the incoming traffic effectively.

```
# Preprocess the request details to feed into the model
features = preprocess_request_details(request_details['req'])

# Convert the features into the model's expected format
prepared_features = vectorizer.transform([features])

# Predict using the model
prediction = model.predict_proba(prepared_features)
print(prediction[0][0])

# Add prediction result to request details
request_details['req']['prediction'] = int(prediction[0][0])

# Append the new entry
if (prediction[0][0] < 0.5):
    log_Malicious_data = []
    request_details['req']['response'] = 'Malicious request detected'
    log_Malicious_data.append(request_details)
# Write the updated log data back to the file
    with open(log_file_path_m, 'w') as log_file:
        json.dump(log_Malicious_data, log_file, indent=4)
    return jsonify('malicious activity detected'), 401
```

Figure 14: Integration of Joblibs with the application

```python
import json


def is_anomalous_book(book):
    """Check if the book has only 'title', 'author', and 'id' attributes."""
    required_keys = {'title', 'author', 'id'}
    return not required_keys.issubset(book.keys())


def is_anomalous_login(request):
    """Check if the login API contains only 'username' and 'password' parameters."""
    api = request.get("api", "")
    request_data = request.get("request_data", {})
    if api == "/api/login" and not all(key in ["username", "password"] for key in request_data):
        return True
    return False


def detect_anomalies(data):
    """Detect anomalies in the dataset."""
    anomalies = []
    for entry in data:
        request = entry.get("req", {})
        response = request.get("response", {})
        books = response.get("books", [])

        # Check for anomalous books
        book_anomaly = any(is_anomalous_book(book) for book in books)

        # Check for anomalous login requests
        login_anomaly = is_anomalous_login(request)

        if book_anomaly or login_anomaly:
            anomalies.append(entry)

    return anomalies


# Read the dataset (assuming it's a JSON file)
with open("dataset.json", "r") as file:
    data = json.load(file)

# Detect anomalies
anomalous_requests = detect_anomalies(data)

# Output the results to a file
output_path = "/logs/anomalies.json"
with open(output_path, "w") as file:
    json.dump(anomalous_requests, file, indent=4)
```

Figure 15: Script to find the anomalies with hardcoded criteria

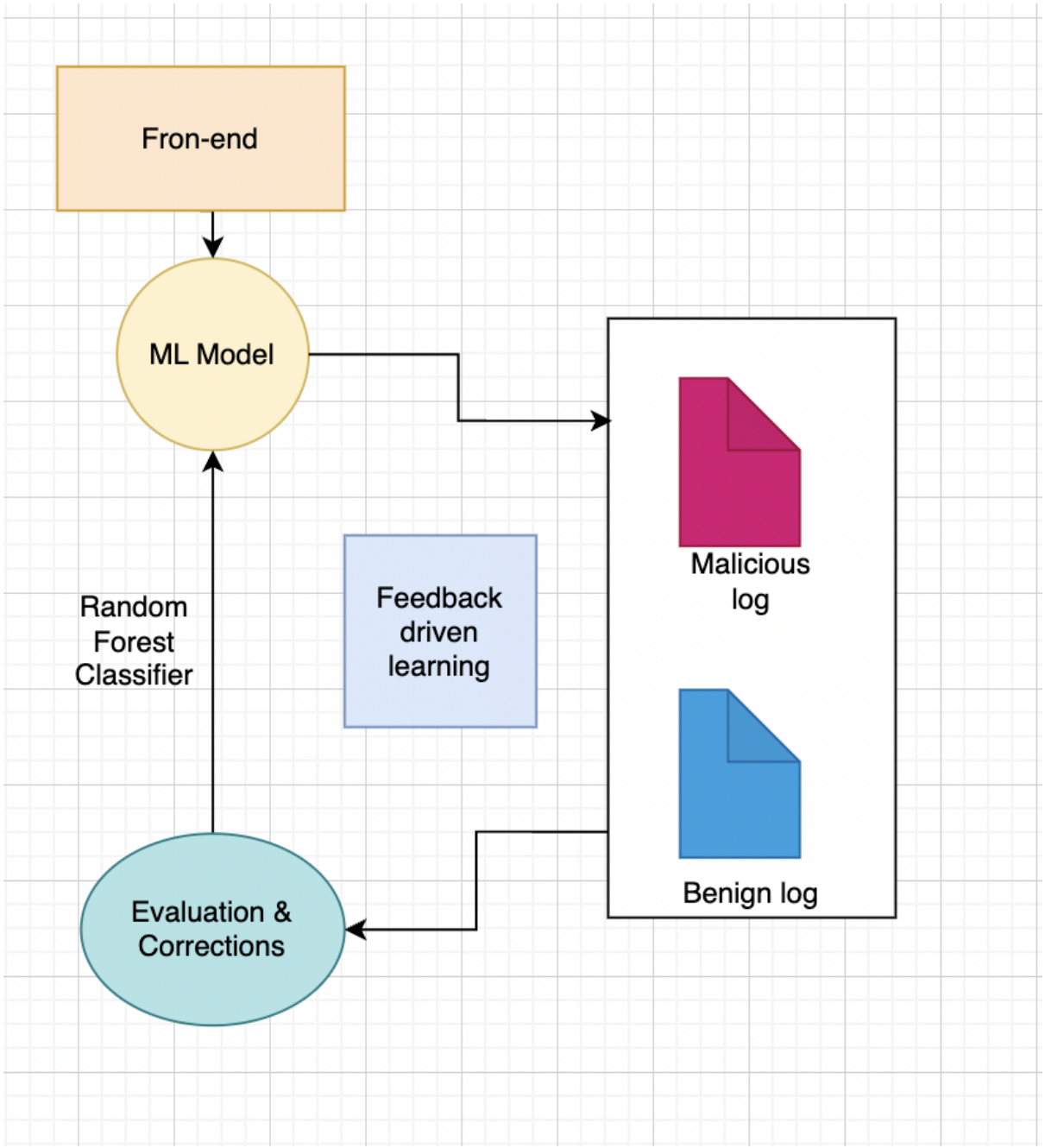

Figure 16: Illustration of the Feedback driven learning model

8 Evaluation

Post-training, the model was tested using the test set. The evaluation focused on the model's accuracy – its ability to correctly classify requests as normal or malicious. Moreover, a classification report was generated, providing detailed insights into the model's precision, recall, and F1 score for each class. These metrics gave us an amble understanding of the model's performance.

8.1 Insights and Outcomes

Initially, the model showed lots of false positive cases as the model was trained with the least dataset. After training with a dataset of over 100+ cases, the model started finding the anomalies of the web application with more accuracy. Some of the test cases and different scenarios are discussed with this report.

Note: The entire process, from data preprocessing to model evaluation, was scripted in Python, utilizing its rich ecosystem of data science libraries. This choice of Python and its associated libraries like Pandas, Scikit-learn, and NumPy enabled us to efficiently handle data processing, model training, and evaluation, making the development process streamlined and effective.

8.2 Experiment / Case Study 1: Malicious Payload Detection

In this case study, our focus was on evaluating the model's effectiveness in identifying malicious payloads. A series of requests, including known attack vectors from the 'Malicious.json' dataset, were sent to the test environment. The model accurately identified a significant majority of these requests as malicious, showcasing its effectiveness in real-time threat detection. A wide range of common vulnerabilities were tried with this step, which includes, CSRF attack, XSS, SQL injection, Command injection etc.

The malicious payloads are chosen from the internet randomly to ensure not the same input is given again. The prediction score was printed to identify the efficiency of finding each dataset.

The SQL injection attack of several payloads resulted in the 0.50 to 0.60 range, which seems to be on the edge, so such cases are identified and trained again with the malicious payload. Refer to fig. 18 for the log generated while predicting the XSS and refer to fig. 17 for the log generated while predicting the SQL injection attack.

8.3 Experiment / Case Study 2: Benign Traffic Analysis

The second case study involved testing the model with normal, benign traffic derived from server logs. This experiment aimed to assess the model's ability to distinguish legitimate traffic from potential threats. The results were encouraging, as the model exhibited a low false positive rate, ensuring that regular user activities were not unjustly flagged as threats. Refer fig. 19 and fig. 20 for the details of the prediction of non-malicious requests.

```
{
    "req": {
        "api": "/api/addBook",
        "authorization": null,
        "request_data": {
            "title": "now",
            "author": "ORDER BY SLEEP(5)#",
            "published_year": "23524",
            "isbn": "242211"
        },
        "cookies": {
            "session": "e0d3f0e6-bfaa-476f-bbfe-087e239e7e1c",
            "JsessionToken": "lib",
            "anti-csrf": "oG-iKBHHtpHzCqHMILxWlA"
        },
        "headers": {
            "Host": "localhost:5000",
            "Connection": "keep-alive",
            "Content-Length": "86",
            "Sec-Ch-Ua": "\"Google Chrome\";v=\"119\", \"Chromium\";v=\"119\", \"Not?A_Brand\";v=\"24\"",
            "Sec-Ch-Ua-Platform": "\"macOS\"",
            "Sec-Ch-Ua-Mobile": "?0",
            "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like (
            "Content-Type": "application/json",
            "Accept": "*/*",
            "Origin": "http://localhost:3000",
            "Sec-Fetch-Site": "same-site",
            "Sec-Fetch-Mode": "cors",
            "Sec-Fetch-Dest": "empty",
            "Referer": "http://localhost:3000/",
            "Accept-Encoding": "gzip, deflate, br",
            "Accept-Language": "en-US,en-GB;q=0.9,en;q=0.8",
            "Cookie": "session=e0d3f0e6-bfaa-476f-bbfe-087e239e7e1c; JsessionToken=lib; anti-csrf=oG-iKBHHt
        },
        "prediction": 0.52,
        "response": "Malicious request detected"
    }
}
```


Figure 17: Predection details of SQL injection vulerability

```
{
    "req": {
        "api": "/api/addBook",
        "authorization": null,
        "request_data": {
            "title": "<xss onbeforescriptexecute=alert(1)><script>1</script>",
            "author": "saff",
            "published_year": "3453453",
            "isbn": "36546"
        },
        "cookies": {
            "session": "e0d3f0e6-bfaa-476f-bbfe-087e239e7e1c",
            "JsessionToken": "lib",
            "anti-csrf": "oG-iKBHHtpHzCqHMILxWlA"
        },
        "headers": {
            "Host": "localhost:5000",
            "Connection": "keep-alive",
            "Content-Length": "124",
            "Sec-Ch-Ua": "\"Google Chrome\";v=\"119\", \"Chromium\";v=\"119\", \"Not?A_Brand\";v=\"24\"",
            "Sec-Ch-Ua-Platform": "\"macOS\"",
            "Sec-Ch-Ua-Mobile": "?0",
            "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Ge
            "Content-Type": "application/json",
            "Accept": "*/*",
            "Origin": "http://localhost:3000",
            "Sec-Fetch-Site": "same-site",
            "Sec-Fetch-Mode": "cors",
            "Sec-Fetch-Dest": "empty",
            "Referer": "http://localhost:3000/",
            "Accept-Encoding": "gzip, deflate, br",
            "Accept-Language": "en-US,en-GB;q=0.9,en;q=0.8",
            "Cookie": "session=e0d3f0e6-bfaa-476f-bbfe-087e239e7e1c; JsessionToken=lib; anti-csrf=oG-iKBHHtpH
        },
        "prediction": 0.25,
        "response": "Malicious request detected"
    }
}
```


Figure 18: Predection details of XSS vulerability

```
{
    "req": {
        "api": "/api/getAllBooks",
        "authorization": null,
        "request_data": {},
        "cookies": {
            "session": "7ea3f27f-cccc-4226-b27a-3cad9bf9c5a9",
            "JsessionToken": "lib",
            "anti-csrf": "BKtYwM2ykhSlADz8MpsL0Q"
        },
        "headers": {
            "Host": "localhost:5000",
            "Connection": "keep-alive",
            "Sec-Ch-Ua": "\"Google Chrome\";v=\"119\", \"Chromium\";v=\"119\", \"Not?A_Brand\";v=\"24\"",
            "Sec-Ch-Ua-Mobile": "?0",
            "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/119.0.0.0 Safari/537.36",
            "Sec-Ch-Ua-Platform": "\"macOS\"",
            "Accept": "*/*",
            "Origin": "http://localhost:3000",
            "Sec-Fetch-Site": "same-site",
            "Sec-Fetch-Mode": "cors",
            "Sec-Fetch-Dest": "empty",
            "Referer": "http://localhost:3000/",
            "Accept-Encoding": "gzip, deflate, br",
            "Accept-Language": "en-US,en-GB;q=0.9,en;q=0.8",
            "Cookie": "session=7ea3f27f-cccc-4226-b27a-3cad9bf9c5a9; JsessionToken=lib; anti-csrf=BKtYwM2ykhSlADz8MpsL0Q"
        },
        "prediction": 0.72,
        "response": {
            "books": [
                {
                    "title": "Wishdon",
                    "author": "Howard",
                    "id": 2324
                },
                {
                    "title": "Mystry of death",
                    "author": "R.H. Robert",
                    "id": 32342
                }
            ]
        }
    }
}
```


Figure 19: Predection details of Normal case 1

```
{
    "req": {
        "api": "/api/addBook",
        "authorization": null,
        "request_data": {
            "title": "Mystry of death",
            "author": "R.H. Robert",
            "published_year": "2023",
            "isbn": "32342"
        },
        "cookies": {
            "session": "7ea3f27f-cccc-4226-b27a-3cad9bf9c5a9",
            "JsessionToken": "lib",
            "anti-csrf": "BKtYwM2ykhSlADz8MpsL0Q"
        },
        "headers": {
            "Host": "localhost:5000",
            "Connection": "keep-alive",
            "Content-Length": "89",
            "Sec-Ch-Ua": "\"Google Chrome\";v=\"119\", \"Chromium\";v=\"119\", \"Not?A_Brand\";v=\"24\"",
            "Sec-Ch-Ua-Platform": "\"macOS\"",
            "Sec-Ch-Ua-Mobile": "?0",
            "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/119.0.0.0 Safari/537.36",
            "Content-Type": "application/json",
            "Accept": "*/*",
            "Origin": "http://localhost:3000",
            "Sec-Fetch-Site": "same-site",
            "Sec-Fetch-Mode": "cors",
            "Sec-Fetch-Dest": "empty",
            "Referer": "http://localhost:3000/",
            "Accept-Encoding": "gzip, deflate, br",
            "Accept-Language": "en-US,en-GB;q=0.9,en;q=0.8",
            "Cookie": "session=7ea3f27f-cccc-4226-b27a-3cad9bf9c5a9; JsessionToken=lib; anti-csrf=BKtYwM2ykhSlADz8MpsL0Q"
        },
        "prediction": 0.63,
        "response": {
            "message": "Book added successfully!!!"
        }
    }
},
```


Figure 20: Predection details of Normal case 2

8.4 Experiment / Case Study 3: False positive cases

This case study focused on false positive cases, which is one of the most important drawbacks that can be considered for the research.

Some of the inputs can be accepted but as it nearly looked like a malicious script, it was identified as malicious by the model. This data is very important for further training, as they are needed to eliminate the major false positive cases in future.


```
"req": {
    "api": "/api/addBook",
    "authorization": null,
    "request_data": {
        "title": "now",
        "author": "SLEEP",
        "published_year": "23524",
        "isbn": "242211"
    },
    "cookies": {
        "session": "e0d3f0e6-bfaa-476f-bbfe-087e239e7e1c",
        "JsessionToken": "lib",
        "anti-csrf": "oG-iKBHHtpHzCqHMILxWlA"
    },
    "headers": {
        "Host": "localhost:5000",
        "Connection": "keep-alive",
        "Content-Length": "73",
        "Sec-Ch-Ua": "\"Google Chrome\";v=\"119\", \"Chromium\";v=\"119\", \"Not?A_Brand\";v=\"24\"",
        "Sec-Ch-Ua-Platform": "\"macOS\"",
        "Sec-Ch-Ua-Mobile": "?0",
        "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like
        "Content-Type": "application/json",
        "Accept": "*/*",
        "Origin": "http://localhost:3000",
        "Sec-Fetch-Site": "same-site",
        "Sec-Fetch-Mode": "cors",
        "Sec-Fetch-Dest": "empty",
        "Referer": "http://localhost:3000/",
        "Accept-Encoding": "gzip, deflate, br",
        "Accept-Language": "en-US,en-GB;q=0.9,en;q=0.8",
        "Cookie": "session=e0d3f0e6-bfaa-476f-bbfe-087e239e7e1c; JsessionToken=lib; anti-csrf=oG-iKBHHt
    },
    "prediction": 0.49,
    "response": "Malicious request detected"
}
```


Figure 21: An example of the False positive case

From Fig. 21, we can see that the input SLEEP was identified as malicious as it is one of the main keywords that can be used by the SQL injection.

Such cases can be accepted but because of the key word, it was identified as malicious. To overcome this issue, the training dataset was updated with such keywords to ensure they represent the benign case.

After updating the model, the input with SLEEP was predicted as normal and the case was not identified as an anomaly which is shown in fig. 22

The study confirms that regularly identifying the false positive cases and training the model, fine tunes the predictability and increases the effectiveness of the anomaly detector.

```
{
    "req": {
        "api": "/api/addBook",
        "authorization": null,
        "request_data": {
            "title": "a;sfja",
            "author": "Sleep",
            "published_year": "342",
            "isbn": "234"
        },
        "cookies": {
            "session": "e0d3f0e6-bfaa-476f-bbfe-087e239e7e1c",
            "JsessionToken": "lib",
            "anti-csrf": "oG-iKBHHtpHzCqHMILxWlA"
        },
        "headers": {
            "Host": "localhost:5000",
            "Connection": "keep-alive",
            "Content-Length": "71",
            "Sec-Ch-Ua": "\"Google Chrome\";v=\"119\", \"Chromium\";v=\"119\", \"Not?A_Brand\";v=\"24\"",
            "Sec-Ch-Ua-Platform": "\"macOS\"",
            "Sec-Ch-Ua-Mobile": "?0",
            "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/119.0.0.0 S
            "Content-Type": "application/json",
            "Accept": "*/*",
            "Origin": "http://localhost:3000",
            "Sec-Fetch-Site": "same-site",
            "Sec-Fetch-Mode": "cors",
            "Sec-Fetch-Dest": "empty",
            "Referer": "http://localhost:3000/",
            "Accept-Encoding": "gzip, deflate, br",
            "Accept-Language": "en-US,en-GB;q=0.9,en;q=0.8",
            "Cookie": "session=e0d3f0e6-bfaa-476f-bbfe-087e239e7e1c; JsessionToken=lib; anti-csrf=oG-iKBHHtpHzCqHMILxWlA"
        },
        "Prediction": 0.66,
        "response": {
            "message": "Book added successfully!!!"
        }
    }
```


Figure 22: Increased efficiency after second training of the model

8.5 Experiment / Case Study 4: Testing with large dataset

In this case, the testing with the enormous dataset has been triggered to the server to see whether the model creates a significant delay that can result in reducing the efficiency of the response.

For conducting the performance test, J-Meter has been used, which is shown in the fig. 23

1000 requests have been triggered and the waiting time was not significant. It concludes that the model prediction has not affected the response time from the server.

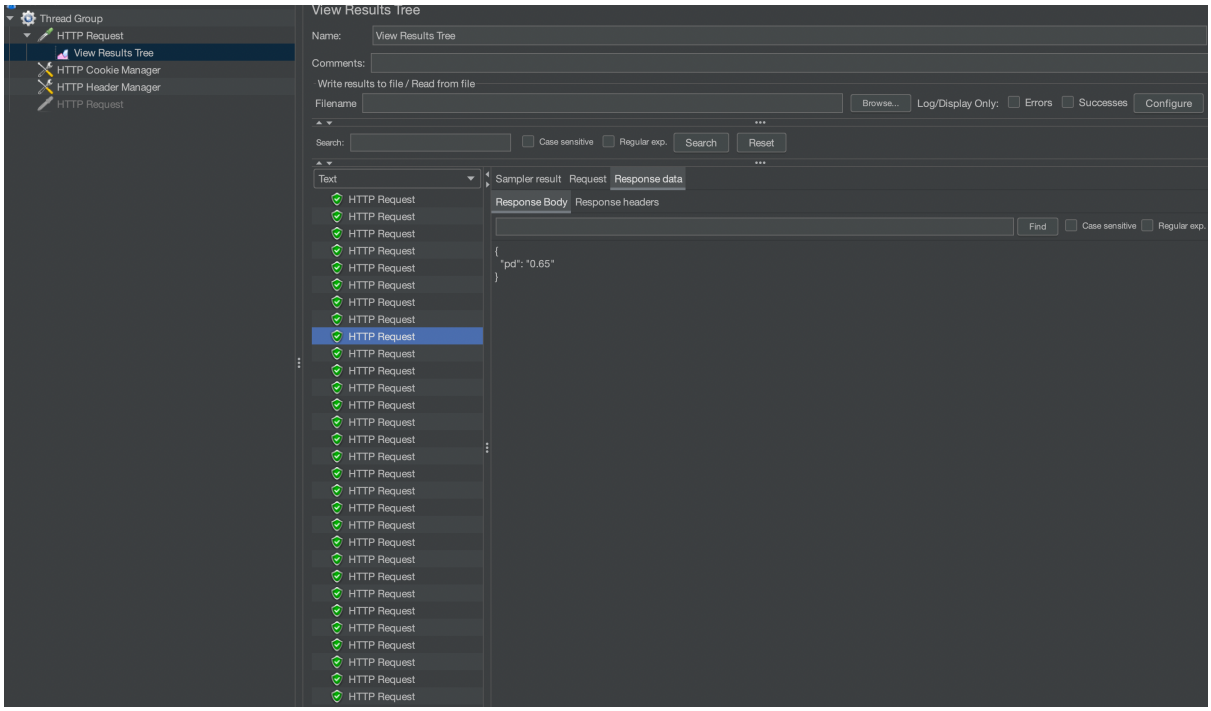

Figure 23: Performance testing results from Jmeter

9 Discussion

Through these experiments, we established the model's efficacy in identifying security threats while maintaining operational efficiency. The separation of benign cases and malicious cases is crucial in terms of continuous learning. If we can successfully implement the model and update it regularly, most of the security threats can be avoided, also it makes sure that no similar threats possibly occur by penetrating the security layer as they are already trained and the model will be at high efficiency to find them.

However, we acknowledge certain limitations in our approach. For instance, the model's reliance on historical data may limit its ability to detect novel attack vectors. To address this, continuous learning and model updating are recommended.

Moreover, the testing environment, while comprehensive, does not fully replicate the complexity of a live production environment. Future work could involve deploying the model in a live setting, under controlled conditions, to further validate its effectiveness.

10 Conclusion and Future Work

In the context of existing literature, our model presents an advancement in automated threat detection, particularly in its application to web APIs. The integration of machine learning with real-time data monitoring represents a significant step forward in developing intelligent, responsive security systems.

our findings suggest that machine learning models, when properly trained and integrated, can significantly enhance the security layer of the application. With the continual updation of the dataset and properly maintaining the benign and malicious cases, at some point, there will be no need for much bothering in terms of security threats to the product.

References

Fahim, M. and Sillitti, A. (2019). Anomaly detection, analysis and prediction techniques in iot environment: A systematic literature review, *IEEE Access* **7**: 81664–81681.

Guezzaz, A., Asimi, Y., Azrour, M. and Asimi, A. (2021). Mathematical validation of proposed machine learning classifier for heterogeneous traffic and anomaly detection, *Big Data Mining and Analytics* **4**(1): 18–24.

Hanif, H., Md Nasir, M. H. N., Ab Razak, M. F., Firdaus, A. and Anuar, N. B. (2021). The rise of software vulnerability: Taxonomy of software vulnerabilities detection and machine learning approaches, *Journal of Network and Computer Applications* **179**: 103009. **URL:** *https://www.sciencedirect.com/science/article/pii/S1084804521000369*

Harer, J. A., Kim, L. Y., Russell, R. L., Ozdemir, O., Kosta, L. R., Rangamani, A., Hamilton, L. H., Centeno, G. I., Key, J. R., Ellingwood, P. M., Antelman, E., Mackay, A., McConley, M. W., Opper, J. M., Chin, P. and Lazovich, T. (2018). Automated software vulnerability detection with machine learning.

Henriques, J., Caldeira, F., Cruz, T. and Simões, P. (2022). An automated closed-loop framework to enforce security policies from anomaly detection, *Computers & Security* **123**: 102949.

Huo, Y., Li, Y., Su, Y., He, P., Xie, Z. and Lyu, M. R. (2023). Autolog: A log sequence synthesis framework for anomaly detection, *arXiv preprint arXiv:2308.09324* .

Institute, P. (2023). Anomaly detection: How to save money and protect your data, *Ponemon Institute* .

Khraisat, A., Gondal, I., Vamplew, P. and Kamruzzaman, J. (2019). Survey of intrusion detection systems: techniques, datasets and challenges, *Cybersecurity* **2**(1): 1–22.

Nassif, A. B., Talib, M. A., Nasir, Q. and Dakalbab, F. M. (2021). Machine learning for anomaly detection: A systematic review, *IEEE Access* **9**: 78658–78700.

Pérez-Vilarelle, L., Sotos Martínez, E. and Yépez Martínez, J. (2022). A comparative study of machine learning algorithms for the detection of vulnerable python libraries, *Computational Intelligence in Security for Information Systems Conference*, Springer, pp. 138–148.

Shen, Z. and Chen, S. (2020). A survey of automatic software vulnerability detection, program repair, and defect prediction techniques, *Security and Communication Networks* **2020**: 1–16.

Vervaet, A. (2021). Monilog: An automated log-based anomaly detection system for cloud computing infrastructures, *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pp. 2739–2743.

Wu, T., Chen, L., Du, G., Zhu, C. and Shi, G. (2021). Self-attention based automated vulnerability detection with effective data representation, *2021 IEEE Intl Conf on Parallel Distributed Processing with Applications, Big Data Cloud Computing, Sustainable Computing Communications, Social Computing Networking (ISPA/BDCloud/SocialCom/SustainCom)*, pp. 892–899.

Zavarzin, D. and Afanaseva, T. (2019). Analysis of software development process in respect to anomaly detection, *in* A. Abraham, S. Kovalev, V. Tarassov, V. Snasel and A. Sukhanov (eds), *Proceedings of the Third International Scientific Conference "Intelligent Information Technologies for Industry" (IITI'18)*, Springer International Publishing, Cham, pp. 80–88.

Zhang, C., Chen, J., Cai, S., Liu, B., Wu, Y. and Geng, Y. (2020). ites: Integrated testing and evaluation system for software vulnerability detection methods, *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pp. 1455–1460.