Final Report

AI-Powered Threat Detection for Networks

Unit: COIT20265
Student 1: Jay Dilipbhai Yadav(12226402)
Student 2: Harshkumar Dharmendrabhai Patel(12218512)
Student 3: Walgama Ranasinghe Arachchilage Akshitha Ishan Ranasinghe (12220598
Student 4: Sonam Pelden(12248722)
Project Mentor: Dr. Fariza Sabrina
Date: 10.02.2025
CQUniversity Australia

Table of Contents

1 Introduction

To increase cybersecurity, the AI-powered threat detection system detects and mitigates cyber threats in real time. This system uses powerful machine learning to detect criminal activity, text-based cyber threats, and security breaches. AWS hosts a scalable, dependable, and available system. With admins, employees, registered users, and unregistered users, the web application offers structured interactions with security. This project uses AI to improve cyber threat detection, reaction time, and user interaction security (Remzi Gürfidan, Ersoy & Oğuzhan KİLİM 2023).

The system contains a community forum for users to post threats, an admin site for threat and user administration, and an employee portal for security staff to monitor activities. Flask's backend leverages Random Forest, SVM, and ANN for accurate threat detection. SQLAlchemy efficiently manages user and threat data. Two-factor authentication, access control, and encryption safeguard critical data.

1.1 System Overview

Threats have increased due to the rapid growth of digital interactions and online platforms, making cyber security a top priority for companies and users. To address these concerns, the AI-powered threat detection system is designed to provide real-time cyber security (Wang, Chen & Yu, 2022). The system uses advanced machine learning to assess data, detect crimes, and prevent security breaches. It boosts speed, accuracy, and response time, making cybersecurity proactive.

The system is designed on a scalable AWS infrastructure, ensuring reliability, availability, and data processing capacity. The web application is designed for administrators, employees, registered users, and unregistered users. various user classes have various access levels, ensuring safe and structured interactions using an intuitive interface. An AI-based cyber security monitoring system protects sensitive data and communication routes against digital threats.

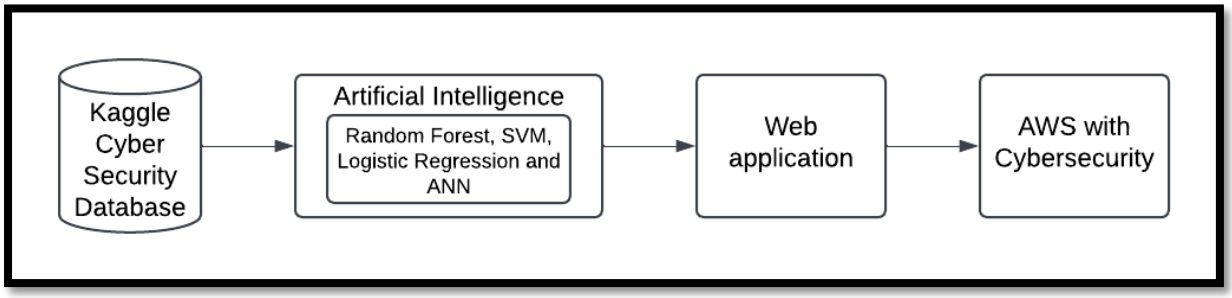


Figure 1 Methodology Block Diagram(self-created)

1.1.1 Description of the Problem from a Business and Technical Perspective

1.1.1.1 Business Perspective

Businesses of all sizes are affected by cyber security threats. Hacks, viruses, phishing, and unauthorised access can cost money, reputation, and legal difficulties. Demand for comprehensive cyber security solutions has never been higher as cyber businesses increasingly rely on digital platforms for operations, communication, and customer interactions. Internet threats affect customer data, IP, and critical infrastructure. Financial and operational issues may arise without real-time threat detection and response. Fraudulent data breaches can lead to GDPR, HIPAA, and PCI-DSS violations in highly regulated businesses including finance, healthcare, and e-commerce (Kala, 2023).

For businesses, manual threat monitoring is time-consuming, ineffective, and prone to error. Security teams routinely handle large amounts of cyber threat data, making it hard to assess risks quickly and respond effectively. Business continuity, operational security, and downtime reduction require an AI-powered automated threat detection system (Corallo, Lazoi & Lezzi 2020). Users expect digital safety.

Any security breach can damage brand reputation and revenue by alienating customers. Cyber threats evolve, so businesses need smart, scalable, defensive security.

1.1.1.2 Technical Perspective

From a technical standpoint, traditional cyber security solutions rely on signature-based threat detection, which is insufficient for zero-day attacks and evolving threats. These conventional techniques struggle to detect freshly evolving threats because they rely on established security rules and recognised attack signatures. The main technical difficulty is reducing cyber threats in real time. Attackers are always improving their strategies, making it essential for security systems to use ML and AI to discover patterns and anomalies in vast data sets (Naik et al., 2021). The AI-powered threat detection system detects NLP-based text-based cyber threats in real time.

Web application scalability, security, and dependability are achieved with RBAC, 2FA, and AWS cloud deployment. The backend employs Flask and SQLAlchemy for data administration, and Random Forest, SVM, and ANN improve detection. For security and performance, the system employs AWS EC2, S3, and VPC. The AI-powered threat detection system solves business and technical challenges to reduce threats, security risks, and cybersecurity compliance.

1.1.2 Key Features and Functionality

A full cybersecurity solution uses AI-powered threat detection system elements. Each component is necessary for system operation. The following elements determine system architecture and performance.

1.1.2.1 Real-Time Cyber Threat Detection

The system detects and eliminates cyber threats. Traditional security systems use predetermined threat signatures, which can't stop new assaults. AI-powered detection learns from new threats, adapts to unexpected assault patterns, and improves detection accuracy.

The system uses NLP to assess text-based cyber threats like phishing, malware, and suspicious user behaviours (Silvestri et al., 2023). It scans user-generated messages for malicious intent to stop them.

1.1.2.2 Role-Based Access and User Management

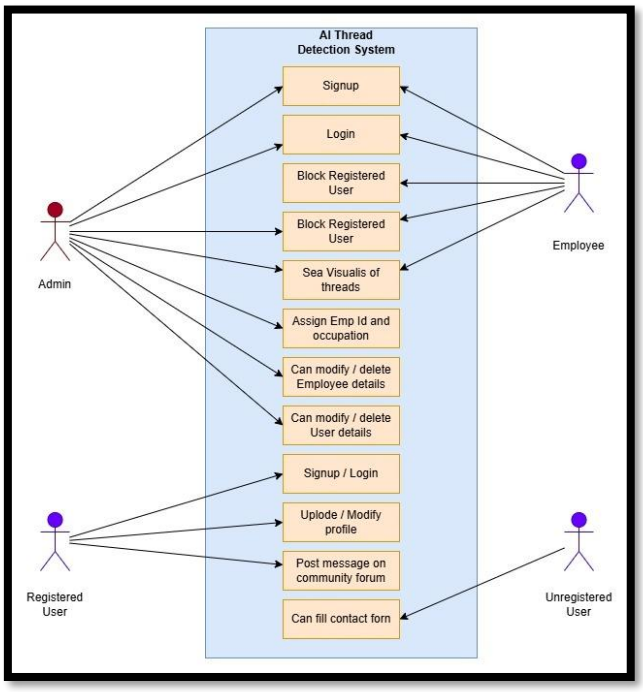


Figure 2 Use Case Diagram(self-created)

To ensure secure and controlled access, the system provides multiple user roles:

- **Administrators**: They have full control over the platform, including user management, security configuration, and reviewing detected threats.
- **Employees**: They are responsible for monitoring flagged activities and taking necessary actions, such as blocking suspicious users.
- **Registered Users**: They can participate in the community forum and interact with other users while their messages are analyzed by the AI-powered threat detection system.
- **Unregistered Users**: They have limited access, primarily to contact the platform for queries or support.

Each role is assigned a set of permissions and security policies, ensuring that only authorized users can perform sensitive operations.

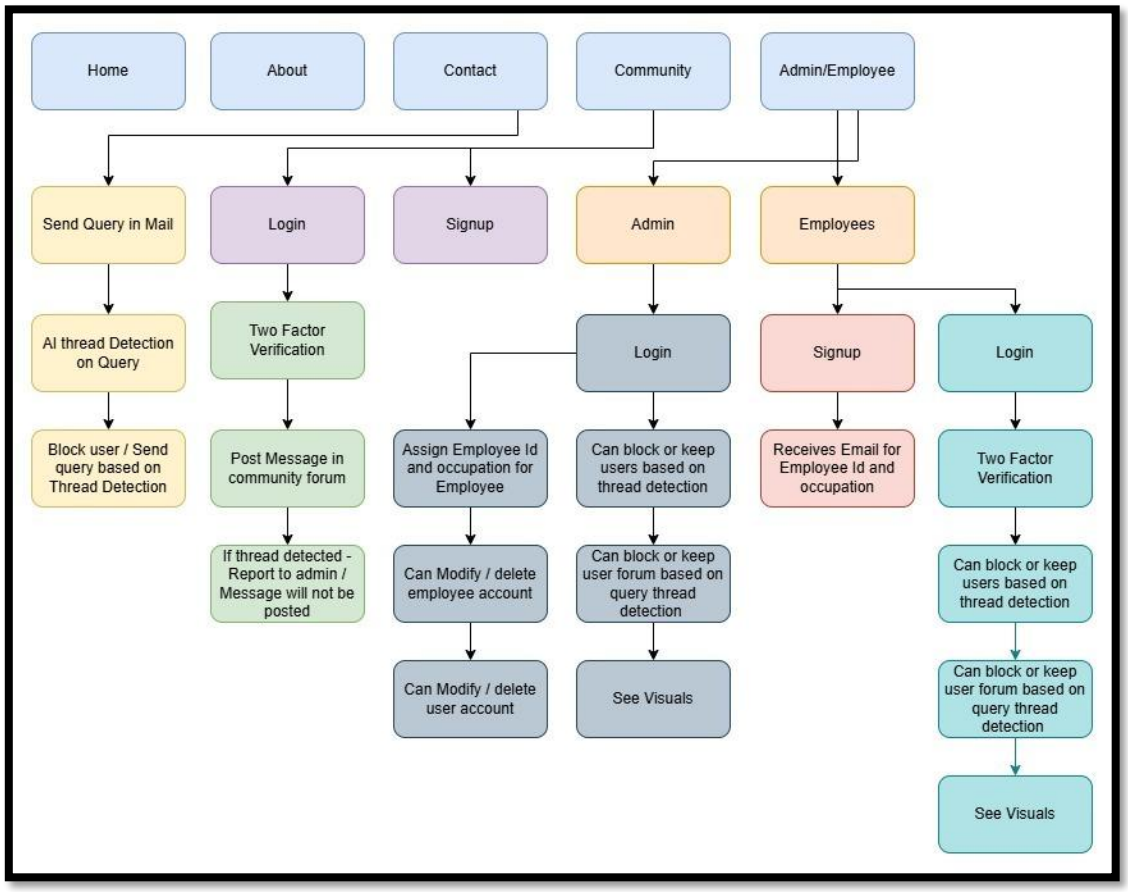


Figure 3 System architecture*(self-created)*

1.1.2.3 AI-Driven Security Measures

The system incorporates advanced AI models to analyze cyber threats with high accuracy. Some of the key machine learning techniques used in the backend include:

Random Forest: A robust classification model used for detecting anomalies in text-based data(Subasi et al., 2017).

Support Vector Machines (SVM): An algorithm that separates malicious content from normal communication patterns.

Artificial Neural Networks (ANN): A deep learning approach that improves the system's ability to detect complex cybersecurity threats.

The backend architecture, built using Flask, enables seamless integration between the AI models and the web application. The system dynamically processes user inputs, allowing real-time response to potential cyber threats (Kadiyala & Kumar, 2022).

1.1.2.4 Scalable AWS Infrastructure

Hosting the system on AWS (Amazon Web Services) ensures high availability, scalability, and security. AWS provides essential services that help maintain a robust and resilient architecture:

Amazon EC2 Instances: Power the backend computations and allow flexible scaling based on system load.

Amazon S3: Secure storage for datasets, logs, and machine learning models.

AWS Backup Services: Ensure disaster recovery by automating data backups and retrieval.

AWS Virtual Private Cloud (VPC): Enhances security by isolating the system within a private cloud environment(Villegas-Ch, Govea & Ortiz-Garces, 2024).

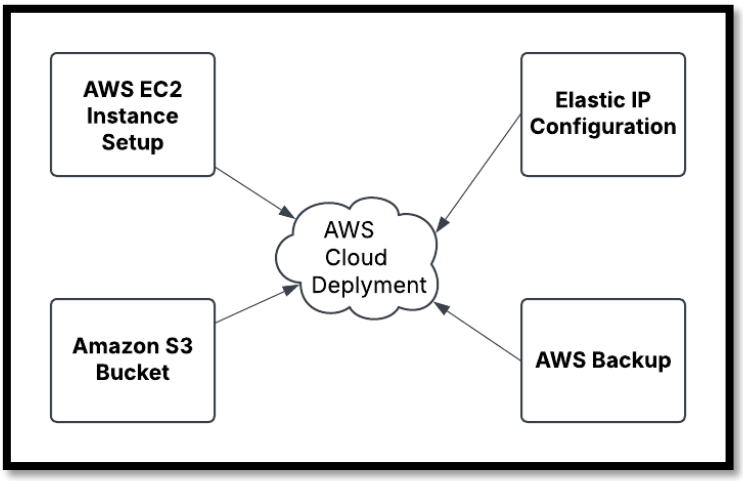


Figure 4 AWS Deployment(self-created)

1.1.2.5 Secure Database Management

A secure SQLAlchemy database is used to store and manage sensitive user and security data (Irungu et al., 2023). The database ensures:

- **Data Integrity**: Preventing unauthorized modifications or corruptions.
- **Efficient Storage**: Optimized data retrieval and indexing.
- **User Authentication and Access Control**: Securely managing user credentials and session authentication.

By implementing role-based database access, the system ensures that only authorized personnel can modify or retrieve specific sets of information.

1.1.2.6 Two-Factor Authentication (2FA)

One of the strongest security features of the platform is the two-factor authentication (2FA) mechanism. This feature enhances login security by requiring users to verify their identity using a one-time password (OTP) sent to their registered email. This prevents unauthorized access, even if login credentials are compromised.

1.1.3 How the System Works

The AI-powered threat detection system follows a well-structured workflow, ensuring seamless threat identification, response, and mitigation.

Step 1: User Interaction

- Users interact with the system through the community forum, contact page, or employee portal. Messages, queries, and responses are continuously monitored and analyzed.

Step 2: AI-Powered Threat Analysis

- Once user-generated content is received, the AI-powered models process the data using text classification, pattern recognition, and anomaly detection techniques. If a potential threat is detected, the system flags the message and alerts an administrator or employee for review.

Step 3: Action-Based Response

Based on the AI's classification:

- If a message is safe, it is approved and posted in the forum.
- If a message contains a threat, it is flagged for further review.
- If malicious intent is confirmed, the user account may be blocked or restricted from further participation.

Step 4: System Security and Logging

- Every action is logged in the security database, ensuring audit trails and compliance with cybersecurity policies. The system can generate reports on detected threats, helping cybersecurity teams assess trends and improve the detection algorithms.

1.1.3.1 Security Enhancements and Preventive Measures

The AI-powered threat detection system is built with a multi-layered security approach to minimize vulnerabilities and protect against cyber threats.

- **End-to-End Encryption** – Ensures secure communication between the user interface and the backend.
- **Regular Security Updates** – AI models are continuously updated to improve threat detection.
- **Real-Time Logging and Monitoring** – Keeps track of user interactions to identify suspicious activities early.
- **Access Control Mechanisms** – Restrict unauthorized users from making system modifications.
- **Automated System Backups** – Ensures that data loss is prevented in case of security breaches or system failures (Bhadouria, Bathla & Arora, 2024).

A comprehensive AI-powered threat detection system detects, prevents, and reduces cyber threats in real time. Machine learning, complex AI models, AWS cloud services, and strong security safeguard digital interactions throughout the system. Two-factor authentication, role-based access management, and scalable infrastructure boost platform accuracy, efficiency, and reliability.

Real-time monitoring, automated threat detection, and organised security policies make the system a potent cyber security platform. Phishing, malware, unauthorised access, and other cybersecurity threats are prevented by this solution. The adaptive and scalable AI-powered threat detection system protects and enhances cybersecurity as cyber threats evolve.

1.2 Delivered Technical Artefacts

Name	File	Description	PDF?
Literature Review	group1- literature-review-report.docx	Review of existing/competing products, solutions, or literature.	No
Methodology	group1- methodology-report. docx	Reports the methodology of AI, Web and AWS component.	No

Design of Machine Learning Algorithms	group1- design-of-machine-learning-algorithms-report. docx	Detailed explanation for design of machine learning algorithms.	No
Design of web application	group1-design-of-web-application-report. docx	It has design of web application, database, usability, performance etc.,	No
Risk Assessment	group1-risk-assessment-report. docx	This report has risk assessment table for the proposed system.	No
Network Securities and Policies	group1-network-securities-and- policies-report. docx	Network securities and policies for AWS.	No
Setup Guidelines and Implementation	group1-setup-guidelines-and- implementation-report. docx	This provides step by step guidance for the setup for AI, Web and AWS.	No
Black Box Testing for the Web Application	group1-black-box-testing-for-the-web-application-report. docx	It has black box testing for the web application.	No
Recommended Security Controls	group1-recommended-security-controls-report. docx	This report has detailed recommended security controls for the proposed system.	No
AWS Deployment	group1-AWS-deployment-report. docx	This report has step-by-step deployment for the web application in AWS.	No
AI component Code	group1-AI-threat-detection-code.ipynb	This has AI code using python and its format is ipynb.	No
Web application Folder(source code)	group1-web-application-source-code	This is source code for the web application.	No

1.3 Contributions

Student Name	Percent	Summary of Contributions	Technical Lead on Artefacts
Jay Dilipbhai Yadav	25%	Contributed the following Data Collection, Data Sampling, Data Cleaning, Text Preprocessing and Model Deployment in AI component. And contributed the following Contact Us Page Setup, Form Submission Handling, Threat Detection, Threat Logging, Automatic Blocking, Email Notification, Message Feedback, Flask Application Setup, Database Modelling Admin and User Functionality in Web application development. AWS EC2 Setup and Configuration, Integration of Backend Components Performance Optimization and Future Planning are contributed to AWS component.	AI component. Web application development. AWS setup. Risk Assessment Report. Recommendations of Security Controls Report. Design of Machine Learning Algorithms Report. Design Of Web Application Report, Methodology Report.
Sonam Pelden	25%	Implemented the following in AI, and there are Feature Engineering, Label Encoding, Model Building. Implemented OTP Verification System, Admin Page Development, Threat Detection Logging, Data Visualization for Admin, Employee Management in Web application component. Configures the AWS like AWS VPC Deployment, Elastic IP Configuration.	AI component. Web application development. AWS setup. Network Securities & Policies Report. Design of Machine Learning Algorithms Report. Design Of Web Application. Report, Methodology Report. Literature Review.
Harshkumar Dharmendrabhai Patel	25%	Implemented and configured the following Exploratory Data Analysis (EDA), Train-Test Split, Community Forum Development, User Authentication System, Password Reset System, User Logout System Signup System with Validation, Admin Panel Development, Employee & User Management, Data	AI component. Web application development. AWS setup. Design of Machine Learning Algorithms Report. Design Of Web Application Report, Methodology Report. Set-up and Guidance Report. AWS Deployment

		Backup & Disaster Recovery, Threat Visualizations and AWS Deployment - Backup System.	
Walgama Ranasinghe Arachchilage Akshitha Ishan Ranasinghe	25%	The following steps are contributed in the all three components, Model Evaluation, Model Comparison, Employee Module - Signup & Login, Admin Dashboard & Employee Management, User Management - Blocking & Unblocking, Community Forum, Password Recovery (Forgot & Reset Password), AWS S3 Integration for File Storage, AWS Backup Configuration and Testing of Web Application.	Literature Review, AI component. Web application development. AWS setup. Design of Machine Learning Algorithms Report. Design Of Web Application Report, Methodology Report, Black Box Testing for the Web Application Report.

1.4 Next Steps

The AI-powered threat detection system scales nicely. However, enhancing and expanding the system might enhance its effectiveness. Adding features like autonomous threat response will enhance the system's capabilities. The software automatically eliminates threats by advising or executing predetermined actions. Managers could avoid security breaches with machine learning-based predictive analysis. The system limits users and visualises threats. Role-specific dashboards with extensive filtering may increase usability and operational effectiveness.

Focus on testing and optimising unimplemented parts. Underexplored features include real-time multi-language Community Forum support and admin panel statistics. The password reset and OTP systems are operational, but stress testing would ensure durability under high user demand. A more detailed user behaviour and system abnormality logging and reporting system would promote transparency and accountability. Rigid user testing across all roles will identify and fix usability issues systemically to provide a smooth user experience.

Finally, introducing system components beyond the project scope may be beneficial. AI-driven network traffic anomaly detection would enhance text-based threat detection. The accessibility and user engagement of web platform mobile apps may improve. Future generations could employ blockchain to immutably log flagged messages and user activity, improving data security. The client can improve system usability, scalability, and agility to handle growing cybersecurity challenges by implementing these recommendations.

2 Literature Review

2.1 Introduction

The growing dependence on cloud computing has revolutionized the way companies' function, allowing for streamlined scalability and adaptability as needed. Nevertheless, the transition has also brought about new and major security problems, and in higher percentage of data breaches happened in cloud environments (Reddy, 2021). AI-powered threat detection is a successful method to better cybersecurity through detecting, analysing, and responding to security threats in real time. The literature review critically attempts to analyse the previous studies on threat detection using AI, highlighting the methodologies adopted, the outcomes and limitations of these studies while contrasting between several approaches to present a holistic analysis.

2.2 AI in Network Security and Threat Detection

Identifying and mitigating the evolving cyber threats in real time has become a necessity driven by the advancement of security mechanisms. Existing security solutions, like signature-based intrusion detection systems (IDS) and rule-based firewalls, have failed against advanced and targeted attacks, including zero-day vulnerabilities and advanced persistent threats (APTs) (Wang, Chen and Yu, 2022). All of these methods utilize data science and machine learning (ML) approaches to assist with anomaly detection, threat classifications, or threat rankings. Below figure (Figure.5) will show talk about use cases where these AI-driven technologies are put to work.

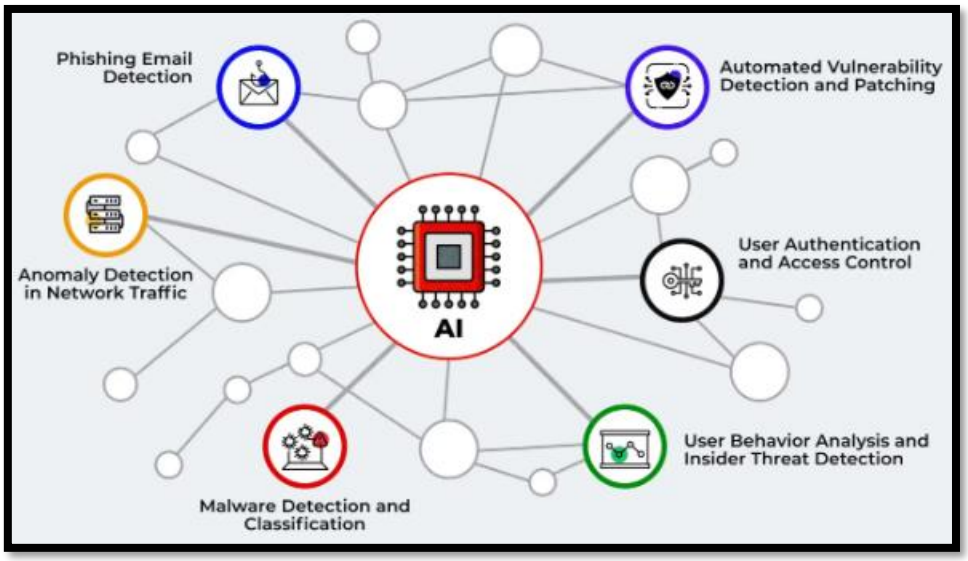


Figure 5 Use case of AI in cybersecurity (Panarin and Bairyev, 2023)

2.3 Machine Learning-Based Intrusion Detection Systems

Intrusion Detection Systems (IDS) are an essential factor for recognizing malicious activities in the network through pattern recognition in data traffic. A few have reported on incorporating AI into IDS aiming to improve detection accuracy and lowering false positives.

A study by Anton and Sinha (2019) proposed a supervised learning method where they used Random Forest (RF) and Support Vector Machine (SVM) classifiers for network intrusion detection. SHI et al. also employed RF and SVM classifiers in their experiment and with an overall accuracy of 94% on the NSL-KDD dataset, RF outperformed SVM in precision and recall.

One of the greatest limitations highlighted is reliance on labeled datasets, which do not always transfer to real-world cyber-attacks. In contrast, Al-Sanjary et al. (2020) proposed an unsupervised learning

method-based k-means clustering analysis approach, implemented by using autoencoders for the network traffic anomaly detection on the network traffic without using any labelled data. Their approach, which did not require the use of predefined signatures, also made them effective at detecting zero-day attacks. But they noted that their system had a higher false positive rate, through which benign network anomalies were misclassified and flagged as threats.

Abuali, Nissirat and Al-Samawi (2023) shows a comparative study proposed Hybrid AI based Integrated IDS using supervised and unsupervised approaches. Their approach uses a Convolutional Neural Network (CNN) to construct feature representations, which is paired with a Long Short-Term Memory (LSTM) network that examines this feature representation for temporal patterns found in the network traffic. The result exceeded the accuracy rate of traditional ML models with 96%. The study found that although it had its successes in the past, real-time deployment in data lacks availability on devices constrained by resources due to the high computational cost.

2.3.1 Deep learning approaches for threat detection

In recent years, deep learning (DL) techniques have been proposed for cybersecurity because they can automatically extract a little number of features of data points from huge amounts of network traffic data and identify more complex attack patterns. In this regard, a number of studies have shown that DL models yield efficient results in network security applications.

A study by Alshomrani et al. (2024), presented a transformer-based model for the detection of real-time network anomalies. It achieved performance at an F1-score ranging from 82% to 99% when tested on a series of datasets and proved to outperform conventional Recurrent Neural Networks (RNNs). The study, however, noted that Transformer models demand are also high on computational resources which hampers their application in real-time threat-monitoring systems. Likewise, Dhadhania et al. (2023) used GNN to identify threats in network traffic. Their model exploited the inherent connection within the data packets to capture the correlated attack behaviour. Although the study reported 97% detection rate, it also stated that GNN-based models need a lot of labelled datasets which are not always present in different sectors such as cybersecurity.

2.3.2 AI for Automated Threat Response

In addition to threat detection, AI-powered security frameworks are integrating automated response capabilities to reduce risks in real-time. A study by Louati, Ktata and Amous (2024) proposed RL-based cybersecurity defense of IPs based on firewall and access control rules dynamic change when a threat is detected. This resulted in an RL model that achieved 78% lower attack success rates than static security policies. However, according to the study, ensuring that the model can adapt to novel attack strategies poses some challenges. AI-based pouring SOAR systems for advanced integrated for cloud-based threat intelligence platforms proposed by Chen and Zhang (2024). Their goal was a system that could automate incident response workflows to reduce the time to detect and mitigate threats by 65% on average. However, the research noted that false-positive events could result in unwanted security responses, which could in turn disrupt normal business practices.

2.4 AI powered threat detection in cloud computing

In the modern world of enterprises, they are now the backbone of their operations, providing cloud-based, scalable, and cost-effective elements necessary for data processes and storage. Nevertheless, traditional security approaches are not enough anymore, as the increasing popularity of multi-cloud and hybrid cloud environments (Figure.6) has resulted in critical security risks (Nwachukwu, Tunde and Uzoma, 2024). These security challenges have led to the development of AI-Powered threat detection that is crucial in maintaining real-time threat identification, identifying misconfiguration & attacking mitigation before they escalate.

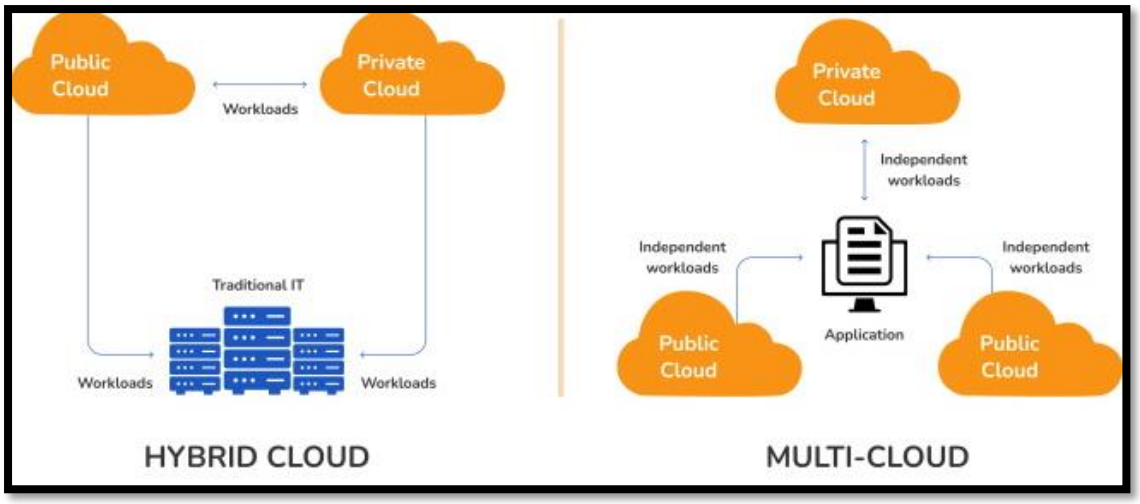


Figure 6 Multi Cloud vs Hybrid Cloud (Barnes, 2022)

2.4.1 AI for Cloud Intrusion Detection and Prevention

Having Intrusion Detection System (IDS)/Intrusion Prevention System (IPS) is the fundamental aspect of cloud security, these systems look for malicious network traffic. Conventional IDS approaches depend on rule-based and signature-based detection, which have difficulty keeping pace with the changing landscape of cyber-attacks. Conversely, AI-based IDS/IPS solutions base their phenomena discovery on heightened ML and DL models that distinguish anomalous and known attack patterns, etc.

A study by Sajid et al. (2024) proposed a hybrid IDS, using Deep CNN for feature extraction and LSTM model for sequential pattern analysis of cloud traffic. Their method obtained an F1-score of 95% on the CICIDS2017 dataset which considerably improved the detection accuracy. Nevertheless, such deep learning models are resource-intensive, which may incur additional costs in cloud platforms. Unlike Hasham, Tariq and Lu (2024) proposed an unsupervised learning method based on the auto-encoders to detect anomalies in multi-cloud, The proposed system achieved a 92% detection rate to zero-day attacks, indicating a high rate of adaptability. However, they also noted a high false-positive rate the model sometimes misclassified legitimate traffic fluctuations for security incidents.

Azam, Islam and Huda (2023), conducted a comparative study proposed a reinforcement learning-based Intrusion Detection System, which adaptively tuned its detection thresholds in response to changing traffic patterns. This model achieved a 30% reduction in false positives as compared to traditional ML approaches with the detection accuracy of 94%. While they have shown high effectiveness, the study pointed to the need for constant retraining of reinforcement learning models, which creates a potential latency problem for real-time threat detection.

2.4.2 Artificial Intelligence for Cloud Misconfigurations and Compliance Monitoring

Misconfigurations in cloud environments are responsible for many security incidents, due to organizations being unable to properly configure access control, encryption settings, and identity management. Automation solutions via AI have increasingly been used to detect and fix cloud misconfigurations.

A study by Stutz et al., (2024) developed an AI-powered compliance monitoring tool which analysed settings of cloud infrastructure and detected policy violations. The tool is built into the APIs at cloud service providers and reported detecting and correcting misconfigurations in 82% of test cases. But, the study noted, API-based security tools would struggle with decoded data as it is encrypted, and that means that some vulnerabilities could remain undetected. Likewise, Malaiyappan et al. (2024) used federated learning for cloud compliance auditing, enabling several organizations to jointly train models

for security without exchanging sensitive information. They reported a 78% drop in cloud misconfiguration incidents. Although effective, this study showed that synchronization and communication overhead represent important problems in the context of federated learning models. In contrast, Mbah and Evelyn (2024) applied NLP techniques to the analysis of cloud security policies for detecting inconsistencies. Their AI-powered policy auditor flag deviations from the policy in 89% of misconfigured cloud environments. But they added that NLP models need regular updates in alignment with ever-evolving cloud security standards like the General Data Protection Regulation (GDPR) and the California Consumer Privacy Act (CCPA).

2.4.3 Incident Response and AI-Powered Threat Intelligence

This led to the development of cloud-based threat intelligence software that used AI to analyse tons of threat data, find malicious patterns, and deliver security insights. Unlike traditional Security Information and Event Management (SIEM) systems, AI security software use predictive analytics to provide the ability to predict and proactively prevent attacks before they occur.

Marri, Varanasi and Chaitanya (2024) investigated how AI-driven threat intelligence can be implemented within cloud systems through machine learning techniques integrated into real-time security analytics. On their platform they detected ransomware attacks with 97% accuracy and reduced incident response times by 60%. However, the researchers also observed that access to diverse threat datasets is crucial for AI-driven threat intelligence solutions but is challenging to obtain because of privacy and regulatory constraints. The study by Chaganti, (2024) found that Generative Adversarial Networks (GANs) could be used to simulate cyberattacks and used to train AI security models. Their findings showed a 90% improvement in the AI model's ability to identify new threats. Nevertheless, the study cautioned that attackers may use adversarial learning to evade AI-based defences. However, Komaragiri and Edward (2022) took a different approach, analysed an enriched AI-driven SOAR (Security Orchestration, Automation, and Response) platform for cloud environments. They automated threat mitigation workflows that removed 75% of the need for manual intervention. Automation can also lead to false positives if it relies too much on automation, causing normal and non-malicious cloud operations to be treated as security threats.

2.5 Real-Time Threat Detection in Communication Systems

The communication systems are an integral part of various digital systems utilized today, that allow for seamless data communication via networks. However, as we depend more and more on cloud-based messaging platforms, email services, and corporate communication channels, the threat landscape has broadened dramatically. Phishing, malware propagation, and insider threats are just some of the tactics cybercriminals use to exploit vulnerabilities in real-time communication systems (Alzaabi and Mehmood, 2024). AI-based systems that use machine learning to detect threats in real time and mitigate the risks they pose by identifying malicious activity, unauthorized access and anomalous behaviour are an emerging promise in this area.

2.5.1 AI-Driven Anomaly Detection in Communication Systems

One of the most common techniques to discover this suspicious activity in communication systems is anomaly detection. ML (Machine learning) models work in real-time on dynamic data streams to detect deviations from standard behaviour. A study by Edozie et al. (2025) proposed an unsupervised anomaly detection model using autoencoders that adapts to enterprise communication logs. Their approach accurately flags potential insider threats 91 percent of the time. The study found that 8% of inferences the ability to recognize normal user behaviours from malicious intent for each identity detection was accompanied by false positive rates as the categorization of them as legitimate user inputs proved difficult. For email threat detection in real time, Nwoye and Nwagwughiagwu, (2024) analysed a hybrid model that combines Random Forest and LSTM networks. Method of spear-phishing

attacks detection with 94% accuracy rate, significantly beaten in the context of the existing spam filters. The authors observed that deep learning-based models depend on large quantities of labelled datasets, which is not always easily available.

In contrast, Mothukuri et al. (2021) introduced a decentralized communication networks and built a federated learning-based anomaly detector. This does provide greater privacy as now training AI models could potentially all be done locally without the need to send over security-sensitive raw data, thus mitigating potential data leaks. While this has the advantage of reducing the computation burden, the study showed it to be computationally more costly compared to centralized learning models overall.

2.5.2 NLP-Based Threat Detection in Communication Systems

Using the NLP Natural Language Processing (NLP) to detect malicious intent in text-based communications is one of the prevalent areas of research. Text analytics is one solution that is able to analyse phone, app, and other forms of text-based communication to flag inappropriate information which could assist in technical threats like phishing, hate speech, and scam messages. Kasri et al. (2025) devising a solution based on a transformer-based NLP model to detect and alert on phishing emails and social engineering attacks in an embedded form of detection and alert mechanism. Their F1-score was 96% but it takes continuous fine tuning to align to new attack patterns.

In contrast, the approach taken by Arazzi et al. (2023). In their work they employed sentiment analysis along with Named Entity Recognition (NER) to identify insider threat in corporate chat applications. Their system detected high-risk messages in 87% of test cases but performed poorly when assessing implicit threats, malicious intent was not explicitly stated.

2.6 Limitations and Challenges

While AI has made some impressive strides, there are still limitations and challenges preventing the widespread adoption of AI tools such as machine learning for threat detection in network infrastructures, cloud computing, and real-time communication systems. Some of the most critical issues includes the high false-positive and false-negative rates generated within an AI based security models. Although, anomaly detection and machine learning-based detection methods enhance the identification of threats, they still incorrectly label legitimate traffic as suspicious or fails to identify sophisticated attacks and leads to operational downtime and security failures (Jeffrey, Tan and Villar, 2023).

The next big challenge is computational overhead of deep leaning models for security analytics. Threat detection systems powered by AI need massive computing power and constant training on large datasets to be effective. However, this imposes a high demand for computational resources, making real-time detection expensive, particularly in cloud environments where scalability is the foremost issue (Akinbolaji, 2024). Additionally, the rise of adversarial attacks against AI models presents a concern, where cybercriminals can manipulate the input given to an AI to prevent detection. So, it draws attention to the importance of building more secure and robust AI models, as adversaries can generate adversarial samples to defeat AI-based security systems.

Privacy and regulatory compliance also pose major issues. AI-based security models need to focus on huge amounts of users' data and network logs that gives a cause of concern for data privacy with compliance and legislation like GDPR and CCPA (Habbal, Ali and Abuzaraida, 2024). Many AI models also work like “black boxes,” making it hard to interpret how they reach their decisions. The absence of transparency lowers the level of trust in AI-powered security alerts and consequently impairs their acceptance for mission-critical cybersecurity operations (Habbal, Ali and Abuzaraida, 2024).

2.7 Summary

AI-empowered threat identification has revolutionized contemporary cybersecurity, making way for improved security analysis, intrusion detection, and threat knowledge. However, the methods of machine learning, deep learning and NLP-based techniques used are still not as effective in identifying such security threats in network security, cloud computing and real-time communication systems. Nonetheless, challenges embedded in false positives, computational overhead, adversarial attacks, and privacy issues still limit wide adoption.

To mitigate these limitations, future work should focus on developing more explainable AI models, making security systems more robust against adversarial examples, and evaluating AI performance in human-in-the-loop frameworks to improve threat mitigation. Moreover, novel approaches such as federated learning and various privacy-preserving AI methods would allow for a greater balance between security objective fulfilment and compliance-related requirements. By overcoming these challenges, AI-enabled threat detection can become a more effective and reliable cybersecurity tool, providing better protection against new cyber threats in cloud environments and digital communication systems.

3 Methodology

3.1 Introduction

The methodology chapter presents a structured approach to constructing an AI-powered threat detection system to identify and remediate cybersecurity vulnerabilities. AWS-hosted Flask web applications and strong machine learning models provide scalability, efficiency, and security in the proposed system. Kaggle Cyber Security Database supervised machine learning uses text and network traffic data. For model training and evaluation, data cleaning, normalisation, EDA, and feature engineering ensure high-quality data. Random Forest, SVM, Logistic Regression, and ANN are four machine learning models that classify cyber risks well. These models are easily integrated into a user-friendly web application during system architecture. S3 buckets, AWS Backup, and Amazon EC2 instances provide secure instance storage, deployment, and disaster recovery. Flask allows the backend to communicate with ML models, while HTML, CSS, and JavaScript make the frontend interactive. This chapter explains how advanced AI, resilient AWS services, and secure web construction practises produce a scalable and efficient threat detection system for real-time insights and proactive cybersecurity.

The block diagram shown in Figure 3.1 defines the whole framework of the overall proposed project starting from the data source which is Kaggle Cyber Security Database. This database serves as the building block for the entire pipeline wherein training and testing of AI models is possible. In the first step that is data cleaning, the goal is to remove duplicates, inconsistencies, irrelevant entries to make sure high-quality data is being used for analysis. Next preliminary processing involves the text being normalized by converting to lowercase, eliminating special characters, and lemmatizing. After that an exploratory analysis, examines patterns and trends that are essential for insights. The TF-IDF Vectorizer transforms text into numerical format, quantifying terms as per their relevance across documents as a metric for machine learning.

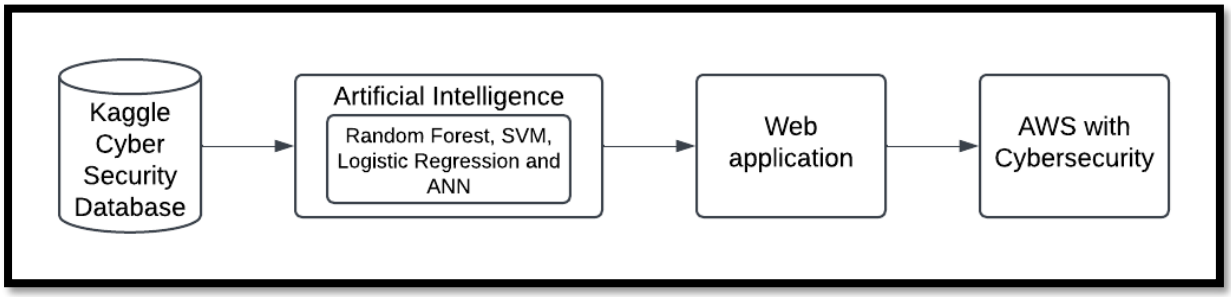


Figure 7 Methodology Block Diagram(self-created)

Cyber security threat detection and prediction statements using Artificial Intelligence component integrates several algorithms such as Random Forest, SVM, Logistic Regression, and ANN. This intelligence is embedded inside a web application hosted on AWS and implements cybersecurity best practices to dynamically and securely manage threats. As shown in Figure 7, this interconnected flow enables seamless processing of data, threat prediction, and secure deployment, which results into an efficient and scalable cybersecurity solution.

3.1.1 Dataset Description

Dataset Link: https://www.kaggle.com/datasets/ramoliyafenil/text-based-cyber-threat-detection/data

To identify and mitigate cyber threats, this project's dataset combines text data with network traffic information. The big dataset facilitates supervised and semi-supervised machine learning for robust threat identification and categorisation. Multi-modal analysis helps understand cybersecurity data's complex relationships and interactions. The dataset's primary parts:

id: Each dataset instance is identified uniquely. It tracks records and ensures data integrity during processing and analysis.

text: This column provides natural language cyberthreats. Malware, phishing, and software flaws are attack vectors.

Entries: This field holds network entity JSON objects. Important traits:

- **sender_id:** Unique communication initiator identifier.
- **label:** Categories threats as malware, attack style, or benign.
- **start_offset and end_offset:** Things in the text are positioned.
- **receiver_ids:** The communication targets entity IDs.

relations: Tuples representing entity connections like sender-receiver linkages. Critical insights on how threats move across a network come from these links.

diagnosis: Threats are thoroughly described in this column. Context helps security specialists understand each threat's severity and repercussions.

solution: Software updates, security controls, and configuration modifications are recommended mitigations. The dataset is diagnostic and actionable due to these recommendations.

The dataset's multi-modality suits numerous cybersecurity applications. Researchers can use it to develop cutting-edge machine learning models to identify cyber threats, analyse threat trends, and implement mitigation measures . NLP, graph-based analysis, and anomaly identification are supported by its extensive textual and structural data.

3.1.2 Threat Detection Using AI

AI is changing the game in cybersecurity by allowing automated threat detection and response. In this project we are focusing on the development of AI-based threat detection system that is working on machine learning techniques to gather insights in the cyber security data. We will preprocess and clean the data, Explore it and then apply classification models on this preprocessed data so we can build a model which can identify different types of attacks as effectively as possible. In Figure 8 given below a detailed structure of the flow diagram of AI Component is shown.

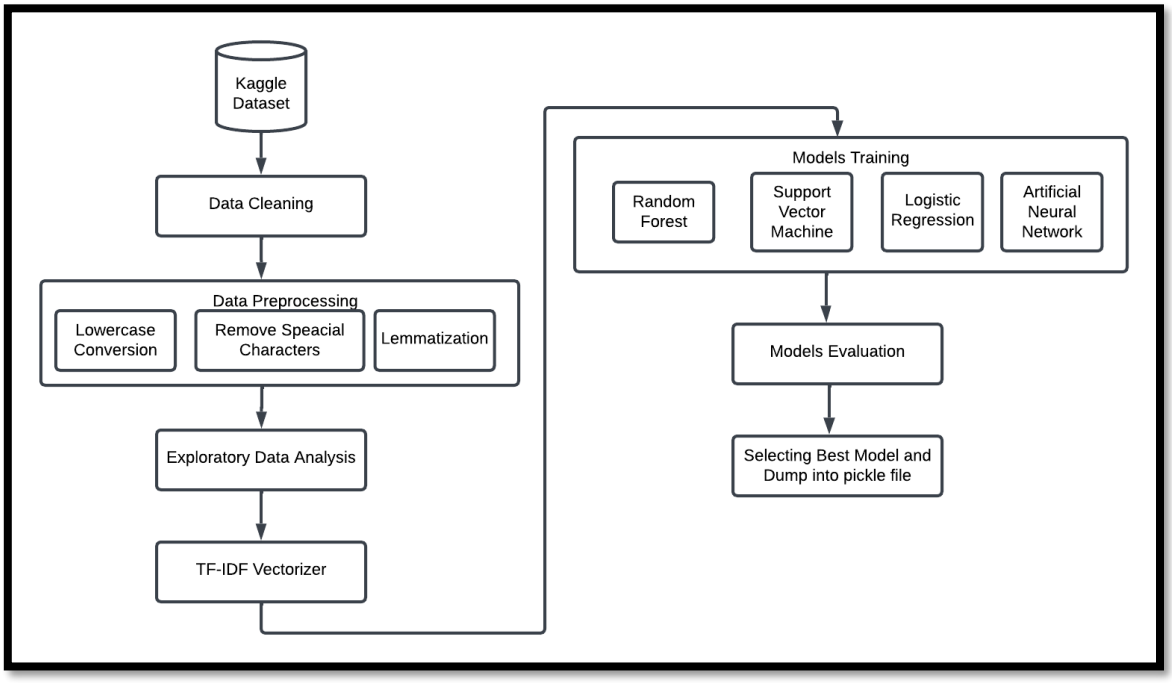


Figure 8 Flow diagram of AI Component(self-created) (self-created)

The flow of AI components includes following steps

- Data Cleaning and Data Preprocessing
- Exploratory Data Analysis (EDA)
- Feature engineering
- Model Training and Model Evaluation.
- Selecting the Best Model and Dumping the Model

3.1.3 Data cleaning and Data Preprocessing

This is the very first step after loading data. Sometimes data set contains irrelevant columns, information, values etc. in it. So, the first task is to clean the data by following different approached. From entire data set only two columns are important so only extracted text and label columns. Now text column contains irrelevant content information. Thus, to remove this unwanted information a function is created which first convert text to lower case, then remove all the stop words like ('is', 'the' etc.) and then lemmatize the content to their root form. In this way a cleaned text is extracted using this function (Symeonidis, Effrosynidis & Arampatzis 2018).

```
def preprocess_text(text_column):
    """
    preprocessing the text column
    """
    stop_words = set(stopwords.words('english'))
    lemmatizer = WordNetLemmatizer()
    def clean_text(text):
        # lowercase conversion
        text = text.lower()
        # remove special characters, numbers
        text = re.sub(r'[^a-z\s]', '', text)
        # remove stopwords and lemmatize words
        text = ' '.join(
            lemmatizer.lemmatize(word)
            for word in text.split()
            if word not in stop_words
        )
        return text

    return text_column.apply(clean_text)

# apply preprocessing
data['clean_text'] = preprocess_text(data['text'])
text_col_index = data.columns.get_loc('text')
data.insert(text_col_index + 1, 'clean_text', data.pop('clean_text'))
data.head()
```

```
# Drop rows with any null values
data.dropna(inplace=True)

print(data.isna().sum())
```

```
#check for missing values in all the columns
print(data.isna().sum())
```

Figure 9 Data cleaning and Pre-processing (self-created)

Figure 9 code illustrates the process of handling missing values and data preprocessing in the dataset. Missing values are identified and filled or removed using appropriate techniques, ensuring a complete dataset. Additionally, text preprocessing steps are applied by creating preprocess_text function. This function includes converting text to lowercase, removing special characters, and applying lemmatization to retain meaningful content. It refines so that it is suitable for machine learning models. This step is going to improve the accuracy of threat detection.

The table shown in Figure 3.4 is the outcome of text data analysis, extracting entities and relationships, and labelling. Columns like "text" hold text regarding detected entities (e.g., "malware," "attack-pattern," "TIME") and their connections, whereas columns like "entities" and "relations" store entity information. These entities' textual locations are presumably in "Start_offset" and "End_offset" columns. This type of analysis is crucial for understanding the nature of cyber threats, identifying targets and vulnerabilities, and developing mitigation strategies

	Unnamed: 0	index	text	entities	relations	Comments	id	label	start_offset	end_offset
0	0	1.0	This post is also available in: 日本語 (Japa...	[{'id': 45800, 'label': 'malware', 'start_offs...	[]	[]	45800.0	malware	288.0	300.0
1	1	2.0	The attack vector is very basic and repeats it...	[{'id': 48941, 'label': 'attack-pattern', 'sta...	[]	[]	48941.0	attack-pattern	69.0	115.0
2	2	3.0	Once executed by the user the first stage malw...	[]	[]	[]	NaN	NaN	NaN	NaN
3	3	4.0	The first known campaign was launched by Crim...	[{'id': 45806, 'label': 'TIME', 'start_offset'...	[]	[]	45806.0	TIME	55.0	68.0
4	4	5.0	The first stage downloaded the ransomware from...	[]	[]	[]	NaN	NaN	NaN	NaN

Figure 10 Sample Dataset (self-created)

The output obtained in Figure 10 is because of Sample_by_class function which samples 2500 of each unique class in the output column 'label'. This prevents biases in machine learning models by enforcing performance across all classes.

	Unnamed: 0	index	text	entities	relations	Comments	id	label	start_offset	end_offset
0	3188	3210.0	The BIOPASS RAT infection flow	[{'id': 9806, 'label': 'malware', 'start_offse...	[]	[]	9806.0	malware	4.0	15.0
1	4874	9489.0	The oRAT droppers that we found in our analysi...	[{'id': 29130, 'label': 'malware', 'start_offs...	[]	[]	29130.0	malware	4.0	8.0
2	3684	NaN	We detected a variant of the Carbanak malware ...	NaN	NaN	NaN	NaN	malware	NaN	NaN
3	5117	14096.0	At the same time, practically all code for the...	[{'id': 48176, 'label': 'malware', 'start_offs...	[]	[]	48176.0	malware	47.0	52.0
4	6468	7724.0	The .NET downloads jm1 — an .MSI installer — t...	[{'id': 23993, 'label': 'malware', 'start_offs...	[]	[]	23993.0	malware	67.0	71.0

Figure 11 Sample Dataset by Class (self-created)

The 1st DataFrame shown in Figure 11 has 476 rows with missing data in the columns index, entities, relations, Comments, id, start_offset, and end_offset .

```
Unnamed: 0      0
index         476
text            0
entities      476
relations     476
Comments      476
id            476
label           0
start_offset  476
end_offset    476
dtype: int64
```

```
Unnamed: 0    0
index         0
text          0
entities      0
relations     0
Comments      0
id            0
label         0
start_offset  0
end_offset    0
dtype: int64
```

Figure 12 Missing values and after dropping missing value dataframes (self-created)

Drop (probably data.dropna(inplace=True)) deleted missing values from all DataFrame columns. 2nd dataframe in Figure 12 has the index, entities, relations, Comments, id, start_offset, and end_offset columns are removed from the DataFrame.

The "text" and "label" columns are selected from the original DataFrame, consisting of important data such as descriptions of the malware and their classification. It allows for us to develop a machine learning model to classify text in a simple way.

	text	label
0	The BIOPASS RAT infection flow	malware
1	The oRAT droppers that we found in our analysi...	malware
3	At the same time, practically all code for the...	malware
4	The .NET downloads jm1 — an .MSI installer — t...	malware
5	Killkill: Stops the backdoor's activities inte...	malware

	text	clean_text	label
0	The BIOPASS RAT infection flow	biopass rat infection flow	malware
1	The oRAT droppers that we found in our analysi...	orat dropper found analysis mimi chat applicat...	malware
3	At the same time, practically all code for the...	time practically code slub malware created scr...	malware
4	The .NET downloads jm1 — an .MSI installer — t...	net downloads jm msi installer installs anothe...	malware
5	Killkill: Stops the backdoor's activities inte...	killkill stop backdoor activity interval chang...	malware

Figure 13 Get text and label column from all data and after preprocessing data output (self-created)

Preprocess_text function helps standardise text by lowercasing it, removing punctuation, stop words along with stemming or lemmatisation. It then sorts the DataFrame (ordering the "clean_text" column to be after "text" for clarity as shown in Figure 13.

3.1.4 Exploratory Data Analysis (EDA)

Before diving into applying Machine learning models, Exploratory data analysis is an important step to understand your dataset, find patterns and gain valuable insights (Bezerra *et al.* 2019).

Label Distribution: Code shown in Figure 3.8 creates the count plot shown on right hand side which makes it easy to visualize how many of each label are in the dataset. This assists in detecting any class imbalance which can affect the model performance.

```
# countplot for label distribution
plt.figure(figsize=(12, 4))
sns.countplot(data=data, x='label', palette='Set2')
plt.title('Distribution of Labels',size=20)
plt.xlabel('Labels',size=15)
plt.ylabel('Frequency',size=15)
plt.xticks(rotation=90)
plt.show()
```

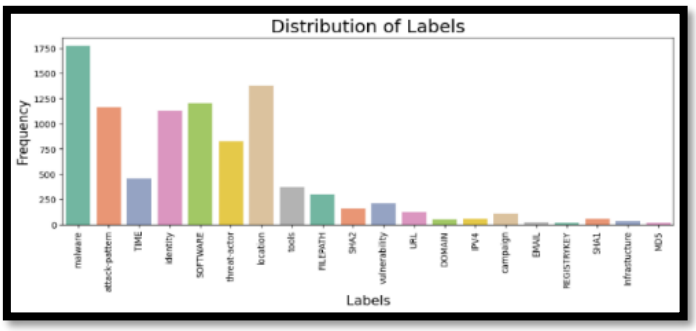


Figure 14 Label distribution (self-created)

The frequency of dataset labels or categories is shown in a bar chart. From Figure 14 malware appears to be the most common attack label. Lower frequencies are linked to labels like "threat-actor," "location," and "tools." This visualisation provides insights on the distribution of categories within the data set, which can be useful for understanding the model of the data and making decisions in later analysis or modelling phases.

Word Cloud Plot: Code shown in Figure 15 creates word cloud plot which shows the words that appear most frequently in the dataset In the text data, the word cloud in Figure 15 presents the most common words. Words like "attack," "malware," "threat," "user," and "phishing" imply a focus on cyber security threats. This is supported by the words "ransomware," "exploit," and "victim." Attack plan analysis is suggested by words like "tool," "technique," and "campaign." The word cloud displays text data's primary cyber security themes and vocabulary.

```
all_text = ' '.join(data['clean_text'])
wordcloud = WordCloud(width=1000, height=400, background_color='white').generate(all_text)
plt.figure(figsize=(12, 6))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Most Frequent Words in Text',size=20)
plt.show()
```

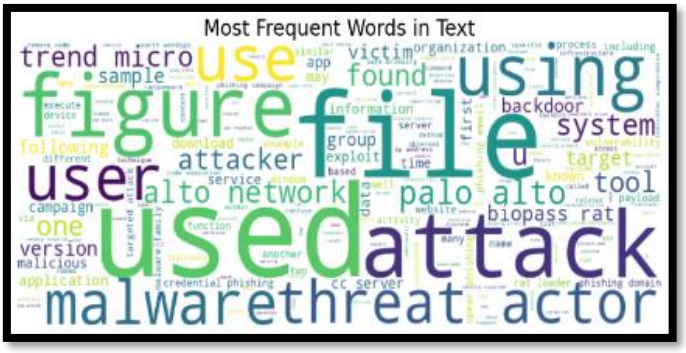


Figure 15 create Word Cloud Plot (self-created)

Violin Plot of Cleaned text Length by Label: Code shown in Figure 16 creates Violin Plot of Cleaned text Length by Label that displays distribution of cleaned text length per each label. As shown in Figure 16 for dataset categories or labels, violin charts show text length distributions. The violin shapes' widths show data point density at different text lengths. Some labels, like "tools" and "FILEPATH," have larger text descriptions than others, like "attack-pattern" and "SHA2." This information can aid in understanding the characteristics of various categories and inform feature engineering or model selection.

```
data['clean_text_length'] = data['clean_text'].apply(len)
plt.figure(figsize=(12, 6))
sns.violinplot(data=data, x='label', y='clean_text_length', palette='muted')
plt.title('Distribution of Cleaned Text Length by Label',size=20)
plt.xlabel('Labels',size=15)
plt.ylabel('Text Length',size=15)
plt.xticks(rotation=90)
plt.show()
```

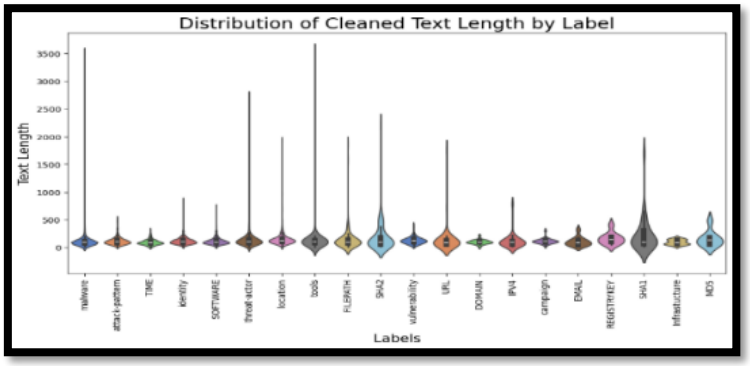


Figure 16 violin plot to calculate text length (self-created)

Box Plot of Word Count by Label: Code shown in Figure 17 creates Box Plot of Word Count by Label that is used to analyse changes in word count between different categories. The box plot in Figure 17 displays the data set's word count distribution by label. Boxes show the interquartile range (IQR) with a median line. Outliers are points that stretch 1.5 times the IQR. The plot demonstrates that labels like "tools" and "FILEPATH" tend to have longer word counts than labels like "malware" and "attack-

pattern," suggesting that descriptions of these labels tend to be longer and more detailed. This information is valuable for understanding the characteristics of separate categories and influencing feature engineering or model selection.

```
data['word_count'] = data['clean_text'].apply(lambda x: len(x.split()))
plt.figure(figsize=(12, 6))
sns.boxplot(data=data, x='label', y='word_count', palette='coolwarm')
plt.title('Word Count Distribution by Label',size=20)
plt.xlabel('Labels',size=15)
plt.ylabel('Word Count',size=15)
plt.xticks(rotation=90)
plt.show()
```

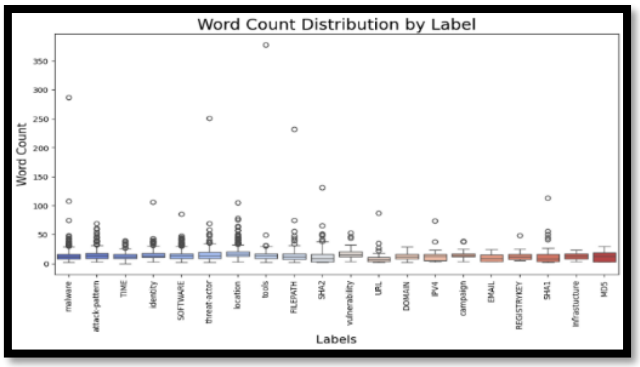


Figure 17 Word count using box plot (self-created)

Top 5 Labels Pie Chart: Code shown in Figure 18 creates pie chart that shows top 5 labels in the dataset. The data set's pie graphic in Figure 8 presents the top five labels' distribution. Most data is marked "malware" (26.6%), followed by "location" (20.8%), "SOFTWARE" (18.1%), "attack-pattern" (17.5%), and "identity" (17.0% This visualisation illustrates the top categories within the data set, which is valuable for understanding the data's composition and making decisions in later analysis or modelling phases.

```
top_labels = data['label'].value_counts().nlargest(5)
plt.figure(figsize=(8, 4))
top_labels.plot.pie(autopct='%1.1f%%',startangle=90,colors=sns.color_palette("pastel"))
plt.title('Top 5 Label with Percentage')
plt.axis("off")
plt.show()
```

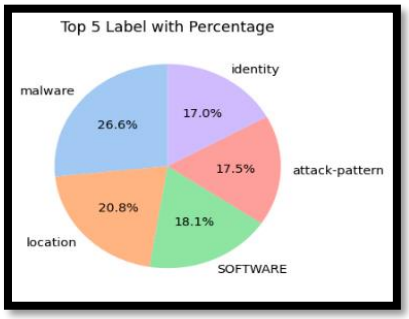


Figure 18 Top 5 labels by their frequency using pie chart (self-created)

Top Labels Most Used Words Bar Plot (Top Words): Code shown in Figure 19 creates Words Bar Plot that displays most used words for most common labels.

```
top_labels = data['label'].value_counts().nlargest(4).index
fig, axes = plt.subplots(2, 2, figsize=(12, 6))
axes = axes.flatten()
for i, label in enumerate(top_labels):
    label_text = ' '.join(data[data['label'] == label]['clean_text'])
    common_words = Counter(label_text.split()).most_common(10)
    words, counts = zip(*common_words)

    sns.barplot(x=list(counts), y=list(words), palette='viridis', ax=axes[i])
    axes[i].set_title(f'Top Words for Label: {label}')
    axes[i].set_xlabel('Frequency')
    axes[i].set_ylabel('Words')
plt.tight_layout()
plt.show()
```

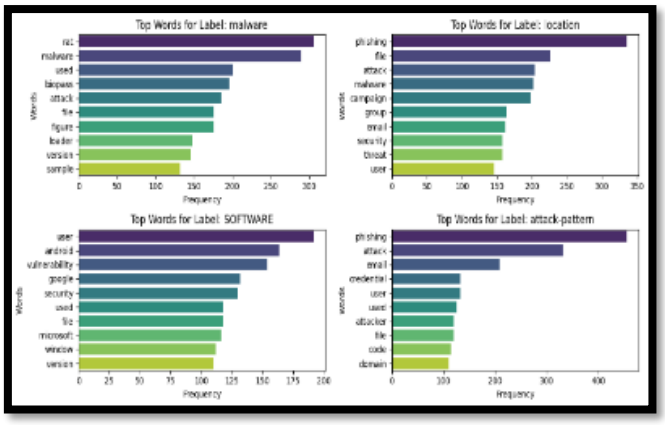


Figure 19 Top 4 labels by frequency (self-created)

The bar charts illustrate the top four dataset labels' most common words: "malware," "location," "SOFTWARE," and "attack-pattern." Insights into the nature and data of each label are provided by this analysis, which can be used to comprehend the data's themes and perhaps guide the construction of a new label or model. The "malware" label includes words like "rat," "biopass," "attack," and "file," suggesting a focus on malware-related keywords and behaviours. The "location" label is linked to words like "attack," "phishing," and "file," indicating a focus on cyber dangers and targeted attacks. The "SOFTWARE" label includes "user," "android," "vulnerability," and "security," suggesting a software problem and security focus. However, the "attack-pattern" label uses words like "phishing," "attack," "email," and "credential," indicating a focus on credential theft. The term "phishing" appears in many charts, highlighting its significance across labels. Labels like "attack," "file," and "user" are common, indicating cyber security relevance. The bar charts provide valuable insights into the data and themes associated with each label, which can be used to improve data understanding and feature engineering or model selection decisions.

Bar Plot of Average Text Length by Label: Using sns.barplot code shown in Figure 20 calculate the average length of text by label . This helps understand if text lengths vary greatly between labels.

A bar chart in Figure 20 shows typical label text descriptions (length). Labels like "SHA1", "SHA2", and "SOFTWARE" tend to have longer text, suggesting more detailed descriptions. The average text length of labels like "EMAIL", "IPV4", and "DOMAIN" is shorter, indicating more concise descriptions. This information can aid in understanding the characteristics of various categories and inform feature engineering or model selection.

```
avg_length = data.groupby('label')['clean_text_length'].mean().reset_index()
plt.figure(figsize=(12, 6))
sns.barplot(data=avg_length, x='label', y='clean_text_length', palette='Set3')
plt.title('Average Text Length by Label',size=20)
plt.xlabel('Labels')
plt.ylabel('Average Text Length')
plt.xticks(rotation=90)
plt.show()
```

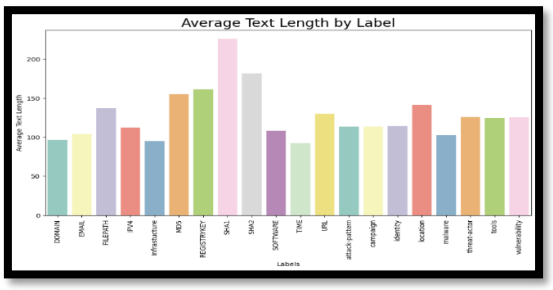


Figure 20 Average text length by label (self-created)

3.1.5 Feature engineering

In simple words the meaning of feature engineering is to construct different variables/features based on the given data. This can be using different mathematical operations or methods (Popov, 2023). In this application two variables in data set are in text form but to make this data set understandable for machine vectorization is important. Label variables have different categories which are converted to numeric form using LabelEncoder method available in python. Also, text variables are converted to vectors using TFIDF method. This method considers the importance of words in a document relative to the entire data set with a maximum of 5000 features to reduce the dimensionality.

```
# vectorize clean_text using TF-IDF
vectorizer = TfidfVectorizer(max_features=5000)
X = vectorizer.fit_transform(data['clean_text']).toarray()

pickle_filename = "vectorizer.pkl"
with open(pickle_filename, 'wb') as file:
    pickle.dump(vectorizer, file)

print(f"Vectorizer model saved as {pickle_filename}.")
# encode labels to numeric values
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(data['label1'])
pickle_filename = "label_encoder.pkl"
with open(pickle_filename, 'wb') as file:
    pickle.dump(vectorizer, file)

print(f"Label encoder model saved as {pickle_filename}.")
```

Figure 21 Feature engineering using Tfidf vectorizer and label encoder (self-created)

The code in Figure 21 performs two main preprocessing tasks, text data vectorization using TF-IDF (Term Frequency-Inverse Document Frequency) and categorical label encoding to numerical values. First, a TfidfVectorizer is initialized with a maximum of 5000 features so that only the most important words are considered. Fit Transform converts the clean_text column into a numerical array, where each row represents a document, and each column depends on the importance of the word measured by the TF-IDF scores. To allow for reuse without having to retrain the model, the trained vectorizer is saved as a pickle file (vectorizer.pkl). Next, LabelEncoder, which is used to convert categorical labels (attack types) to numerical values, allowing training for the model with these labels. The transformed labels (y) are stored as a numerical array, and the trained label encoder is saved in an easily accessible format as label_encoder.pkl.

3.1.6 Train-Test Split

Separating prepared data into training and testing sets is crucial. On the larger training set, machine learning models are trained. The smaller testing set evaluates model performance on unseen data. It helps to evaluate how well models generalise to new examples and prevent overfitting to training data.

```
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```

Figure 22 Data split (self-created)

Code shown in Figure 22 is using Scikit-learn's train_test_split function splits training and testing sets, allowing we to select the ratio.Data divided into training set and testing set in a ratio of 8:2 that is 80% data is for training model and 20% data kept for testing the results of the model (Rácz, Bajusz & Héberger 2021).

3.1.7 Model training and model evaluation

Once data is divided into training and testing sets, the next phase is of fitting models to predict the type of attack (Oliynyk, Mayer & Rauber 2023). Four models, Random Forest, SVM, Logistic Regression and Artificial Neural Network (ANN) have been used for training Model selection and deployment. Based on evaluation metrics such as accuracy, precision, recall, and F1-score best-performing model is selected. In order to identify the model with the best accuracy on the testing data. When model evaluation is complete, the model with the best performance (typically the highest accuracy on testing data) is chosen for deployment. The Random Forest model does well in this case. The pickle dump

function serialises and stores the trained model in a file to save it for future use (Pruneski et al., 2022). After training, load the model to predict cyber threats on new text data.

3.1.8 Tools and libraries

Here's a table of tools and libraries used in AI threat detection.

Library/Tool	Purpose
pandas	Data manipulation and preprocessing
numpy	Numerical operations and array handling
matplotlib	Data visualization (e.g., plotting distributions)
seaborn	Statistical data visualization (e.g., count plots, box plots)
sklearn (scikit-learn)	Machine learning model training, evaluation, and preprocessing (e.g., TfidfVectorizer, LabelEncoder, train_test_split, classification_report, accuracy_score,building ML models)
wordcloud	Generating word cloud visualizations for text data
collections (Counter)	Counting word occurrences for most frequent word analysis
pickle	Saving trained models (vectorizer and label encoder) for reuse
TensorFlow/Keras	Building and evaluating the Artificial Neural Network (ANN) model

3.2 Web application

The methodology focuses on developing a Flask-based web application to predict web vulnerabilities. The web application created provide an effective communication medium for users with real time threat detection. The application will implement AI for message analysis, secure user authentication, and an easy-to-use interface. The application will be implemented in a cloud infrastructure keeping scalability and reliability in mind. There are 3 key phases to develop AI-powered Threat Detection Web Application:

1. **Frontend Development:** It includes using HTML, CSS, and JavaScript to create a responsive and user-friendly interface.
2. **Backend Development:** This phase includes the implementation of a Flask-based API for handling user input and integrating the ML model.
3. **Database Management**: Used for database management, handling the SQL database to store and manage user and employee data.
4. **Model Integration & Preprocessing:** It includes loading a pre-trained ML model that is Random Forest for processing the input and predicting web vulnerabilities (Wang et al., 2022).

3.2.1 System Architecture

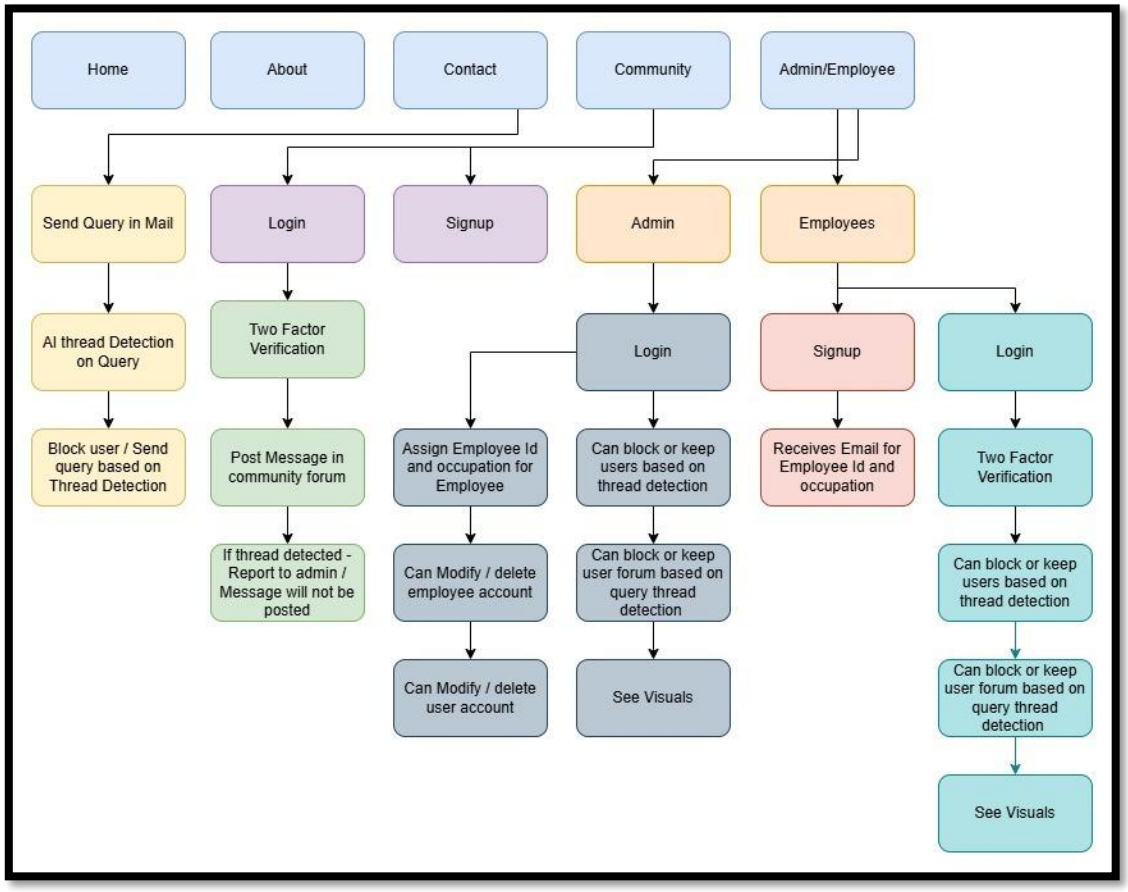


Figure 23 System architecture *(self-created)*

This block diagram shown in Figure 23 depicts web application's architecture. Workflow is structured and the key functionalities presented through various modules. The Home page acts as a launch section from where users can explore sections like About, Contact, Community, and Admin/Employee functionalities. The user journey begins with options such as an email query, login, or sign-up. Security is considered by secured two-factor authentication.

The Contact section allows anyone to send a query, which is passed through the AI threat detection system that will either consider the query as a threat and stop the user from further communication or allow it through if considered safe. The Community module permits messages to be posted by users, which go through threat detection system using AI to detect any harmful content. If a message is found to be threat, a notification is sent to the user. Also, the message is sent to admin for further actions. In the Admin/Employee module, other functionalities become present. Employees that have signed up will get an Employee ID and details about their profession from an email. They will then log in using two factor authentication for moderating actions on user interactivity and block or retain users according to detected threats, along with visual analytics. Admins have full control over user account management and assignment of employee roles. This architecture ensures smoothly integrated roles of users, aimed toward security while retaining a user-friendly approach along with scalability. The power of AI threat detection, two-factor authentication, and administrative oversight create a safe, well-managed platform.

3.2.2 Tools and Techniques

Frontend Development: Front end Developed with both frontend and backend integration, this AI-powered threat detection tool ensures smooth user interface and processing of data. Frontend is built using HTML, CSS and JavaScript.HTML provides the structure of web pages, CSS polishes the design

and makes it modern and responsive. JavaScript adds elements of interactivity and works with the Flask backend for real-time processing and response (Goh et al., 2022). Without reloading the page predictions are fetched from the AI model.

Backend Development: Backend is based on Flask. Flask is a lightweight framework ideal for building modular web applications with minimum overhead. Its built-in template engine, Jinja2, allows for generation of HTML dynamically, making it fit for building interactive user interfaces. By combining the flexibility of Flask with the vast ecosystem of Python, developers can use libraries such as NumPy, Pandas, and Scikit-learn to efficiently build enterprise-grade machine learning pipelines. Flask allows the transfer of data between the front-end and AI model seamlessly (Singh & Paul, 2020). SQLAlchemy ensures secure, efficient database interactions for user and role management.

Machine Learning: The machine learning technique uses the TF-IDF (Term Frequency-Inverse Document Frequency) Vectorizer, which takes the text data and generates the numerical features while keeping the important words (Liu et al., 2018). It uses a Random Forest Model to achieve its high accuracy and also reduce the dimensionality, making it suitable for complex text classification tasks (Ao et al., 2019).For converting numerical outputs into readable categories Label Encoder is used.

3.2.3 Libraries used

Library	Purpose
Flask	A web framework used to create the web application.
render_template	Renders HTML templates for web pages.
request	Handles incoming HTTP requests (e.g., form data).
redirect	Redirects users to different routes.
url_for	Generates URLs for different routes dynamically.
session	Manages user sessions for authentication.
flash	Displays temporary messages (e.g., success/error messages).
SQLAlchemy	A database toolkit for managing the SQLite/PostgreSQL database in Flask.
generate_password_hash	Hashes passwords for secure storage.
check_password_hash	Verifies hashed passwords during login.
joblib	Loads pre-trained machine learning models (e.g., for AI-based threat detection).
nltk	A natural language processing (NLP) library used for text analysis.
re	Provides regular expression operations for text cleaning.
random	Generates random numbers (e.g., OTP generation).
flask_mail	Sends emails (e.g., OTP verification and notifications).
Message	Creates email messages for sending via Flask-Mail.
os	Provides functions to interact with the operating system (e.g., file handling).
secure_filename	Ensures safe handling of uploaded files by sanitizing filenames.
datetime	Manages date and time-related operations.

stopwords	Provides a list of common words (e.g., "the", "is") to remove during text preprocessing.
WordNetLemmatizer	Reduces words to their base form (e.g., "running" → "run") for better NLP processing.

3.3 AWS deployment with Cyber Security

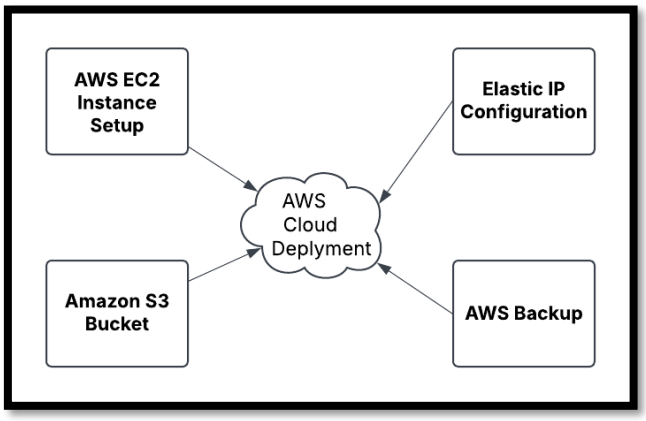


Figure 24 AWS Deployment (self-created)

The above figure 24 shows that steps included in the AWS deployment and those steps are explained in the follows:

Virtual Private Cloud (VPC) and Elastic IP: Cloud resources are secure in the VPC. It keeps instances and services in a private network for security and scalability. The screenshots show how an EC2 instance can be assigned a public IPv4 address called elastic IP to make it internet accessible. This functionality helps maintain application connections even when the instance changes (Amazon Web Services, 2025a).

Amazon EC2 instances: EC2 hosts the AI-powered threat detection application. Us-east-1c availability zone instance vulnerability_app is shown in screenshots. The t2.small instance type is cost-effective and ideal for lightweight applications. The EC2 instance passes health checks, ensuring reliability. For internet connectivity, it is also assigned an Elastic IP. Scale the instance type using AWS Compute Optimiser suggestions if the application needs more performance (Amazon Web Services, 2025b).

Amazon S3 (Simple Storage Service): Amazon S3 is secure and reliable backup storage. In the screenshots, vulnerabilityapp-bucket2 is hosted in us-east-1. This bucket enables easy storage and retrieval of application files. The screenshots show a drag-and-drop interface for file uploads. S3 storage is appropriate for huge datasets and application logs because to its scalability and durability (Amazon Web Services, 2025c).

AWS Backup Service: AWS Backup automates backups for disaster recovery. The screenshots show how to create a DisasterRecoveryPlan with scheduled essential data backups. The plan can store data for a set time to restore lost data. The flexibility to assign resources and establish retention policies protects critical data, improving application stability (Amazon Web Services, 2025d).

AWS Deployment and Security: Secure setups, such as two-factor authentication and IAM role management, support the deployment of the application. The web application, accessible via the Elastic IP, uses AWS infrastructure for high availability and security. The setup meets modern security standards by isolating resources in a VPC and employing secured Elastic IPs to protect the application and users (Amazon Web Services, 2025e).

4 Design of Web Application

4.1 Introduction

The web application employs (Figure 25) AI to detect and mitigate cyber threats in real time while fostering secure and user-friendly communication. The application uses advanced AI and cyber security to protect users and data. Its architecture allows many user roles, including administrators, employees, registered, and unregistered users, for easy operations and secure interactions.

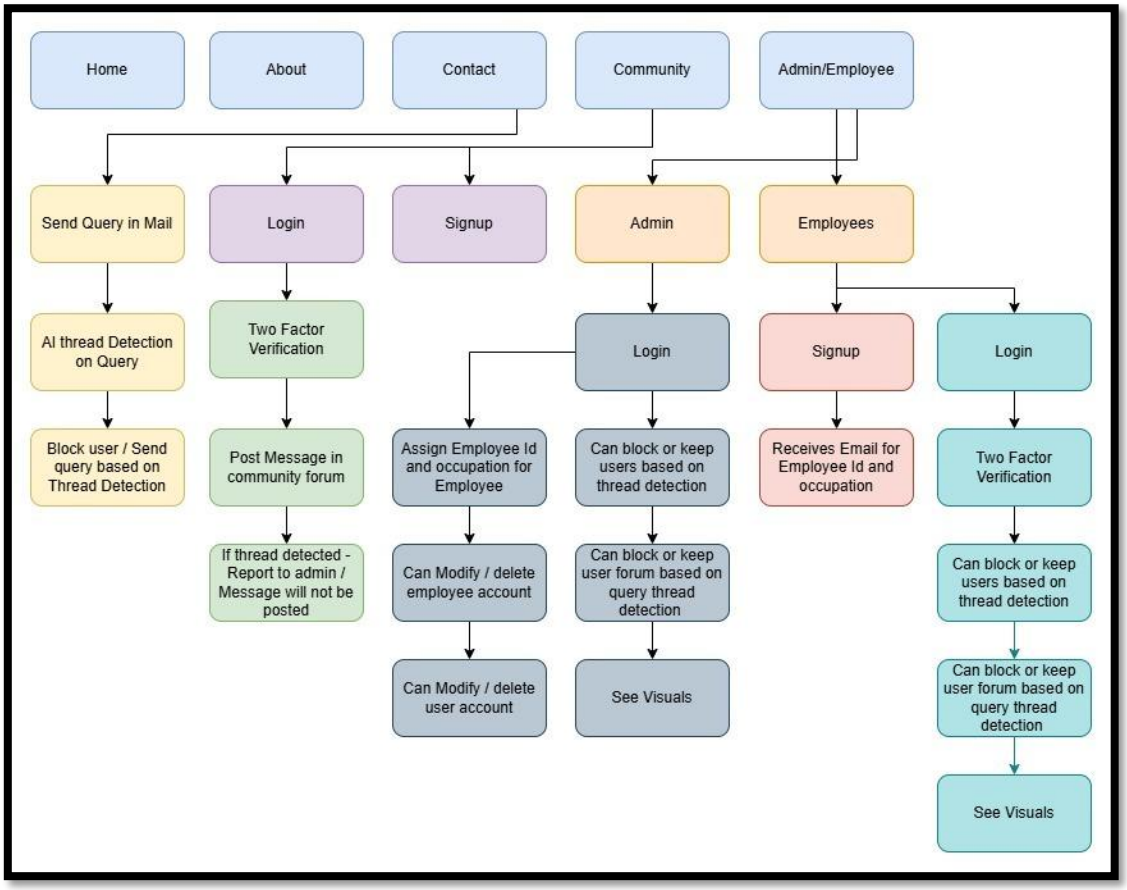


Figure 25 Web application design (self-created)

About, Contact, Community, and Admin/Employee are accessible from the homepage. To identify potential threats and limit users, an AI-based threat detection system examines Contact section queries. Monitoring threats and communication integrity ensures administrators. Real-time threat analysis is possible on the Community forum for registered users. Administrators receive alerts about flagged messages, which prevent posting and enable proactive reactions.

Administrators and staff play key roles in user interactions. To secure account access, employees are assigned unique IDs and occupations via two-factor authentication at signup. They monitor user activity, manage accounts, and respond to threats. Admins can alter accounts, assign roles, and view user activity.

The web application prioritises security with two-factor authentication, AI-based message analysis, and a good account management system. This flexible and secure communication system is scalable and user-friendly. AI for threat identification and real-time admin and employee decision-making ensures a reliable foundation for current cyber security.

4.2 Database Design

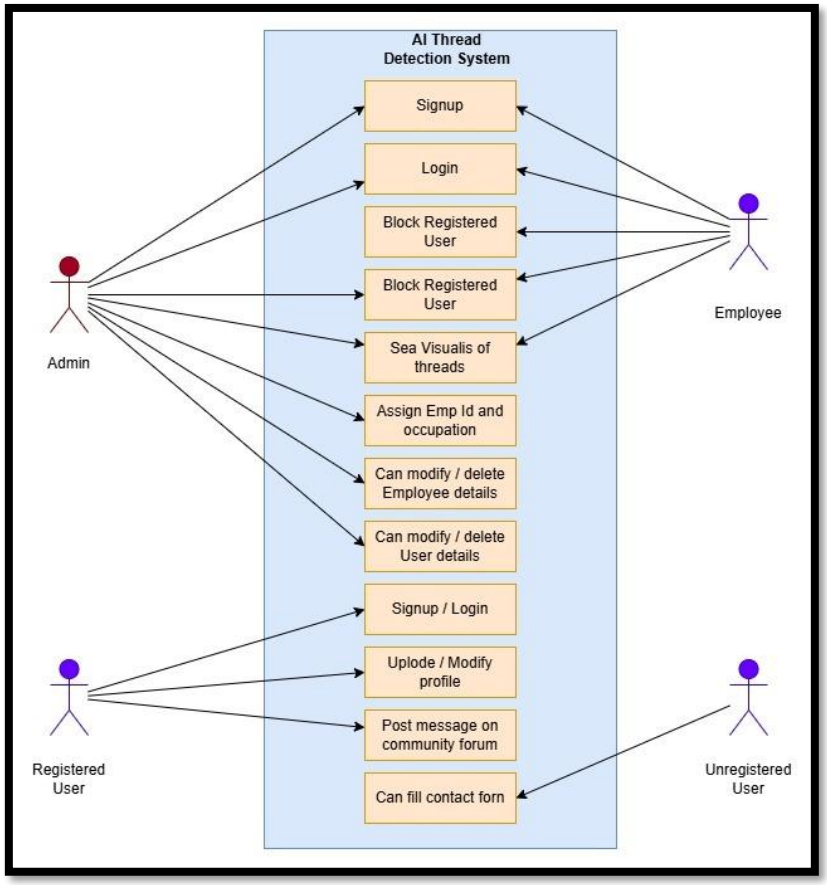


Figure 26 Use Case Diagram (self-created)

The above use case diagram (Figure 26) explained as follows:

- **Admin Role:** The admin can manage user and employee data, including blocking users, modifying/deleting details, and assigning employee roles (ID and occupation). They also oversee system threads.
- **Employee Role:** Employees can log in, view thread visuals, and perform tasks assigned by the admin related to thread monitoring or data updates.
- **Registered User:** Registered users can sign up, log in, update their profiles, post on community forums, and engage with the system interactively.
- **Unregistered User:** Unregistered users are limited to filling out contact forms to communicate or request access.
- **System Design:** The "AI Threat Detection System" ensures clear role-based access and functionality, enhancing usability and security.

4.3 Implementation

4.3.1 System Setup

It includes installing Flask and required libraries using anaconda prompt .The files are organized in a modular design focusing on development and deployment. Static folders contain frontend assets like CSS (style. css) and JavaScript (script. js). The templates folder contains all HTML files for each web page (home, about, contact, admin, etc).

Figure 27 project *structure in visual studio code (self-created)*

Machine Learning artifacts such as trained model (best_model. pkl), vectorizer (vectorizer. pkl) and the label encoder (label_encoder. pkl) are placed inside model folder as shown in Figure 27. Flask application's starting point is app.py that handles routing, user inputs, model predictions, and API integration for providing threat detection results dynamically.

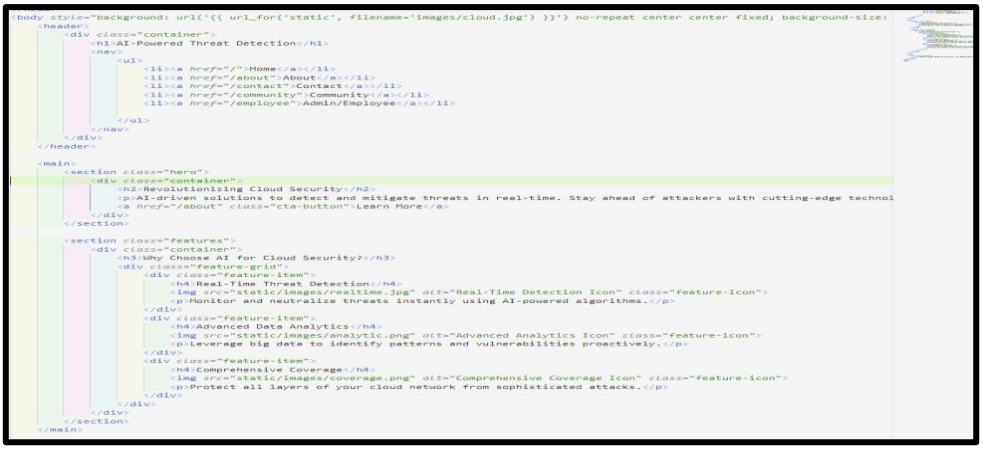
4.3.2 Frontend and Backend Implementation

For the front end, the AI-Powered Threat Detection system combines HTML, CSS, and JavaScript to create a clean, responsive, and appealing interface. HTML organizes content on all pages; CSS gives modern touch-and-feel and interactivity. Three different webpages are created:

- Home page
- About page
- Contact page
- Community page
- Admin/Employee page

4.3.3 Home page

It represents the main landing page for the AI-Powered Threat Detection project. It uses Flask's url_for function to dynamically load static assets (CSS, and images) and provides a modern, responsive layout with the following sections.For backend implementation flask framework is used. It provides navigation between different webpages and throw light on importance of AI in cybersecurity. In Figure 28 front end, backend code is given with the home page.



```
@app.route('/')
def home():
    return render_template('home.html')
```

Figure 28 home page (self-created)

4.3.4 About page

This page represents the About Us page for the AI-powered threat detection system. It provides information about the mission and team members, using a clean, organized layout. It has same header and footer section as home page.

```
<main>
    <section>
        <h2>Our Mission</h2>
        <p>To provide state-of-the-art AI-driven threat detection solutions for securing cloud networks against evolving cyber th
    </section>

    <section>
        <h2>Our Team</h2>
        <div class="team-member">
            <img src="{{ url_for('static', filename='images/member1.png') }}" alt="Team Member 1">
            <h3>Akshitha </h3>
            <p><strong>CEO & Founder:</strong> Visionary leader with a passion for AI-driven security solutions.</p>
        </div>
        <div class="team-member">
            <img src="{{ url_for('static', filename='images/member2.jpeg') }}" alt="Team Member 2">
            <h3>Jay</h3>
            <p><strong>CTO:</strong> Technical innovator driving advancements in AI and machine learning.</p>
        </div>
        <div class="team-member">
            <img src="{{ url_for('static', filename='images/member3.png') }}" alt="Team Member 3">
            <h3>Sonam</h3>
            <p><strong>Cybersecurity Specialist:</strong> Expert in cloud security architecture and threat mitigation.</p>
        </div>
        <div class="team-member">
            <img src="{{ url_for('static', filename='images/member3.png') }}" alt="Team Member 3">
            <h3>Harsh</h3>
            <p><strong>Cybersecurity Specialist:</strong> Expert in cloud security architecture and threat mitigation.</p>
        </div>
    </section>
</main>
```

```
@app.route('/contact')
def contact():
    return render_template('contact.html')
```

Figure 29 About page with mission and team member's detail (self-created)

Code in figure 29 displays team members with images and roles and is structured for easy readability.

4.3.5 Contact page

Contact Us page for the AI-Powered Threat Detection project. It includes a user-friendly form to send a query that will go through AI powered system, essential contact details, and an embedded map for location reference. Backend code in Figure 30 clearly shows how data submitted in form is checked for security purpose and classified as threat or non-threat message. If the message is non-threat, it will be sent on admin email id. If it is detected as a threat it is reported to admin for further action.

```html
<section>
    <div class="contact-info">
        <p><i class="fas fa-phone"></i> Phone: +61 43 34 24 662</p>
        <p><i class="fas fa-envelope"></i> Email: CQU</p>
        <p><i class="fas fa-map-marker-alt"></i> Ann Street, Brisbane, Queensland</p>
    </div>
</section>
{% if block_message %}
<div style="color: red; font-weight: bold; margin-bottom: 10px;">
    {{ block_message }}
</div>
endif %}

<form action="/submit_form" method="post">
    <label for="name">Your Name:</label><br>
    <input type="text" id="name" name="name" placeholder="Enter your name" required><br><br>

    <label for="email">Your Email:</label><br>
    <input type="email" id="email" name="email" placeholder="Enter your email" required><br><br>

    <label for="message">Your Message:</label><br>
    <textarea id="message" name="message" rows="5" placeholder="Write your message here..." required></textarea><br><br>

    <button type="submit">Send Message</button>
</form>

<section class="map">
    <h3>Find Us Here</h3>
    <iframe
        src="https://www.google.com/maps/embed?pb=!1m18!1m12!1m3!1d3540.057133333613!2d153.02168267477987!3d-27.4674806166103
        width="100%"
        height="400"
        style="border:0;"
        allowfullscreen=""
        loading="lazy">
    </iframe>
</section>
```

```python
@app.route('/submit_form', methods=['POST'])
def submit_form():
    try:
        if request.method == 'POST':
            # Get form data
            name = request.form['name']
            email = request.form['email']
            message_content = request.form['message']

            # Check if the user is already blocked
            blocked_user = MalwareDB.query.filter_by(email=email, is_blocked=True).first()
            if blocked_user:
                message = "You are blocked from submitting messages."
                return render_template('contact.html', block_message=message)
            # Preprocess and predict
            clean_message = preprocess_text(message_content)
            vectorized_message = vectorizer.transform([clean_message]).toarray()
            prediction = model.predict(vectorized_message)[0]
            threat_type = label_encoder.inverse_transform([prediction])[0]
            print(f"Threat Type Detected: {threat_type}")
            # List of threat keywords
            threat_keywords = [
                'malware', 'attack-pattern', 'time', 'identity', 'software',
                'threat-actor', 'location', 'tools', 'filepath',
                'vulnerability', 'url', 'domain', 'ipv4', 'campaign',
                'email', 'registrykey', 'sha1', 'infrastructure', 'md5', 'hash'
            ]
            # Check if the message is a threat
            is_threat = any(keyword in threat_type.lower() for keyword in threat_keywords)

            if is_threat:
                # Log threat to MalwareDB
                threat_record = MalwareDB(
                    username=name,
                    email=email,
                    mobile="N/A",
                    message_content=message_content,
                    threat_type=threat_type
                )
                db.session.add(threat_record)
                db.session.commit()

                # **Automatic Blocking (Optional)**
                user_threats = MalwareDB.query.filter_by(email=email).count()
                if user_threats >= 3:  # If user has sent 3+ threats, block them
                    user = MalwareDB.query.filter_by(email=email).first()
                    user.is_blocked = True
                    db.session.commit()
                    message = "You have been blocked due to multiple flagged messages."
                    return render_template('contact.html', block_message=message)

                message = f"Your message was flagged as a threat ({threat_type}) and reported to the admin."
                return render_template('contact.html', block_message=message)

            else:
                # Send an email for non-threatening messages
                try:
                    msg = Message(
                        subject="New Message Received - Non-Threat",
                        sender=app.config['MAIL_USERNAME'],
                        recipients=['jayy3685@gmail.com']
                    )
                    msg.body = f"""
                    Hello,

                    A new non-threatening message was received from {name} ({email}):

                    Message Content:
                    {message_content}

                    Thank you.

                    Regards,
                    AI-Powered Threat Detection System
                    """
                    mail.send(msg)
                    message = "Your message has been successfully sent. Thank you for contacting us."
                except Exception as e:
                    logging.error(f"Error sending email: {e}")
                    message = "An error occurred while sending the notification email. Please try again."

                return render_template('contact.html', block_message=message)

    except Exception as e:
        logging.error(f"Error in /submit_form: {e}")
        message = "An error occurred while processing your request. Please try again later."
        return render_template('contact.html', block_message=message)
```

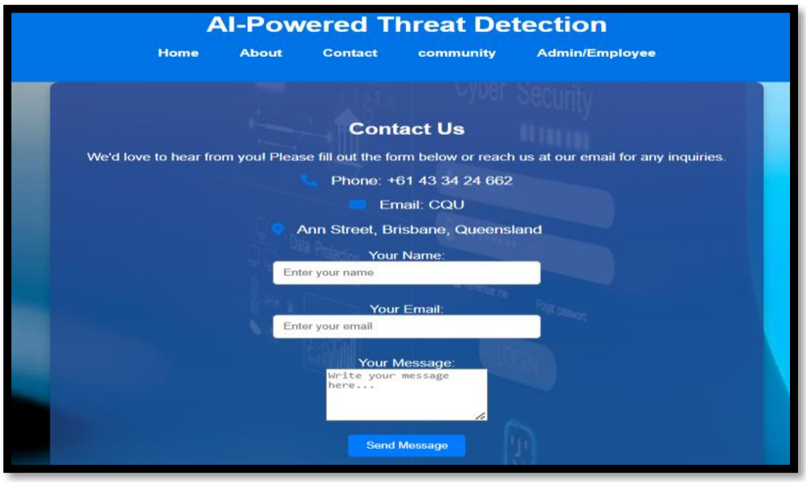


Figure 30 AI powered query form in contact page (self-created)

4.3.6 Community page

community page has both login and signup buttons for the user. The login button is for existing users to access their accounts, and Sign Up is for new users, specifically employees, to create an account. This section fosters collaboration by allowing users to join the community, share ideas, ask questions, and contribute to discussions. It creates a seamless entry point for employees to engage with the community, enhancing interaction and participation (Figure 31).

```
<section class="community-section" style="display: flex; justify-content: center; align-items: center; text-align: center; min-height: 60vh; padding: 20px; background: linear-gradie
    <div style="width: 100%; max-width: 600px;">
        <h2 style="font-size: 2rem; margin-bottom: 20px; color: #fff;">Welcome to the Community</h2>
        <p style="font-size: 1.1rem; margin-bottom: 30px; color: #fff;">Join us and become part of our growing community of like-minded individuals. Share ideas, ask questions, and

        <div style="display: flex; justify-content: center; gap: 20px;">
            <a href="/login" style="display: inline-block; padding: 12px 24px; font-size: 1rem; font-weight: bold; text-decoration: none; color: #fff; background: rgba(160, 28, 2
            <a href="/signup" style="display: inline-block; padding: 12px 24px; font-size: 1rem; font-weight: bold; text-decoration: none; color: #fff; background: rgba(0, 100, 0,
        </div>
    </div>
</section>
```

```
@app.route('/community')
def community():
    return render_template('community.html')
```

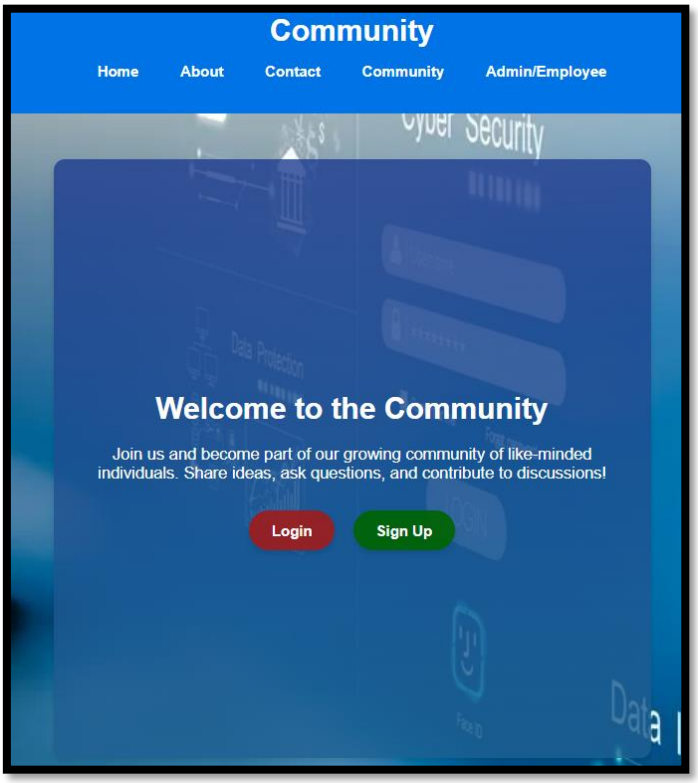


Figure 31 Community page with login and signup (self-created)

If user clicks on signup button user is redirected to signup page using the code shown in Figure 32 where a signup form is there with all important information regarding user like username, E-mail, Mobile number, password, confirm password, age and gender. These fields include email validation and strong password validation.

```
<main>
    <h2>Sign Up</h2>
    <p>Join us and gain access to state-of-the-art cloud security solutions. Please fill in your details below to create an account.</p>

    {% if error %}
        <p class="error-message">{{ error }}</p>
    {% endif %}

    <form action="/signup" method="post">
        <label>Username:</label>
        <input type="text" name="username" required>

        <label>Email:</label>
        <input type="email" name="email" required>

        <label>Mobile:</label>
        <input type="text" name="mobile" required>

        <label>Password:</label>
        <input type="password" name="password" required>

        <label>Confirm Password:</label>
        <input type="password" name="confirm_password" required>

        <label>Age:</label>
        <input type="number" name="age" required>

        <label>Gender:</label>
        <select name="gender">
            <option value="male">Male</option>
            <option value="female">Female</option>
        </select>

        <button type="submit">Sign Up</button>
    </form>

    <p>Already have an account? <a href="/login">Login here</a>.</p>
</main>
```

```
@app.route('/signup', methods=['GET', 'POST'])
def signup():
    if request.method == 'POST':
        username = request.form['username']
        email = request.form['email']
        mobile = request.form['mobile']
        password = request.form['password']
        confirm_password = request.form['confirm_password']
        age = request.form['age']
        gender = request.form['gender']

        # Validate if passwords match
        if password != confirm_password:
            return render_template('signup.html', error="Passwords do not match.")

        # Validate password strength
        if not re.match(r'^(?=.*[A-Za-z])(?=.*\d)(?=.*[@$!%*#?&])[A-Za-z\d@$!%*#?&]{8,}$', password):
            return render_template('signup.html', error="Password must be at least 8 characters long, include a letter, a number, and a special character.")

        # Check if username or email already exists
        if User.query.filter((User.email == email) | (User.username == username)).first():
            return render_template('signup.html', error="Username or email already exists.")

        # Check if user is blocked
        if User.query.filter_by(email=email, is_blocked=True).first():
            return render_template('signup.html', error="You are blocked and cannot sign up.")


        # Generate a unique user ID and hash the password
        user_id = generate_user_id()
        hashed_password = generate_password_hash(password)

        # Add the new user to the database
        try:
            new_user = User(
                username=username,
                email=email,
                mobile=mobile,
                password=hashed_password,
                age=age,
                gender=gender,
                user_id=user_id
            )
            db.session.add(new_user)
            db.session.commit()
            return render_template('signup_complete.html', message=f"Signup Successful! Your User ID: {user_id}")
        except Exception as e:
            return render_template('signup.html', error=f"An error occurred: {e}")

    return render_template('signup.html')
```

Figure 32 Sign up page for user(self-created)

After successful signup with the input fields, the system will generate user id and notify the user to use while login into the account using below code in figure 33.

```
<main>
    <h2>{{ message }}</h2>
    <p>Thank you for signing up! Your account has been successfully created.</p>
    <p><strong>Your Generated ID:</strong> {{ user_id }}</p>
    <a href="/login">Go to Login</a>
</main>
```

Figure 33 Sign up success page for user with Generation of User ID (self-created)

After successful signup user can click on login button to login and user is redirected to login page (Figure 34). To log in, users need to enter their registered email ID, their unique Employee ID, and their password. Additionally, the login page includes a Forgot Password feature.

```
<main>
    <div class="login-container">
        <h2>Login</h2>
        {% if error %}
            <p style="color: red;">{{ error }}</p>
        {% endif %}
        <form action="/login" method="POST">
            <label for="email">Email:</label>
            <input type="email" id="email" name="email" placeholder="Enter your email" required>

            <label for="user_id">User ID:</label>
            <input type="text" id="user_id" name="user_id" placeholder="Enter your User ID" required>

            <label for="password">Password:</label>
            <input type="password" id="password" name="password" placeholder="Enter your password" required>

            <button type="submit">Login</button>
        </form>
        <div class="links">
            <a href="/forgot_password">Forgot Password?</a>
        </div>
    </div>
</main>
```

```
# Assuming otp_storage and mail are already set up
otp_storage = {}
OTP_VALIDITY_PERIOD = timedelta(minutes=5)  # OTP validity duration

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        user_id = request.form['user_id']
        email = request.form['email']
        password = request.form['password']

        # Validate user login credentials
        user = User.query.filter_by(user_id=user_id, email=email).first()
        if not user:
            return render_template('login.html', error="Invalid credentials.")

        # Check if the user is blocked
        if user.is_blocked:
            return render_template('login.html', error="You are blocked and cannot log in.")

        # Verify the password
        if not check_password_hash(user.password, password):
            return render_template('login.html', error="Invalid password.")

        # Generate OTP for 2FA
        otp = str(random.randint(100000, 999999))
        otp_expiry_time = datetime.now() + OTP_VALIDITY_PERIOD
        otp_storage[email] = {'otp': otp, 'expiry_time': otp_expiry_time}

        # Send OTP via email
        try:
            msg = Message('Your Login OTP', sender='jayy3685@gmail.com', recipients=[email])
            msg.body = f'Your OTP is: {otp}. It is valid for 5 minutes.'
            mail.send(msg)
        except Exception as e:
            return render_template('login.html', error=f"Failed to send OTP: {e}")

        # Redirect to OTP verification page
        flash("A one-time password (OTP) has been sent to your email. Please enter it to proceed.", "info")
        return redirect(url_for('verify_otp', email=email))

    return render_template('login.html')
```

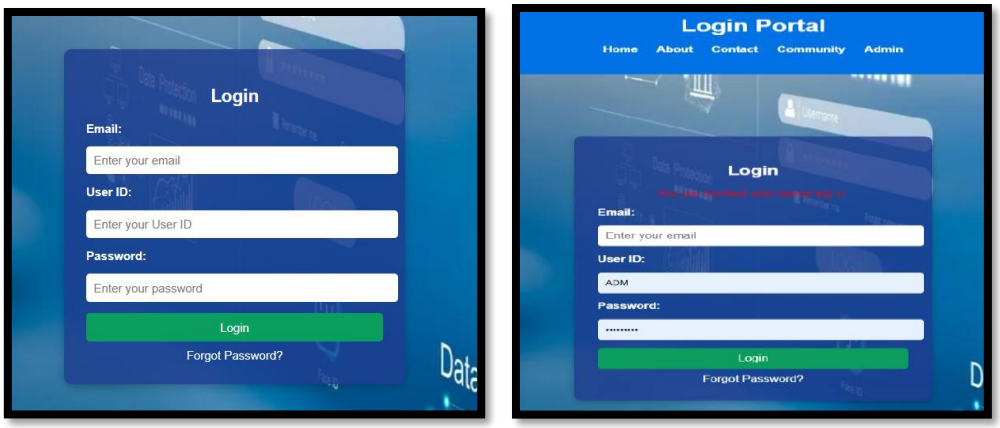


Figure 34 Login page (self-created)

If blocked user (blocked by admin for posting a threat message on community) wants to login into the community, then it will notify that, “your account was blocked!”. On the login page if a user forgets their password, they can click on the Forgot Password option on the login page. This will redirect them to the first step of the reset process, where they must enter their registered email ID and click on Submit.

An OTP (One-Time Password) will then be sent to the registered email address for verification. On the next page, users must input the received OTP and create a new password, ensuring they also confirm the password for accuracy. Once completed, they can click on Reset Password to securely update their login credentials. Additionally, if the OTP is not received or has expired, users can click on the Resend OTP button to generate a new OTP, ensuring a smooth and secure password recovery process. This feature using code in Figure 35 adds a critical layer of convenience and security for users.

Figure 35 forgot password functionality (self-created)

During resetting the password or login with correct credential user is redirected to OTP page and then the system will send OTP from admin mail to the user's registered mail id using code in Figure 36. After successful verification, it will be redirected to the user account.

```
<div class="container">
    <h2>Enter OTP</h2>

    {% if error %}
        <p class="error">{{ error }}</p>
    {% endif %}
    {% if message %}
        <p class="message">{{ message }}</p>
    {% endif %}

    <form method="POST">
        <label for="otp">OTP:</label>
        <input type="text" id="otp" name="otp" placeholder="Enter OTP" required><br>
        <button type="submit">Verify OTP</button>
            <!-- Flash Messages -->
    {% with messages = get_flashed_messages(with_categories=true) %}
    {% if messages %}
        <ul class="flashes">
            --{%-for-category,-message-in-messages-%}
                <li class="{{ category }}">{{ message }}</li>
            {% endfor %}
        </ul>
    {% endif %}
{% endwith %}
    </form>
</div>
```

```
@app.route('/verify_otp', methods=['GET', 'POST'])
def verify_otp():
    email = request.args.get('email')

    if request.method == 'POST':
        if 'otp' in request.form:
            entered_otp = request.form['otp']
            stored_otp = otp_storage.get(email)

            if entered_otp == stored_otp:
                # OTP is correct, set user session and redirect to profile page
                session['user_email'] = email  # Store user email in session (or another identifier)
                del otp_storage[email]  # Optional: delete the OTP after use
                return redirect(url_for('profile'))  # Redirect to profile page
            else:
                # OTP is invalid, generate a new one and send it
                new_otp = str(random.randint(100000, 999999))
                otp_storage[email] = new_otp  # Store the new OTP

                # Send the new OTP via email
                try:
                    msg = Message('Your Login OTP', sender='jayy3685@gmail.com', recipients=[email])
                    msg.body = f'Your OTP is: {new_otp}. It is valid for 5 minutes.'
                    mail.send(msg)
                    return render_template('verify_otp.html', email=email, error="Invalid OTP. A new OTP has been sent.")
                except Exception as e:
                    return f"Failed to send OTP: {e}"

    return render_template('verify_otp.html', email=email)
```

Figure 36 Verify OTP page (self-created)

Once the user is logged in after successful OTP verification, then the user will redirect to the community account page, and it shows the profile page using code in Figure 37 of the user and displays the welcome message User can edit update their profile and also they can upload profile picture.

```
<!-- Profile Content -->
<div class="container">
    {% if user.profile_picture %}
        <img src="{{ url_for('static', filename='uploads/' + user.profile_picture) }}" alt="Profile Picture" class="profile-picture">
    {% else %}
        <img src="{{ url_for('static', filename='uploads/default.png') }}" alt="Default Profile Picture" class="profile-picture">
    {% endif %}

    <h1>Welcome, {{ user.username }}</h1>

    {% if message %}
        <p>{{ message }}</p>
    {% endif %}

    <form method="POST" enctype="multipart/form-data">
        <label>Username:</label>
        <input type="text" name="username" value="{{ user.username }}" required>

        <label>Mobile:</label>
        <input type="text" name="mobile" value="{{ user.mobile }}" required>

        <label>Age:</label>
        <input type="number" name="age" value="{{ user.age }}" required>

        <label>Gender:</label>
        <select name="gender">
            <option value="Male" {% if user.gender == "Male" %}selected{% endif %}>Male</option>
            <option value="Female" {% if user.gender == "Female" %}selected{% endif %}>Female</option>
        </select>

        <label>Profile Picture:</label>
        <input type="file" name="profile_picture">

        <button type="submit">Update Profile</button>
        <a href="{{ url_for('community_forum', email=user.email) }}">Go to Community Page</a>
    </form>

</div>
```

```
@app.route('/profile', methods=['GET', 'POST'])
def profile():
    email = session.get('user_email')
    user = User.query.filter_by(email=email).first()
    if not user:
        return redirect(url_for('login', error="User not found."))

    if request.method == 'POST':
        user.username = request.form['username']
        user.mobile = request.form['mobile']
        user.age = request.form['age']
        user.gender = request.form['gender']

        if 'profile_picture' in request.files and allowed_file(request.files['profile_picture'].filename):
            file = request.files['profile_picture']
            filename = secure_filename(file.filename)
            if not os.path.exists(app.config['UPLOAD_FOLDER']):
                os.makedirs(app.config['UPLOAD_FOLDER'])
            file.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))
            user.profile_picture = filename

        db.session.commit()
        return render_template('profile.html', user=user, message="Profile updated successfully.")

    return render_template('profile.html', user=user)
```

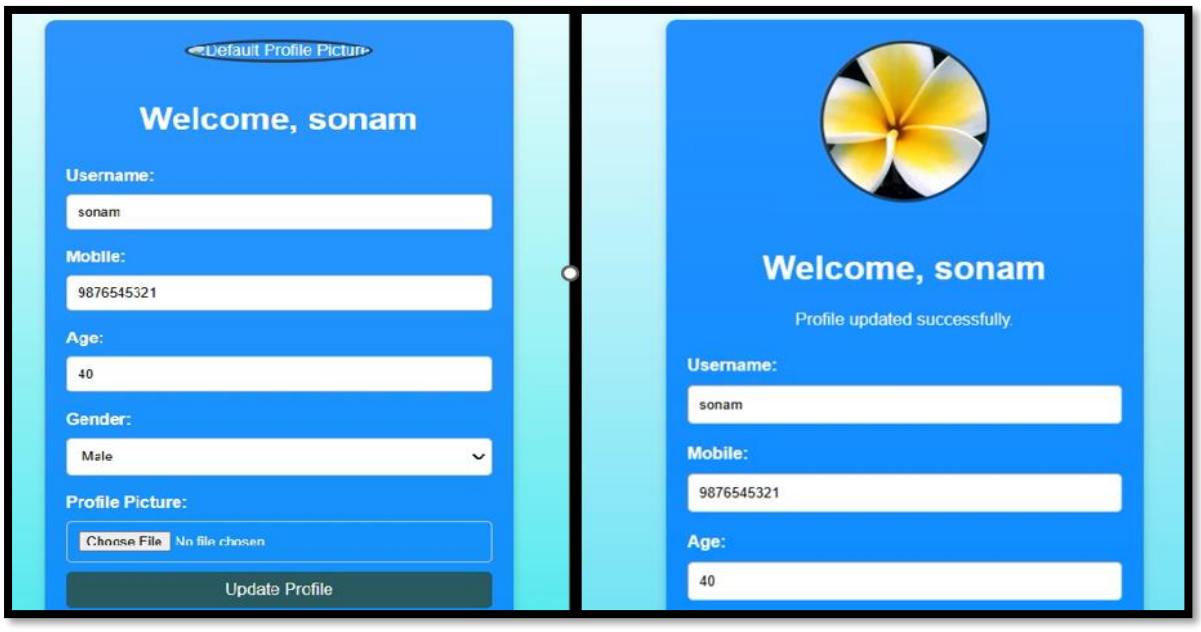


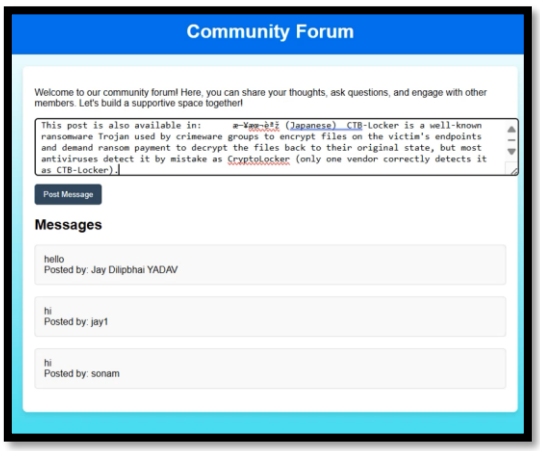
Figure 37 User profile page(self-created)

On profile page when user clicks on “go to community link” user is redirected to Community Forum page, where users can engage with the platform by posting their messages, sharing thoughts, or asking questions.

At the top of the forum, users can enter their messages in the input field and click the Post Message button to publish their content. Once posted, the messages are displayed below for everyone to see, creating an interactive and collaborative environment.

```
<!-- Main Content -->
<div class="container">
    <p>Welcome to our community forum! Here, you can share your thoughts, ask questions, and engage with other members. Let's build a supportive space together!</p>

    <!-- Post Message Form -->
    <form method="POST">
        <textarea name="message" rows="4" placeholder="Write your message here..." required></textarea><br>
        <button type="submit">Post Message</button>
    </form>

    <h2>Messages</h2>
    {% for message in messages %}
    <div class="message-box">
        <div class="content">{{ message.message_content }}</div>
        <div class="actions">
            <span>Posted by: {{ message.user.username }}</span>
        </div>
    </div>
    {% endfor %}


</div>
        <!-- Flash Messages -->
        {% with messages = get_flashed_messages(with_categories=true) %}
        {% if messages %}
            <ul class="flashes">
                {% for category, message in messages %}
                    <li class="{{ category }}">{{ message }}</li>
                {% endfor %}
            </ul>
        {% endif %}
    {% endwith %}
```

```
@app.route('/community_forum', methods=['GET', 'POST'])
def community_forum():
    email = session.get('user_email')
    if not email:
        return redirect(url_for('login', error="Please log in to access the community forum."))

    user = User.query.filter_by(email=email).first()
    if not user:
        return redirect(url_for('login', error="User not found."))

    if request.method == 'POST':
        message_content = request.form['message']

        clean_message = preprocess_text(message_content)
        vectorized_message = vectorizer.transform([clean_message]).toarray()
        prediction = model.predict(vectorized_message)[0]
        threat_type = label_encoder.inverse_transform([prediction])[0]
        print(threat_type)

        # List of threat types to check
        threat_keywords = [
            'malware', 'attack-pattern', 'time', 'identity', 'software',
            'threat-actor', 'location', 'tools', 'filepath',
            'vulnerability', 'url', 'domain', 'ipv4', 'campaign',
            'email', 'registrykey', 'sha1', 'infrastructure', 'md5', 'hash'
        ]

        # Check if the message is a threat
        is_threat = threat_type.lower() in threat_keywords

        if is_threat:
            # Store the threat in the AdminThreat table
            threat_message = AdminThreat(
                user_id=user.user_id,
                username=user.username,
                email=user.email,
                mobile=user.mobile,
                message_content=message_content,
                threat_type=threat_type
            )
            db.session.add(threat_message)
            db.session.commit()

            # Inform the user that the message was flagged and reported, including the threat type
            flash(f"Your message was flagged as a threat ({threat_type}) and reported to the admin.", "warning")
            return redirect(url_for('community_forum'))

        # If it's not a threat, post the message to the community forum
        new_message = CommunityMessage(user_id=user.id, message_content=message_content)
        db.session.add(new_message)
        db.session.commit()

        return render_template('community_forum.html', user=user, messages=CommunityMessage.query.all(), threat_type=None)

    # Display all messages
    messages = CommunityMessage.query.order_by(CommunityMessage.timestamp.desc()).all()
    return render_template('community_forum.html', user=user, messages=messages)
```



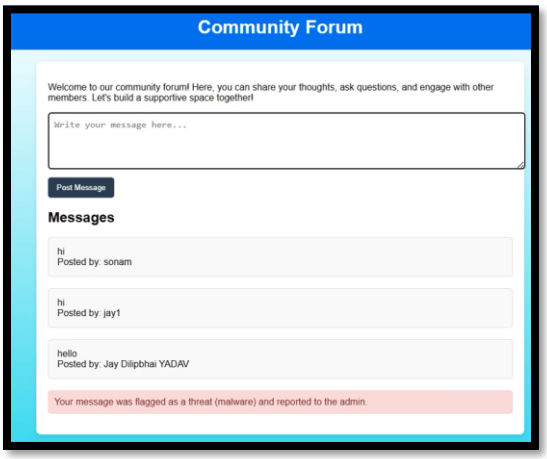


Figure 38 Community forum page(self-created)

Additionally, users can view their own posts alongside contributions from other community members. This functionality encourages open communication and interaction, making the platform a shared space for discussions and idea exchange among its users. The community forum page checks for threat message using pretrained model using code shown in Figure 38. If user clicks post message, then, then typed text will be cleaned, pre-processed, vectorized and then it will be predicted using Random Forest trained model.If message is a threat message it is reported to admin for further action. Logout functionality is also added to go back to login page.

4.3.7 Employee/admin page

The Admin/Employee page has two buttons as shown in in Figure 39 that is Login and Signup for Employees. There is no sign up for admin as the system has only one admin account with the following credentials

- Email ID: jayy3685@gmail.com
- ID: ADM
- Password: admin1234

```
<main>
    <section>
        <h2>Employee Login/Signup</h2>
        <a href="{{ url_for('employee_signup') }}">
            <button style="background-color: green; color: white; padding: 10px 20px; border: none; border-radius: 5px;">Signup</button>
        </a>
        <a href="{{ url_for('admin_login') }}">
            <button style="background-color: red; color: white; padding: 10px 20px; border: none; border-radius: 5px; margin-left: 10px;">Login</button>
        </a>
    </section>
</main>
```

Figure 39 admin/employee page with login signup buttons(self-created)

If an employee clicks on the Signup button employee is redirected to the Signup page that includes a form with some basic details as name, email id, gender etc. These fields include email validation and strong password validation (Figure 40).

```
<main>
    <form method="POST" action="{{ url_for('employee_signup') }}">
        <div>
            <label for="username">Username:</label>
            <input type="text" id="username" name="username" value="{{ form_data.get('username', '') }}" required>
        </div>
        <div>
            <label for="email">Email:</label>
            <input type="email" id="email" name="email" value="{{ form_data.get('email', '') }}" required>
            {% if error.get('email') %}
                <span class="error">{{ error['email'] }}</span>
            {% endif %}
        </div>
        <div>
            <label for="mobile">Mobile:</label>
            <input type="text" id="mobile" name="mobile" value="{{ form_data.get('mobile', '') }}" required>
        </div>
        <div>
            <label for="password">Password:</label>
            <input type="password" id="password" name="password" required>
        </div>
        <div>
            <label for="confirm_password">Confirm Password:</label>
            <input type="password" id="confirm_password" name="confirm_password" required>
            {% if error.get('password') %}
                <span class="error">{{ error['password'] }}</span>
            {% endif %}
        </div>
        <div>
            <label for="age">Age:</label>
            <input type="number" id="age" name="age" value="{{ form_data.get('age', '') }}" required>
        </div>
        <div>
            <label for="gender">Gender:</label>
            <select id="gender" name="gender" required>
                <option value="" disabled selected>Select Gender</option>
                <option value="Male" {% if form_data.get('gender') == 'Male' %}selected{% endif %}>Male</option>
                <option value="Female" {% if form_data.get('gender') == 'Female' %}selected{% endif %}>Female</option>
                <option value="Other" {% if form_data.get('gender') == 'Other' %}selected{% endif %}>Other</option>
            </select>
        </div>
```

```
@app.route('/employee/signup', methods=['GET', 'POST'])
def employee_signup():
    error = {}  # To store field-specific error messages
    form_data = {}  # To store form data to prepopulate fields

    if request.method == 'POST':
        # Retrieve form data
        username = request.form.get('username', '').strip()
        email = request.form.get('email', '').strip()
        mobile = request.form.get('mobile', '').strip()
        password = request.form.get('password', '')
        confirm_password = request.form.get('confirm_password', '')
        age = request.form.get('age', '')
        gender = request.form.get('gender', '')

        # Populate form_data dictionary
        form_data = {
            'username': username,
            'email': email,
            'mobile': mobile,
            'age': age,
            'gender': gender,
        }

        # Validation: Check if all required fields are provided
        if not username:
            error['username'] = "Username is required."
        if not email:
            error['email'] = "Email is required."
        if not mobile:
            error['mobile'] = "Mobile number is required."
        if not password:
            error['password'] = "Password is required."
        if not confirm_password:
            error['confirm_password'] = "Please confirm your password."
        if not age:
            error['age'] = "Age is required."
        if not gender:
            error['gender'] = "Gender is required."

        # Validation: Check for password match
        if password and confirm_password and password != confirm_password:
            error['password'] = "Passwords do not match."

        # Validation: Check if email is already registered
        if email and Employee.query.filter_by(email=email).first():
            error['email'] = "This email is already registered."

        # If there are errors, re-render the form with error messages and form data
        if error:
            return render_template('employee_signup.html', error=error, form_data=form_data)

        new_employee = Employee(
            username=username,
            email=email,
            mobile=mobile,
            password=generate_password_hash(password),
            age=int(age),
            gender=gender
        )
        db.session.add(new_employee)
        db.session.commit()

        # Redirect to signup success page after saving
        return redirect(url_for('signup_success'))

    # For GET request, pass empty error and form_data to the template
    return render_template('employee_signup.html', error={}, form_data={})
```

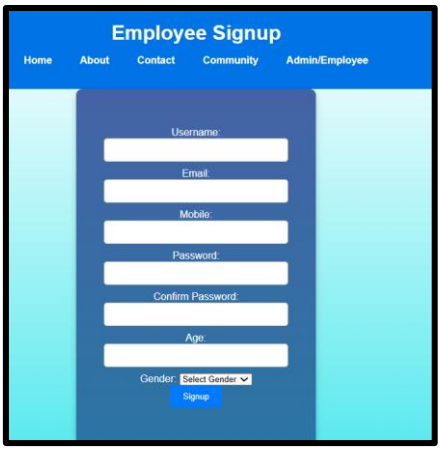


Figure 40 Employee Signup page (self-created)

After successfully signing up, the signup success message employee will receive with information that employee will receive their Employee ID and occupation details within 24 hours. The employee id and occupation are assigned by admin (Figure 41).

```python
# Employee signup success page route
@app.route('/employee/signup_success', methods=['GET'])
def signup_success():
    return render_template(
        'signup_success.html',
        message="Signup successful! You will receive your Employee ID and occupation detail  within 24 hours."
    )
```

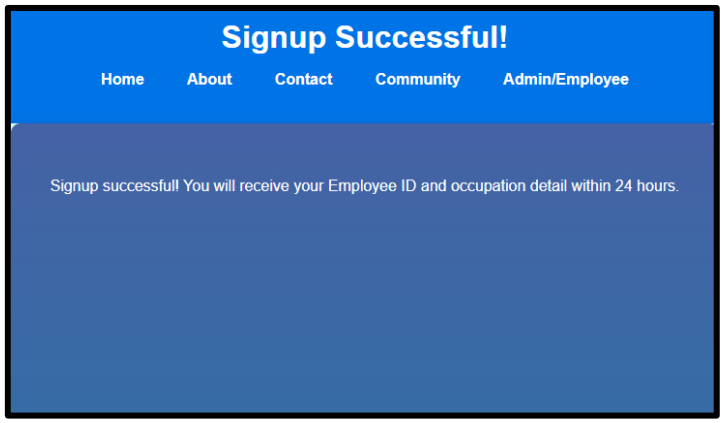


Figure 41 Employee Signup-success (self-created)

If employee click on login button employee is redirected to the login page as shown in before. Common login page is created using code shown in Figure 42 for employee and system admin by placing a dropdown to select role (employee or system admin). Admin can login using below credentials

- Email ID: jayy3685@gmail.com
- ID: ADM
- Password: admin1234.

```html
<div class="content">
    <form method="POST">
        <label for="email">Email Address:</label>
        <input type="email" id="email" name="email" placeholder="Enter your email" required><br><br>

        <label for="password">Password:</label>
        <input type="password" id="password" name="password" placeholder="Enter your password" required><br><br>

        <label for="role">Select Role:</label>
        <select id="role" name="role" required>
            <option value="admin">System Admin</option>
            <option value="system_admin">Employee</option>
        </select><br><br>

        <label for="employee_id">Employee ID:</label>
        <input type="text" id="employee_id" name="employee_id" placeholder="Enter your Employee ID" required><br><br>

        <button type="submit">Login</button>
    </form>

</div>
```

```python
@app.route('/admin', methods=['GET', 'POST'])
def admin():
    try:
        # Check if the admin is logged in
        if not session.get('admin'):
            return redirect(url_for('admin_login'))

        # Admin credentials
        ADMIN_ID = "ADM"
        ADMIN_EMAIL = 'jayy3685@gmail.com'
        ADMIN_PASSWORD = 'admin1234'

        # Handle login logic if not logged in yet
        if request.method == 'POST':
            if 'logout' in request.form:  # Handle admin logout
                session.pop('admin', None)
                return redirect(url_for('admin_login'))

            if 'employee_logout' in request.form:  # Handle employee logout
                session.pop('employee', None)
                return redirect(url_for('admin_login'))

            email = request.form.get('email')
            password = request.form.get('password')

            # Admin login validation
            if email == ADMIN_EMAIL and password == ADMIN_PASSWORD:
                session['admin'] = True
            else:
                return render_template('admin.html', error="Invalid credentials.")
```

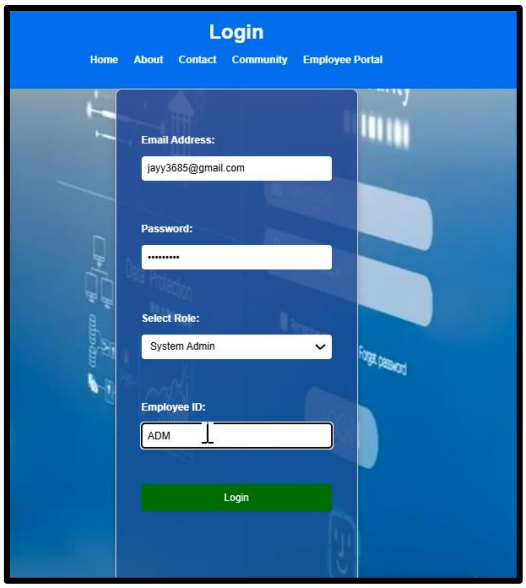


Figure 42 Admin login (self-created)

After successful login system admin is redirected to admin portal with following functionalities as shown in Figure 43.

- Block user for malware posts on community page
- Assign id and occupation to employees
- Show visuals related to threat data
- Edit and delete employee's detail
- Edit user detail
- Block the user for malware query through contact page.

```
<header>
    <div class="container">
        <h1>Admin Dashboard</h1>
        <nav>
            <ul>
                <li><a href="{{ url_for('admin') }}">Malware posts</a></li>
                <li><a href="{{ url_for('show_visuals') }}">Show Visuals</a></li>
                <li><a href="{{ url_for('admin_assignid') }}">Assign Employee ID</a></li>
                <li><a href="{{ url_for('show_malware') }}"> Malware Messages</a></li>
                <li><a href="{{ url_for('employee_details') }}">Employee Details</a></li>
                <li><a href="{{ url_for('user_details') }}">User Details</a></li>
            </ul>
        </nav>
    </div>
</header>
```

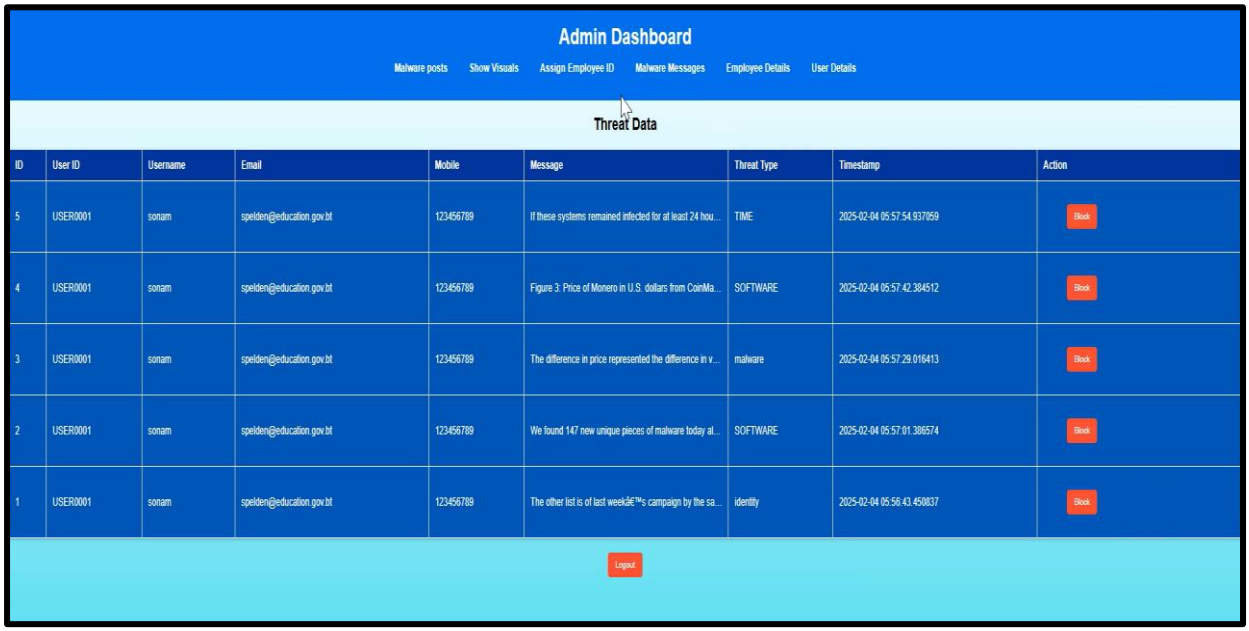


Figure 43 Admin dashboard (self-created)

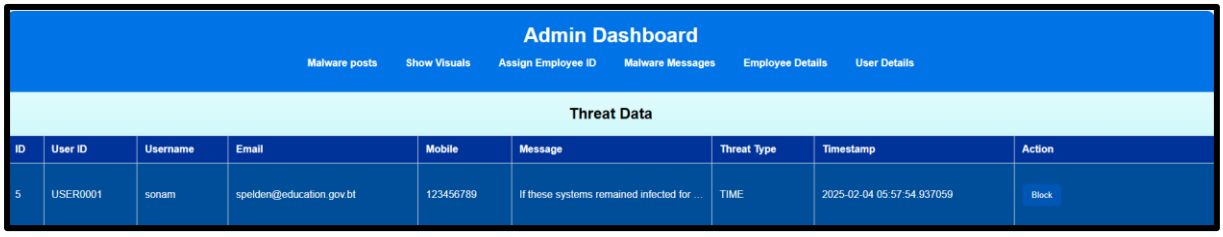
On admin dashboard the landing page have details in the form of a table including all the details of user who has posted malware message on community forum with a block button as shown in Figure 43.

4.3.8 Block user for malware posts on community page

Admin has right to block the user by clicking on block button for posting malware messages on community page. Once the user is blocked user won't be able to login again by checking is_blocked status in user table as shown in code in Figure 44.

```
<table>
    <thead>
        <tr>
            <th>ID</th>
            <th>User ID</th>
            <th>Username</th>
            <th>Email</th>
            <th>Mobile</th>
            <th>Message</th>
            <th>Threat Type</th>
            <th>Timestamp</th>
            <th>Action</th>
        </tr>
    </thead>
    <tbody>
        {% for threat in threats %}
            <tr>
                <td>{{ threat.id }}</td>
                <td>{{ threat.user_id }}</td>
                <td>{{ threat.username }}</td>
                <td>{{ threat.email }}</td>
                <td>{{ threat.mobile }}</td>
                <td>{{ threat.message_content }}</td>
                <td>{{ threat.threat_type }}</td>
                <td>{{ threat.timestamp }}</td>
                <td>
                    <form method="POST" action="{{ url_for('block_user') }}">
                        <input type="hidden" name="user_id" value="{{ threat.user_id }}">
                        <button type="submit" class="btn btn-danger">
                            Block
                        </button>
                    </form>
                </td>
            </tr>
        {% endfor %}
    </tbody>
</table>
```

```
@app.route('/block_user', methods=['POST'])
def block_user():
    if request.method == 'POST':
        user_id_to_block = request.form.get('user_id')

        try:
            # Use update() to directly update the database
            result = db.session.query(User).filter_by(user_id=user_id_to_block).update({
                'is_blocked': True
            })
            db.session.commit()

            if result:
                flash(f"User {user_id_to_block} has been blocked.", "success")
                return redirect(url_for('admin'))  # Redirect back to the admin page
            else:
                flash("User not found.", "error")
                return redirect(url_for('admin'))

        except SQLAlchemyError as e:
            logging.error(f"Error blocking user with ID {user_id_to_block}: {e}")
            flash("Error blocking user.", "error")
            return redirect(url_for('admin'))

    return redirect(url_for('admin'))
```



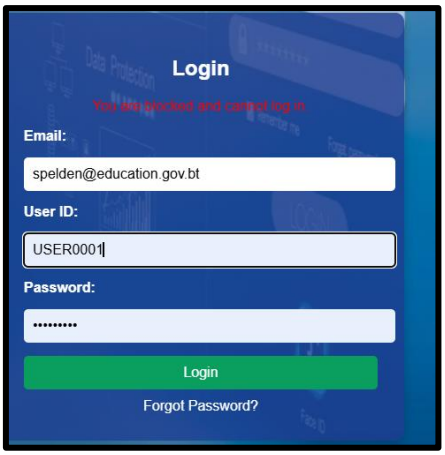


Figure 44 Admin block user functionality(self-created)

4.3.9 Show visuals related to threat data

If admin click on show visuals admin is redirected to show visual page where bar chart is plotted for different types of threat. Also there is a hyperlink as given in Figure 45 below to navigate back to the admin dashboard page.

```
<header>
    <h1>Threat Visuals</h1>
</header>

<main>
    {% if error %}
        <p class="error-message">{{ error }}</p>
    {% else %}
        <h2>Bar Chart: Threats Detected</h2>
        <img src="{{ chart_url }}" alt="Bar Chart">
    {% endif %}

    <p><a href="{{ url_for('admin') }}">Back to Admin</a></p>
</main>
```

```
@app.route('/show_visuals', methods=['GET', 'POST'])
def show_visuals():
    try:
        # Fetch threats from the database
        threats = AdminThreat.query.all()

        # Count threats by type
        threat_counts = {}
        for threat in threats:
            threat_counts[threat.threat_type] = threat_counts.get(threat.threat_type, 0) + 1

        # Generate Bar Chart
        fig = Figure()
        ax = fig.subplots()
        ax.bar(threat_counts.keys(), threat_counts.values(), color='blue')
        ax.set_title("Threats Detected Count")
        ax.set_xlabel("Threat Type")
        ax.set_ylabel("Count")

        # Convert figure to base64 string
        img = io.BytesIO()
        fig.savefig(img, format="png")
        img.seek(0)
        chart_url = base64.b64encode(img.getvalue()).decode('utf8')

        # Render visuals template
        return render_template('visuals.html', chart_url=chart_url)

    except Exception as e:
        logging.error(f"Error generating visuals: {e}")
        return render_template('visuals.html', error="An error occurred while generating visuals.")
```

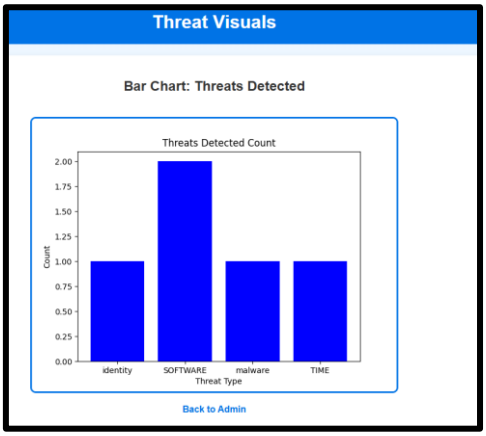


Figure 45 Admin Show visual functionality (self-created)

4.3.10 Assign employee id and occupation

After successful signup of employee all the details related to employees are displayed on assign employee id page in the form of table on admin portal. Admin will assign employee id and occupation manually by writing in text box as shown in Figure 46 given below. Employees will get email with employee id and occupation detail that they can use during login to the employee dashboard.

```
<section>
    {% endwith %}
    <table>
        <thead>
            <tr>
                <th>ID</th>
                <th>Username</th>
                <th>Email</th>
                <th>Mobile</th>
                <th>Age</th>
                <th>Gender</th>
                <th>Occupation</th>
                <th>Generated ID</th>
                <th>Action</th>
            </tr>
        </thead>
        <tbody>
            {% for employee in employees %}
                <tr>
                    <td>{{ employee.id }}</td>
                    <td>{{ employee.username }}</td>
                    <td>{{ employee.email }}</td>
                    <td>{{ employee.mobile }}</td>
                    <td>{{ employee.age }}</td>
                    <td>{{ employee.gender }}</td>
                    <td>{{ employee.occupation }}</td>
                    <td>{{ employee.employee_id if employee.employee_id else 'Not Assigned' }}</td>
                    <td>
                        {% if not employee.employee_id %}
                            <form method="POST" action="{{ url_for('admin_assignid') }}">
                                <input type="hidden" name="employee_id" value="{{ employee.id }}">
                                <label for="occupation">Occupation:</label>
                                <input type="text" id="occupation" name="occupation" value="{{ employee.occupation or '' }}" required>
                                <button type="submit">Assign ID</button>
                            </form>
                        {% else %}
                            <span>Assigned</span>
                        {% endif %}
                    </td>
                </tr>
            {% endfor %}
        </tbody>
```

```
@app.route('/admin/assignid', methods=['GET', 'POST'])
def admin_assignid():
    employees = Employee.query.all()  # Fetch all employees from the database

    if request.method == 'POST':
        employee_id = request.form.get('employee_id')
        occupation = request.form.get('occupation')  # Get the occupation from the form
        employee = Employee.query.get(employee_id)

        if employee:
            # Generate a unique employee ID (you can modify the logic for ID generation)
            generated_id = f"E-{employee.id:04d}"  # Example: "E-0001"

            try:
                # Update the employee record with the generated ID and occupation
                employee.employee_id = generated_id
                employee.occupation = occupation  # Save the occupation

                # Commit the changes to the database
                db.session.commit()

                # Send email with the assigned ID and occupation
                msg = Message(
                    subject="Your Employee ID and Occupation Assignment",
                    sender=app.config['MAIL_USERNAME'],
                    recipients=[employee.email]
                )
                msg.body = f"""
                    Dear {employee.username},

                    Your Employee ID has been assigned: {generated_id}
                    Your Occupation: {occupation}

                    Thank you for registering.

                    Regards,
                    Admin Team
                """
                mail.send(msg)

                flash(f"Employee ID and occupation assigned, email sent to {employee.email}", "success")
            except Exception as e:
                # Rollback the transaction if an error occurs
                db.session.rollback()
                flash(f"Error updating database: {str(e)}", "danger")
        else:
            flash("Employee not found.", "danger")

    return render_template('admin_assignid.html', employees=employees)
```

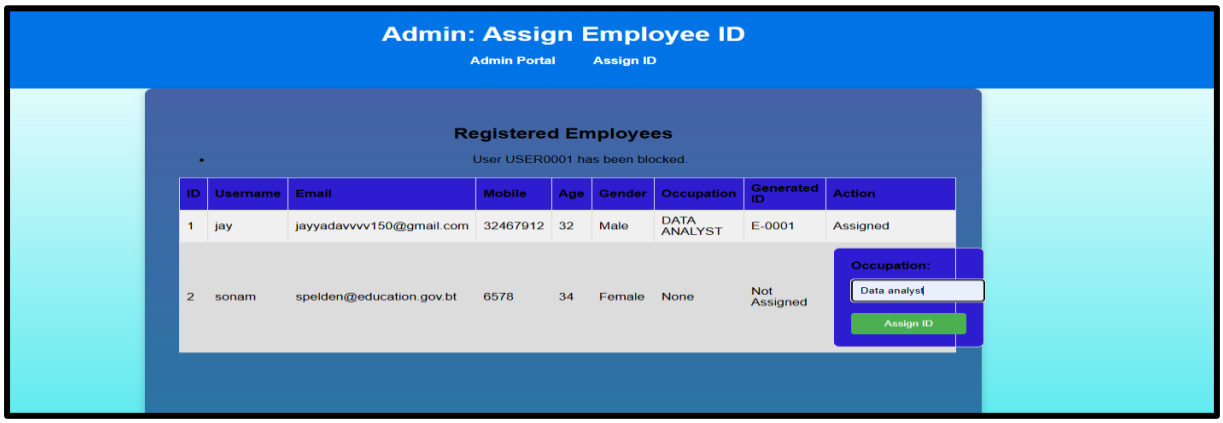


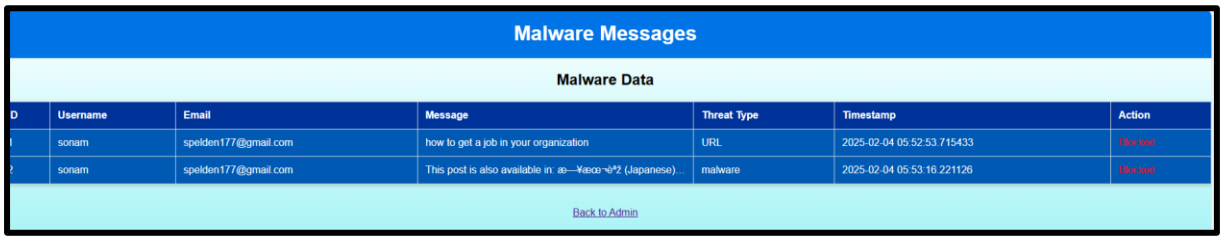
Figure 46 Admin assigning employee id and occupation (self-created)

4.3.11 Block the user for malware query through contact page

Admin malware query page displays the details of users who has sent malware query through contact page with block button. When admin click on block user is blocked from sending query through contact page as shown in Figure 47.

```
<div class="content">
    <h2>Malware Data</h2>

    <!-- Show Malware Table -->
    <table>
        <thead>
            <tr>
                <th>ID</th>
                <th>Username</th>
                <th>Email</th>
                <th>Message</th>
                <th>Threat Type</th>
                <th>Timestamp</th>
                <th>Action</th>
            </tr>
        </thead>
        <tbody>
            {% for malware in malware %}
                <tr>
                    <td>{{ malware.id }}</td>
                    <td>{{ malware.username }}</td>
                    <td>{{ malware.email }}</td>
                    <td>{{ malware.message_content }}</td>
                    <td>{{ malware.threat_type }}</td>
                    <td>{{ malware.timestamp }}</td>
                    <td>
{% if not malware.is_blocked %}
    <form method="POST">
        <input type="hidden" name="user_id" value="{{ malware.id }}">
        <input type="hidden" name="action" value="block">
        <button type="submit" class="btn btn-danger">Block User</button>
    </form>
{% else %}
    <span style="color: red;">Blocked</span>
{% endif %}
td>

                </tr>
            {% endfor %}
        </tbody>
    </table>
</div>
<p><a href="{{ url_for('admin') }}">Back to Admin</a></p>
```

```
    db.session.commit()

    # **Automatic Blocking (Optional)**
    user_threats = MalwareDB.query.filter_by(email=email).count()
    if user_threats >= 3:  # If user has sent 3+ threats, block them
        user = MalwareDB.query.filter_by(email=email).first()
        user.is_blocked = True
        db.session.commit()
        message = "You have been blocked due to multiple flagged messages."
        return render_template('contact.html', block_message=message)
```



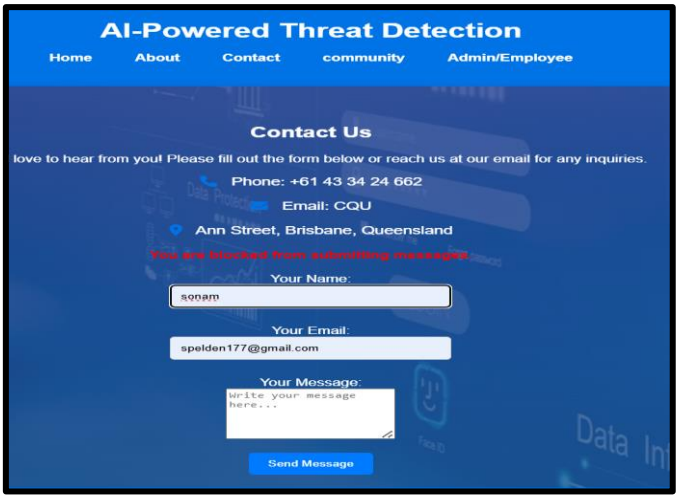
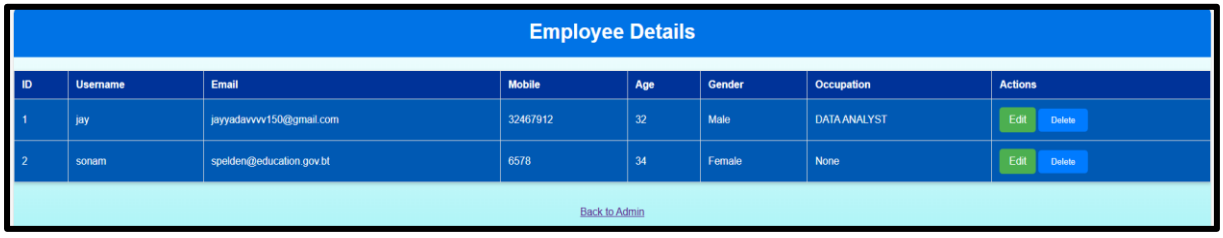


Figure 47 Admin blocking user from sending malware query (self-created)

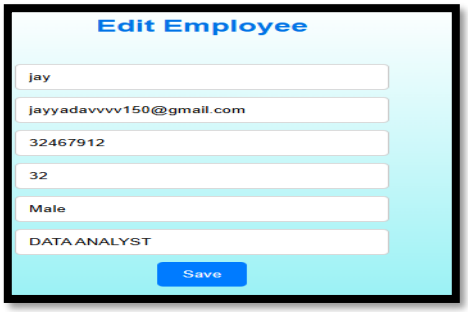
4.3.12 Edit and delete employee's detail

The admin page also allows editing and deletion of the employee's relevant details. The admins could edit the employee form as would be required or delete it permanently. This will allow for correct and updated management of employees within the system using the code given in Figure 48.

```
<body>
    <header>
        <div class="container">
            <h1>Employee Details</h1>
        </div>
    </header>

<table>
    <tr>
        <th>ID</th>
        <th>Username</th>
        <th>Email</th>
        <th>Mobile</th>
        <th>Age</th>
        <th>Gender</th>
        <th>Occupation</th>
        <th>Actions</th>
    </tr>
    {% for employee in employees %}
    <tr>
        <td>{{ employee.id }}</td>
        <td>{{ employee.username }}</td>
        <td>{{ employee.email }}</td>
        <td>{{ employee.mobile }}</td>
        <td>{{ employee.age }}</td>
        <td>{{ employee.gender }}</td>
        <td>{{ employee.occupation }}</td>
        <td>
            <a href="{{ url_for('edit_employee', id=employee.id) }}" class="btn">Edit</a>

            <form action="{{ url_for('delete_employee', id=employee.id) }}" method="post" style="display:inline;">
                <button type="submit" onclick="return confirm('Are you sure?')">Delete</button>
            </form>
        </td>
    </tr>
    {% endfor %}
</table>
<p><a href="{{ url_for('admin') }}">Back to Admin</a></p>
<footer>
    <p>© 2024 AI Threat Detection Solutions | All Rights Reserved</p>
</footer>
</body>
```




```
<h2>Edit Employee</h2>
<form method="post">
    <input type="text" name="username" value="{{ employee.username }}" required>
    <input type="email" name="email" value="{{ employee.email }}" required>
    <input type="text" name="mobile" value="{{ employee.mobile }}" required>
    <input type="number" name="age" value="{{ employee.age }}" required>
    <input type="text" name="gender" value="{{ employee.gender }}" required>
    <input type="text" name="occupation" value="{{ employee.occupation }}">
    <button type="submit">Save</button>
</form>
<p><a href="{{ url_for('admin') }}">Back to Admin</a></p>
```



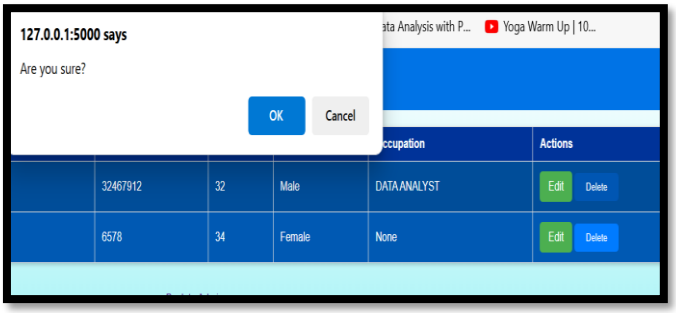
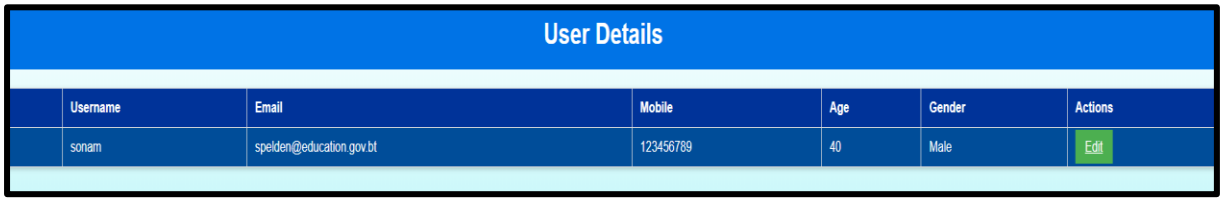


Figure 48 Admin edit and delete employee's functionality (self-created)

There are two buttons edit and delete, if admin clicks on edit admin is redirected to edit employee page where admin can change any field related to employee information but if admin click on delete it will permanently delete employee from system record.

4.3.13 Editing user detail

```
<table>
    <tr>
        <th>ID</th>
        <th>Username</th>
        <th>Email</th>
        <th>Mobile</th>
        <th>Age</th>
        <th>Gender</th>
        <th>Actions</th>
    </tr>
    {% for user in users %}
    <tr>
        <td>{{ user.id }}</td>
        <td>{{ user.username }}</td>
        <td>{{ user.email }}</td>
        <td>{{ user.mobile }}</td>
        <td>{{ user.age }}</td>
        <td>{{ user.gender }}</td>
        <td>
            <a href="{{ url_for('edit_user', id=user.id) }}" class="btn">Edit</a>
        </td>
    </tr>
    {% endfor %}
</table>
<p><a href="{{ url_for('admin') }}">Back to Admin</a></p>
```




```
@app.route('/admin/edit_user/<int:id>', methods=['GET', 'POST'])
def edit_user(id):
    if not session.get('admin'):
        return redirect(url_for('admin_login'))

    user = User.query.get_or_404(id)

    if request.method == 'POST':
        user.username = request.form['username']
        user.email = request.form['email']
        user.mobile = request.form['mobile']
        user.age = request.form['age']
        user.gender = request.form['gender']

        db.session.commit()
        return redirect(url_for('user_details'))

    return render_template('edit_user.html', user=user)
```

Figure 49 Admin edit user detail functionality (self-created)

On this page all the relevant information related to users posting messages on community page is displayed using code in Figure 49 in the form of table with edit button. When admin clicks edit button admin is redirected to edit user page where information related to user can be updated.

On employee login page if employee selects the role as employee rather than system admin and enters the credential received on mail (employee id) then employee is navigated to employee dashboard.

```
<div class="content">
    <form method="POST">
        <label for="email">Email Address:</label>
        <input type="email" id="email" name="email" placeholder="Enter your email" required><br><br>

        <label for="password">Password:</label>
        <input type="password" id="password" name="password" placeholder="Enter your password" required><br><br>

        <label for="role">Select Role:</label>
        <select id="role" name="role" required>
            <option value="admin">System Admin</option>
            <option value="system_admin">Employee</option>
        </select><br><br>

        <label for="employee_id">Employee ID:</label>
        <input type="text" id="employee_id" name="employee_id" placeholder="Enter your Employee ID" required><br><br>

        <button type="submit">Login</button>
    </form>

</div>
```

```
    elif role == 'employee':
        # Query the database directly using SQLAlchemy
        employee = db.session.execute(
            db.select(Employee).filter_by(email=email, employee_id=employee_id)
        ).scalars().first()

        if employee:
            print(f"Employee found: {employee.username} (ID: {employee.employee_id})")

            # Check if the password matches
            if check_password_hash(employee.password, password):
                session['employee'] = employee.id  # Save employee ID in session
                return redirect(url_for('employee_dashboard'))  # Redirect to employee dashboard
            else:
                return render_template('admin_login.html', error="Invalid password for System Admin.")
        else:
            return render_template('admin_login.html', error="Employee not found.")  # Employee not found

    # Default error for invalid role selection or credentials
    return render_template('admin_login.html', error="Invalid role or credentials.")
```

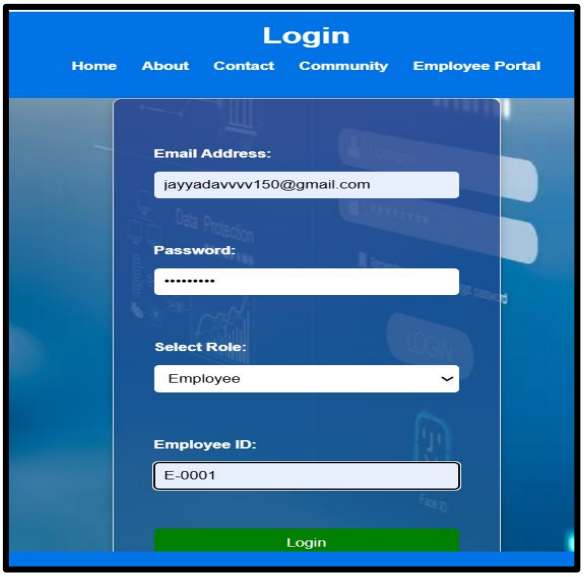


Figure 50 Employee login(self-created)

Employee dashboard shows the users detail in the form of table with block button who has posted the malware message on community forum (Figure 50). Employee dashboard has following functionality:

- Block user for malware posts on community page
- Show visuals related to threat data
- Block the user for malware query through contact page.

4.3.14 Block user for malware posts on community page

Employees also have right to block the user by clicking on block button for posting malware messages on community page. Once the user is blocked user won't be able to login again by checking is_blocked status in user table as shown in code in Figure 51.

```
<table>
    <thead>
        <tr>
            <th>ID</th>
            <th>User ID</th>
            <th>Username</th>
            <th>Email</th>
            <th>Mobile</th>
            <th>Message</th>
            <th>Threat Type</th>
            <th>Timestamp</th>
            <th>Action</th>
        </tr>
    </thead>
    <tbody>
        {% for threat in threats %}
        <tr>
            <td>{{ threat.id }}</td>
            <td>{{ threat.user_id }}</td>
            <td>{{ threat.username }}</td>
            <td>{{ threat.email }}</td>
            <td>{{ threat.mobile }}</td>
            <td>{{ threat.message_content }}</td>
            <td>{{ threat.threat_type }}</td>
            <td>{{ threat.timestamp }}</td>
            <td>
                <form method="POST" action="{{ url_for('emp_block_user') }}">
                    <input type="hidden" name="user_id" value="{{ threat.user_id }}">
                    <button type="submit" class="btn btn-danger">
                        Block
                    </button>
                </form>
            </td>
        </tr>
        {% endfor %}
    </tbody>
</table>
```

```
@app.route('/employee/block_user', methods=['POST'])
def emp_block_user():
    if request.method == 'POST':
        user_id_to_block = request.form.get('user_id')

        try:
            # Use update() to directly update the database
            result = db.session.query(User).filter_by(user_id=user_id_to_block).update({
                'is_blocked': True
            })
            db.session.commit()

            if result:
                flash(f"User {user_id_to_block} has been blocked.", "success")
                return redirect(url_for('employee_dashboard'))  # Redirect back to the admin page
            else:
                flash("User not found.", "error")
                return redirect(url_for('admin'))

        except SQLAlchemyError as e:
            logging.error(f"Error blocking user with ID {user_id_to_block}: {e}")
            flash("Error blocking user.", "error")
            return redirect(url_for('aemployee_dashboard'))
```

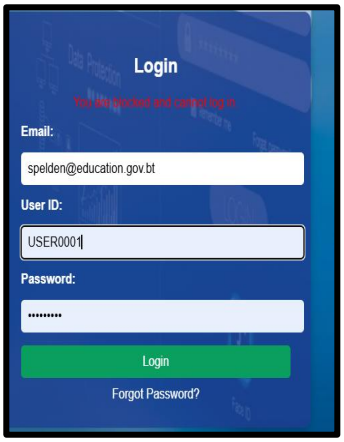
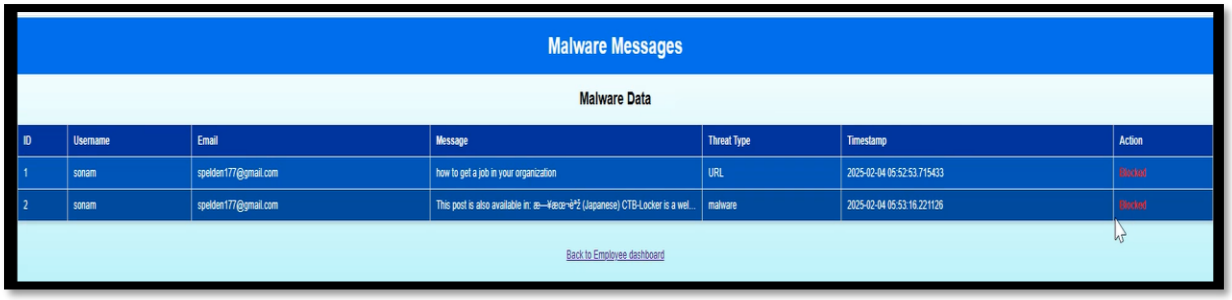


Figure 51 Employee block user functionality (self-created)

4.3.15 Block the user for malware query through contact page

Employee malware query page displays the details of users who has sent malware query through contact page with block button. When employee click on block user is blocked from sending query through contact page as shown in Figure 52.

```
        <table>
            <thead>
                <tr>
                    <th>ID</th>
                    <th>Username</th>
                    <th>Email</th>
                    <th>Message</th>
                    <th>Threat Type</th>
                    <th>Timestamp</th>
                    <th>Action</th>
                </tr>
            </thead>
            <tbody>
                {% for malware in malware %}
                    <tr>
                        <td>{{ malware.id }}</td>
                        <td>{{ malware.username }}</td>
                        <td>{{ malware.email }}</td>
                        <td>{{ malware.message_content }}</td>
                        <td>{{ malware.threat_type }}</td>
                        <td>{{ malware.timestamp }}</td>
                        <td>
    {% if not malware.is_blocked %}
        <form method="POST">
            <input type="hidden" name="user_id" value="{{ malware.id }}">
            <input type="hidden" name="action" value="block">
            <button type="submit" class="btn btn-danger">Block User</button>
        </form>
    {% else %}
        <span style="color: red;">Blocked</span>
    {% endif %}
</td>

                    </tr>
                {% endfor %}
            </tbody>
        </table>
    </div>
    <p><a href="{{ url_for('employee_dashboard') }}">Back to Employee dashboard</a></p>
```




```
    db.session.commit()

    # **Automatic Blocking (Optional)**
    user_threats = MalwareDB.query.filter_by(email=email).count()
    if user_threats >= 3:  # If user has sent 3+ threats, block them
        user = MalwareDB.query.filter_by(email=email).first()
        user.is_blocked = True
        db.session.commit()
        message = "You have been blocked due to multiple flagged messages."
        return render_template('contact.html', block_message=message)
```

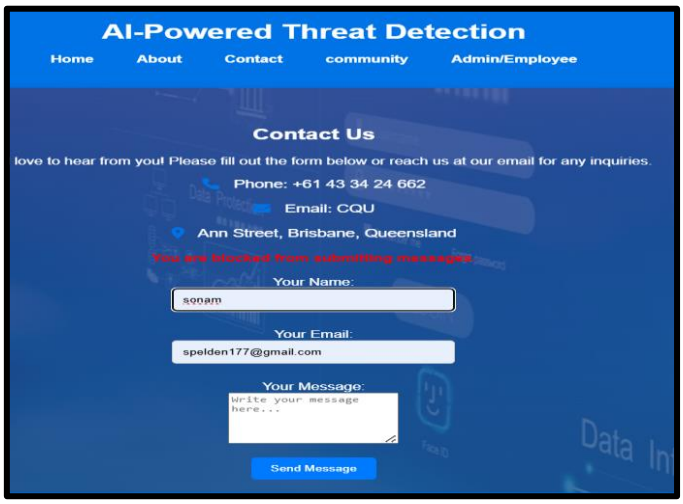


Figure 52 Block malware query user (self-created)

4.3.16 Show visuals related to threat data

If employee click on show visuals employee is redirected to show visual page where bar chart is plotted for different types of threat. Also there is a hyperlink as given in Figure 53 below to navigate back to the admin dashboard page.

```
<header>
    <h1>Threat Visuals</h1>
</header>

<main>
    {% if error %}
        <p class="error-message">{{ error }}</p>
    {% else %}
        <h2>Bar Chart: Threats Detected</h2>
        <img src="{{ chart_url }}" alt="Bar Chart">
    {% endif %}

    <p><a href="{{ url_for('employee_dashboard') }}">Back to Employee dashboard</a></p>
</main>
```

```
@app.route('/employee/show_visuals', methods=['GET', 'POST'])
def employee_show_visuals():
    try:
        # Fetch threats from the database
        threats = AdminThreat.query.all()

        # Count threats by type
        threat_counts = {}
        for threat in threats:
            threat_counts[threat.threat_type] = threat_counts.get(threat.threat_type, 0) + 1

        # Generate Bar Chart
        fig = Figure()
        ax = fig.subplots()
        ax.bar(threat_counts.keys(), threat_counts.values(), color='blue')
        ax.set_title("Threats Detected Count")
        ax.set_xlabel("Threat Type")
        ax.set_ylabel("Count")

        # Convert figure to base64 string
        img = io.BytesIO()
        fig.savefig(img, format="png")
        img.seek(0)
        chart_url = base64.b64encode(img.getvalue()).decode('utf8')

        # Render visuals template
        return render_template('visualsE.html', chart_url=chart_url)

    except Exception as e:
        logging.error(f"Error generating visuals: {e}")
        return render_template('visualsE.html', error="An error occurred while generating visuals.")
```

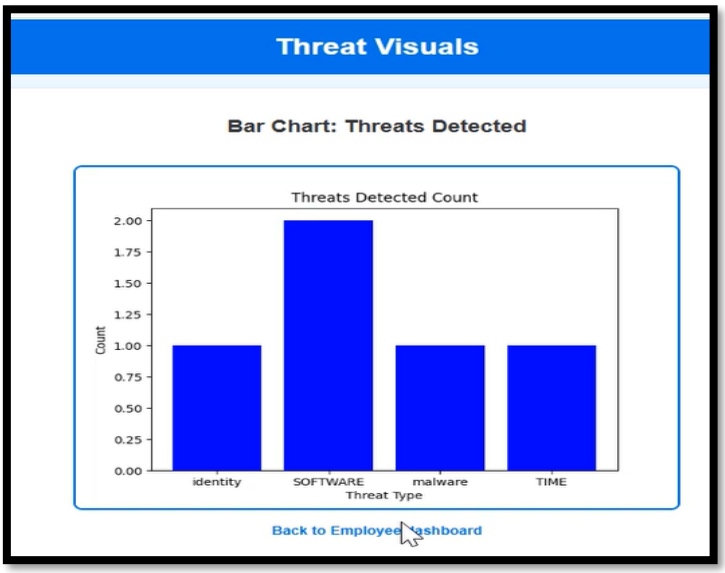


Figure 53 Show visual functionality on employee dashboard (self-created)

The system used machine learning techniques and secure AWS services into the applied AI-powered threat detection system of real-time cybersecurity solutions. By using different AI models such as Random Forest, SVM, Logistic Regression, and ANN for training and then selecting the model that achieves outstanding accuracy in detecting threats from network traffic and text data enhances system accuracy. The system integrates with the AWS infrastructure to achieve scalability, security, and

efficient data storage. The Flask web application ensured easy interaction, with multi-layer authentication, community forums, and a comprehensive admin panel with multiple functionalities like blocking user from posting threat data and manging employees' information in the organization. Every page where user can post, or query is threat protected using best AI model. So, it can be summarized that methodology used is a scalable, efficient, and secure approach allowing for the proactive identification and remediation of attacks in cybersecurity.

4.4 Usability

Feature	Description
Responsive Design	Utilizes HTML, CSS, and JavaScript to provide a user-friendly and responsive interface.
Intuitive Navigation	Easy navigation through homepage, contact, community, and admin/employee pages.
Role-Based Access	Separate interfaces for users, employees, and admin ensure clarity and usability.
Form Validations	Email and password validation in login and signup forms enhance usability.
Error Handling	Features like forgot password and OTP resend ensure smooth user experience.

4.5 Security

Feature	Description
AI-Powered Threat Detection	Real-time analysis of messages and queries using advanced AI to detect threats.
Two-Factor Authentication	Secure OTP-based authentication for employees and admin roles.
Data Encryption	Secure storage of user credentials using hashing techniques like generate_password_hash.
AWS VPC and IAM Roles	VPC isolates resources, and IAM roles manage secure access permissions.
Threat Management	Admins and employees can block users and review flagged messages for enhanced security.

4.6 Performance

Feature	Description
Efficient Backend	Flask framework ensures efficient handling of user inputs and API integration.
Optimized Models	Pre-trained Random Forest and other ML models enable fast and accurate threat detection.
AWS Scalability	EC2 instances and S3 storage dynamically scale to meet application demand.
Low Latency	AI predictions processed and delivered in real-time for a seamless experience.
Load Balancing	AWS infrastructure supports high performance under increased user traffic.

4.7 Reliability

Feature	Description
AWS Backup	Scheduled backups through AWS Backup ensure disaster recovery and data restoration.
Error Reporting	Logs failed login attempts and flagged messages for monitoring and remediation.
Redundancy	Elastic IP ensures availability during system updates or failures.
Health Monitoring	Regular EC2 instance status checks to ensure system reliability.
Consistent Functionality	Maintains robust operations across all user roles and scenarios for uninterrupted service.

The AI-powered threat detection web application solution delivers a complete cyber security solution. The system uses AI models and scalable AWS infrastructure to detect and mitigate threats in real time. Its usability ensures a seamless experience for users across roles, while security standards are met via features like two-factor authentication, encrypted data storage, and role-based access. Flask with pre-trained models like Random Forest work well and respond quickly with high user traffic. AWS services like EC2, S3, and frequent backup ensure system uptime and data recovery. Dynamic and proactive threat management starts with this secure, scalable, and user-friendly cybersecurity solution.

5 Design of Machine Learning Algorithms

5.1 Introduction

Machine learning models are crucial in data analysis and predictions. The work presented compared four different models: Artificial Neural Network (ANN), Random Forest, Support Vector Machine (SVM), and Logistic Regression. Every model has its strengths and weaknesses, and is evaluated based on accuracy, precision, recall, and F1-score. We are using deep learning Artificial Neural Network (ANN) model, Random Forest, support vector machine and logistic regression models. In the bottom detailed explanation about each algorithm is provided (Oliynyk, Mayer & Rauber 2023).

5.2 Machine Learning Models

5.2.1 Random Forest

Random Forest is an ensemble learning model which is constructed by building many decision trees and combining their output for classification purpose (Ghiasi & Zendehboudi, 2020). Each tree is trained on a random subset of data and features, reducing the overfitting and improving generalization. The final prediction in random forest is made based on majority voting across different trees. Here random forest model is trained with 100 decision trees where n estimators define the number of trees. This ensemble method improves accuracy, reduce overfitting and robust to noise data. A complete structure of random forest model is given as below.

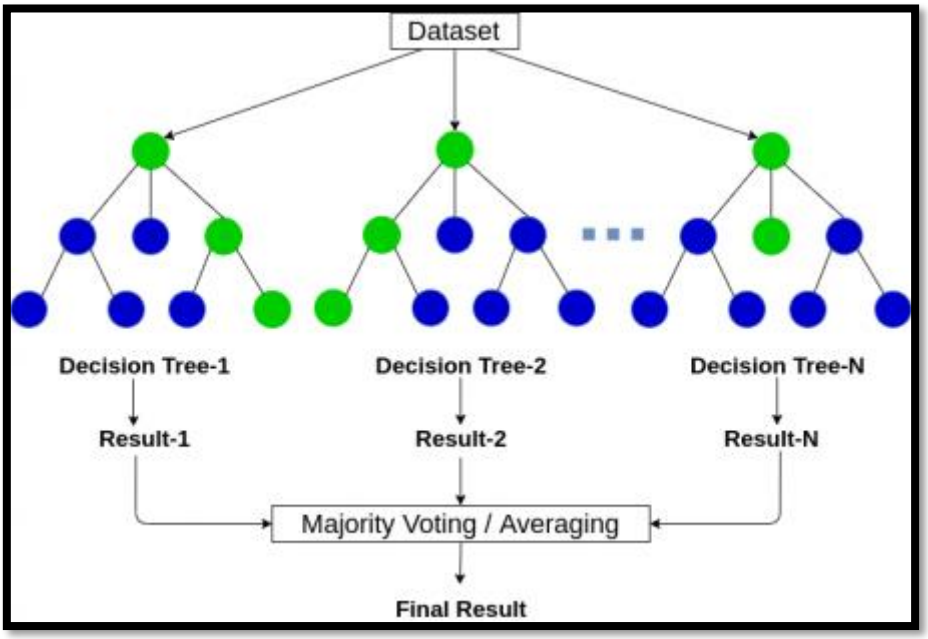


Figure 54 Random Forest flow (Anas Brital | Random Forest algorithm explained . n.d.)

For random forest as shown in Figure 54 the dataset is used to train several decision trees, with every tree making its own independent predictions. To make an output, the selection is to be made using majority voting (for classification) and average (for regression) for ensuring accuracy over single predictions and minimizing overfitting.

5.2.2 Support Vector Machine (SVM)

Support vector machine is a supervised machine learning method which is used for both classification and regression-based tasks (Miorelli *et al.* 2021). The aim of this algorithm is to find the optimal hyperplane separating data into distinct classes with maximum margin. Different types of kernels are

used based on the type of data. For linearly separable data set, linear kernel is used, for non-linear 'rbf' kernel can be applicable. SVM is effective for high-dimensional datasets and resistant to the overfitting. In this application SVM model with linear Kernel used and probability parameter kept as True which enables probability predictions (Huang, 2018). A complete structure of Support Vector Machine (SVM) algorithm is given as below (Figure 55).

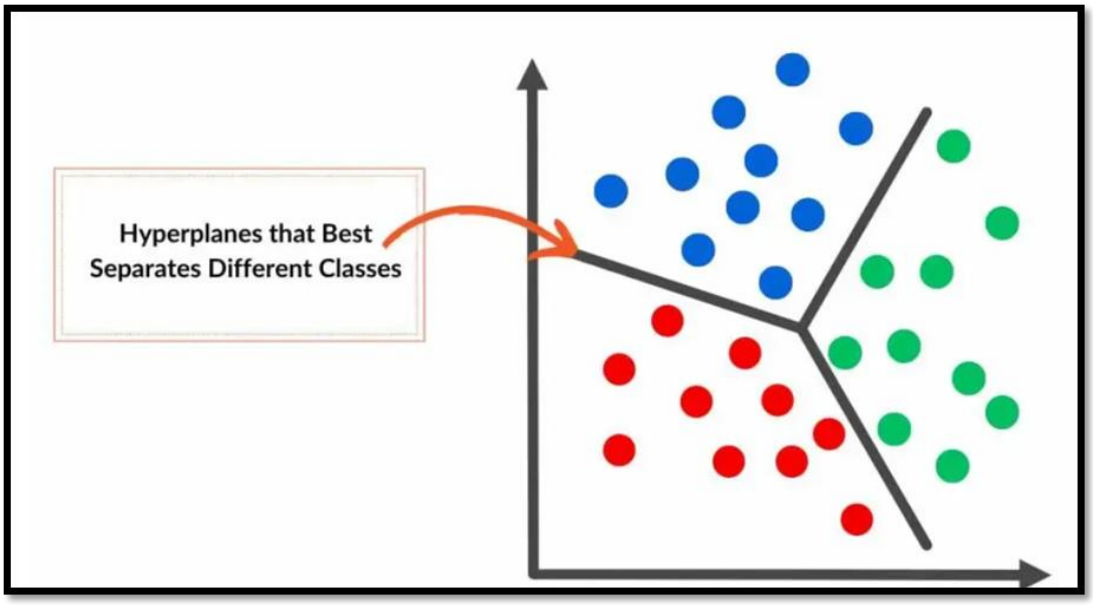


Figure 55 SVM flow (Alam 2024)

5.2.3 Logistic regression

Logistic regression is a linear model used for binary or multi-class classification purpose. This algorithm predicts the probabilities using the sigmoid function (Kibria & Matin 2022). Logistic regression is a simple, computationally efficient and interpretable which makes it suitable for linearly separable data. While this algorithm is not that much efficient for non-linear relationships. This algorithm often serves as a strong baseline for comparison in classification tasks. This applicable is use case of multi-class classification.

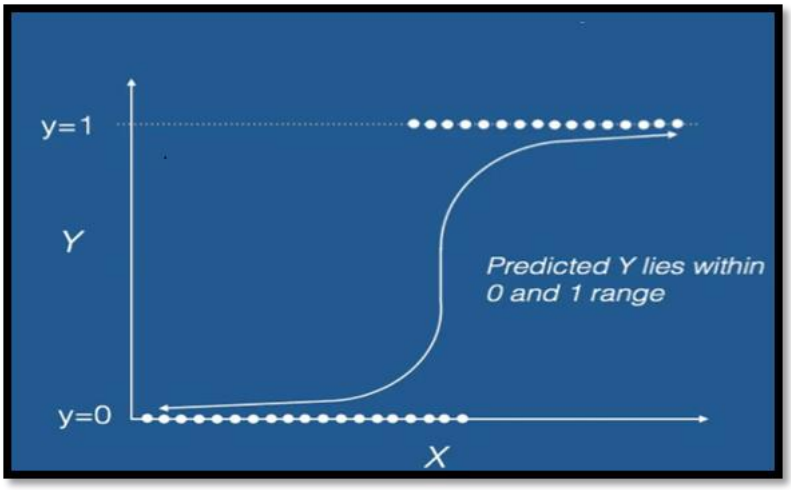


Figure 56 Logistic Regression flow (Rajput 2018)

Figure 56 describes the Logistic Regression, a classification algorithm that works in predicting a binary outcome condition. The S-shaped sigmoid curve maps input values(x) onto probabilities between 0&1 and hence determines the class label via thresholding, usually done at 0.5(Hess & Hess, 2019).

5.2.4 Artificial Neural Network (ANN)

ANN is a deep learning algorithm which is completely inspired by human brain. This model consists of interconnected layers of neurons (Chavlis & Poirazi, 2021). Each neuron applies weights and activation function. ANN works well with unstructured and large amount of data sets and also works well with non-linear relationships data sets. The only concern is they require more computational resources while training models. The web application we are working is multi-class classification problem. While training this model three layers are used input layer, hidden layer and output layer(Figure 57).

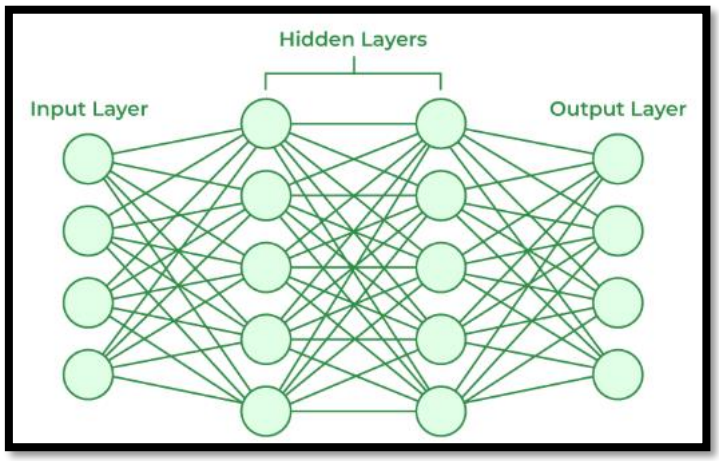


Figure 57 ANN algorithm structure (GeeksforGeeks 2024)

From figure 57 an artificial neural network consists of an input layer, one or more hidden layers, and an output layer. The input layer is responsible for doing the processing of initial features; consequently, hidden layers use functions like ReLU as activation functions to introduce non-linearity. Less often, dropout layers are also used, disabling random neurons during training epochs in order to prevent overfitting. The output layer applies an activation function to map classes into a final output probability, softmax for multi-class classifications and sigmoid for binary-class classifications. While compiling this model, Adam optimizer can be used to adjusts learning rates dynamically for efficient training.

5.3 Implementation

5.3.1 Artificial Neural Network (ANN)

```
# ANN model
model = Sequential()
# Input Layer
model.add(Dense(128, input_dim=X_train.shape[1], activation='relu'))
# Hidden Layers
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(32, activation='relu'))
# Output Layer
model.add(Dense(len(label_encoder.classes_), activation='softmax'))
# Compile and Train
model.compile(optimizer=Adam(), loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history = model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test), verbose=2)
```

Figure 58 ANN Model training (Self-created)

The code in Figure 58 implements an Artificial Neural Network (ANN) using TensorFlow and Keras. It consists of an input layer, three hidden layers with ReLU activation, and an output layer using softmax for classification.

5.3.2 Hyperparameter for ANN

It includes the follows:

- Input Layer: Has 128 neurons with ReLU activation, matching the feature dimensions of the input data.
- Hidden Layer 1: Includes 64 neurons with ReLU activation.
- Dropout (0.5): Introduces regularization by randomly dropping 50% of neurons during training to prevent overfitting.
- Hidden Layer 2: Includes 32 neurons with ReLU activation.
- Output Layer: Uses a softmax activation function to handle multi-class classification.

5.3.3 Random Forest

A robust and accurate model is created using ensemble learning by integrating several decision trees. Code fits training data to a RandomForestClassifier with 100 decision trees. collection the number of trees based on the data collection and processing resources.

```
# Random Forest model
random_forest_model = RandomForestClassifier(n_estimators=100, random_state=42)
random_forest_model.fit(X_train, y_train)
```

Figure 59 Random Forest Model training (Self-created)

5.3.4 Hyperparameter for Random Forest

- n_estimators=100: This defines the number of decision trees the Random Forest will build. A higher number improves model performance but increases computation time.
- random_state=42: Ensures reproducibility by controlling the randomness used during tree construction and sampling.

5.3.5 Support Vector Machine (SVM)

```
# SVM model
svm_model = SVC(kernel='linear', probability=True, random_state=42)
svm_model.fit(X_train, y_train)
```

Figure 60 SVM Model training (Self-created)

A powerful classification method that determines the hyperplane with the biggest margin between data points of different classes. Using training data, a linear SVC model is created. In this case, data

characteristics affect the kernel function. The code in Figure 60 creates and trains a Support Vector Machine (SVM) model on a linear kernel.

5.3.6 Hyperparameter for SVM

- kernel='linear': Specifies the kernel type used in the SVM algorithm. The linear kernel is suitable for data that is linearly separable.
- probability=True: Enables probability estimates for each class, useful for calculating metrics like AUC.
- random_state=42: Ensures consistent results by fixing the randomness in the training process.

5.3.7 Logistic Regression

```
# Logistic Regression model
log_model = LogisticRegression(max_iter=1000, random_state=42)
log_model.fit(X_train, y_train)
```

Figure 61 Logistic regression training (Self-created)

A widely used linear classification model. It calculates the likelihood of being a class member. The code builds and fits a LogisticRegression model to training data. Logistic regression is typically a good place to begin when classifying text because of its interpretability and efficiency. The code in Figure 61 initiates a Logistic Regression model with a maximum 1000 iterations for convergence and fix for reproducibility. It then proceeds to train the Logistic Regression model using the fit() method on the training data (X_train and y_train), thus fitting the parameters of the model to predict the target variable best.

5.3.8 Hyperparameter for Logistic Regression

- max_iter=1000: Sets the maximum number of iterations allowed for the solver to converge. A higher value is often required for complex datasets.
- random_state=42: Ensures reproducibility of the logistic regression training process.

5.4 Model Evaluation

After training, it is essential to evaluate model performance on unknown testing data and it will further help to determine best model for our problem.

```
# ANN evaluation
y_pred_ann_probs = model.predict(X_test)
y_pred_ann = np.argmax(y_pred_ann_probs, axis=1)
print("ANN Classification Report:")
print(classification_report(y_test, y_pred_ann, target_names=label_encoder.classes_))
print(f"ANN Accuracy: {accuracy_score(y_test, y_pred_ann):.4f}")

# Random Forest evaluation
random_forest_pred = random_forest_model.predict(X_test)
random_forest_probs = random_forest_model.predict_proba(X_test)[:, 1]  # For binary AUC calculation
print("Random Forest Classification Report:")
print(classification_report(y_test, random_forest_pred, target_names=label_encoder.classes_))
print(f"Random Forest Accuracy: {accuracy_score(y_test, random_forest_pred):.4f}")

# SVM evaluation
svm_pred = svm_model.predict(X_test)
svm_probs = svm_model.predict_proba(X_test)[:, 1]  # For binary AUC calculation
print("SVM Classification Report:")
print(classification_report(y_test, svm_pred, target_names=label_encoder.classes_))
print(f"SVM Accuracy: {accuracy_score(y_test, svm_pred):.4f}")

# Logistic Regression evaluation
log_pred = log_model.predict(X_test)
log_probs = log_model.predict_proba(X_test)[:, 1]  # For binary AUC calculation
print("Logistic Regression Classification Report:")
print(classification_report(y_test, log_pred, target_names=label_encoder.classes_))
print(f"Logistic Regression Accuracy: {accuracy_score(y_test, log_pred):.4f}")
```

Figure 62 Model Evaluation (Self-created)

As shown in the code in Figure 62 predictions regarding the testing data are made using the model.predict function. This predicts class labels for each text instance from the testing set. Classes and

accuracy are calculated using Sklearn.metrics' classification_report and accuracy_score functions. All machine learning models, such as ANN, Random Forest, SVM, as well as Logistic Regressionare evaluated on below performance metrics.

Precision: Precision measures the proportion of correctly predicted positive cases out of all predicted positive cases.

$$Precision = True\ Positives/(True\ Positives + False\ Positives)$$

Recall: Recall (Sensitivity) measures the proportion of correctly predicted positive cases out of all actual positive cases.

$$Recall = True\ Positives/(True\ Positives + False\ Negatives)$$

Accuracy: Accuracy is the ratio of correctly predicted cases to the total number of cases.

$$Accuracy = (True\ Positives + True\ Negatives)/Total\ Samples$$

F1-Score: The F1-score is the harmonic mean of precision and recall, balancing both metrics.

$$F1\ Score = 2 \times (Precision \times Recall)/(Precision + Recall)$$

5.5 Model Comparison

A comparison of model performance can be made by examining the evaluation findings from the previous phase. From code given in Figure 63, it is clear that to identify the model with the best accuracy on the testing data, a bar chart is used to display the accuracy of all the models.

```
# Accuracy comparison
model_names = ['ANN', 'Random Forest', 'SVM', 'Logistic Regression']
accuracies = [
    accuracy_score(y_test, y_pred_ann),
    accuracy_score(y_test, random_forest_pred),
    accuracy_score(y_test, svm_pred),
    accuracy_score(y_test, log_pred)
]

plt.figure(figsize=(10, 6))
plt.bar(model_names, accuracies, color=['blue', 'green', 'orange', 'red'])
plt.title("Accuracy Comparison of Models", fontsize=16)
plt.ylabel("Accuracy", fontsize=14)
plt.xlabel("Models", fontsize=14)
plt.show()
```

Figure 63 Model Comparison (Self-created)

5.6 Model Deployment

```
best_model = random_forest_model
pickle_filename = "best_model.pkl"
with open(pickle_filename, 'wb') as file:
    pickle.dump(best_model, file)

print(f"Best model saved as {pickle_filename}.")
```

Figure 64 Model deployment (Self-created)

The above code figure 64 shows that, selected best model is deployed as pickle file which will be deployed in the web application for the real-time threat detection.

5.7 Results and analysis

Class Name	Precision	Recall	F1-score
DOMAIN	1	0.75	0.86
EMAIL	0.8	1	0.89
FILEPATH	0.77	0.88	0.82
IPV4	0.85	1	0.92
Infrastucture	1	1	1
MD5	0	0	0
REGISTRYKEY	1	1	1
SHA1	1	0.69	0.82
SHA2	0.9	0.71	0.79
SOFTWARE	0.92	0.88	0.9
TIME	0.78	0.82	0.8
URL	1	0.95	0.97
attack-pattern	0.86	0.9	0.88
campaign	0.73	0.83	0.78
identity	0.88	0.89	0.88
location	0.85	0.85	0.85
malware	0.94	0.9	0.92
threat-actor	0.86	0.93	0.89
tools	1	0.91	0.95
vulnerability	0.83	0.88	0.85
accuracy	0.88		
macro avg	0.85	0.84	0.84
weighted avg	0.88	0.88	0.88

Figure 65 Performance metrics table for ANN (Self-created)

The table shown in Figure 65 presents the evaluation metrics (Precision, Recall, F1-score) for an Artificial Neural Network (ANN) model across various class labels in a cybersecurity context. It highlights the model's performance, with overall accuracy of 0.88, macro average F1-score of 0.84, and weighed average F1-score of 0.88 across 1893 instances.

Class Name	Precision	Recall	F1-score
DOMAIN	1	0.75	0.86
EMAIL	1	1	1
FILEPATH	1	0.8	0.89
IPV4	1	1	1
Infrastucture	1	0.75	0.86
MD5	1	0.5	0.67
REGISTRYKEY	1	1	1
SHA1	0.69	0.69	0.69
SHA2	0.64	0.79	0.71
SOFTWARE	0.9	0.9	0.9
TIME	0.98	0.84	0.9
URL	1	0.95	0.97
attack-pattern	0.89	0.88	0.89
campaign	0.71	0.87	0.78
identity	0.91	0.89	0.9
location	0.89	0.87	0.88
malware	0.9	0.94	0.92
threat-actor	0.87	0.96	0.91
tools	0.97	0.96	0.97
vulnerability	0.81	0.91	0.86
accuracy	0.9		
macro avg	0.91	0.86	0.88
weighted avg	0.9	0.9	0.9

Figure 66 Performance metrics table for Random Forest (Self-created)

The table In Figure 66 displays the evaluation metrics (Precision, Recall, F1-score) for a Random Forest model applied to a multi-class classification problem. The model achieves an overall accuracy of 0.90, with macro-average F1-score of 0.88 and weighted-average F1-score of 0.90, indicating strong performance across most classes.

Class Name	Precision	Recall	F1-score
DOMAIN	1	0.33	0.5
EMAIL	1	1	1
FILEPATH	0.79	0.67	0.72
IPV4	0.88	0.64	0.74
Infrastucture	1	0.12	0.22
MD5	0	0	0
REGISTRYKEY	0.67	1	0.8
SHA1	1	0.46	0.63
SHA2	0.81	0.45	0.58
SOFTWARE	0.76	0.85	0.8
TIME	0.84	0.72	0.78
URL	0.91	0.5	0.65
attack-pattern	0.72	0.77	0.74
campaign	0.71	0.74	0.72
identity	0.83	0.78	0.8
location	0.68	0.74	0.71
malware	0.78	0.87	0.82
threat-actor	0.84	0.82	0.83
tools	0.86	0.64	0.73
vulnerability	0.67	0.88	0.76
accuracy	0.77		
macro avg	0.79	0.65	0.68
weighted avg	0.78	0.77	0.77

Figure 67 Performance metrics table for SVM (Self-created)

The table In Figure 67 evaluates the SVM model's performance across multiple classes, showing an overall accuracy of 0.77 with a macro-average F1-score of 0.68. While some classes, like "EMAIL" and "SOFTWARE," perform well, others like "DOMAIN" and "Infrastructure" show lower recall, indicating room for improvement.

Class Name	Precision	Recall	F1-score
DOMAIN	0	0	0
EMAIL	1	0.5	0.67
FILEPATH	0.76	0.44	0.56
IPV4	1	0.18	0.31
Infrastucture	0	0	0
MD5	0	0	0
REGISTRYKEY	1	0.25	0.4
SHA1	0	0	0
SHA2	0.5	0.05	0.1
SOFTWARE	0.67	0.81	0.73
TIME	0.89	0.48	0.63
URL	0.86	0.3	0.44
attack-pattern	0.64	0.74	0.68
campaign	1	0.13	0.23
identity	0.76	0.74	0.75
location	0.58	0.68	0.62
malware	0.65	0.87	0.75
threat-actor	0.79	0.75	0.77
tools	0.87	0.34	0.49
vulnerability	0.53	0.52	0.52
accuracy	0.68		
macro avg	0.62	0.39	0.43
weighted avg	0.69	0.68	0.66

Figure 68 Performance metrics table for Logistic Regression (Self-created)

The table In Figure 68 demonstrates an overall accuracy of 0.68 with a weighted average F1-score of 0.66. However, it performs poorly in classes like "DOMAIN" and "Infrastructure" (F1-score of 0), while achieving relatively better performance in "SOFTWARE" and "malware" detection.

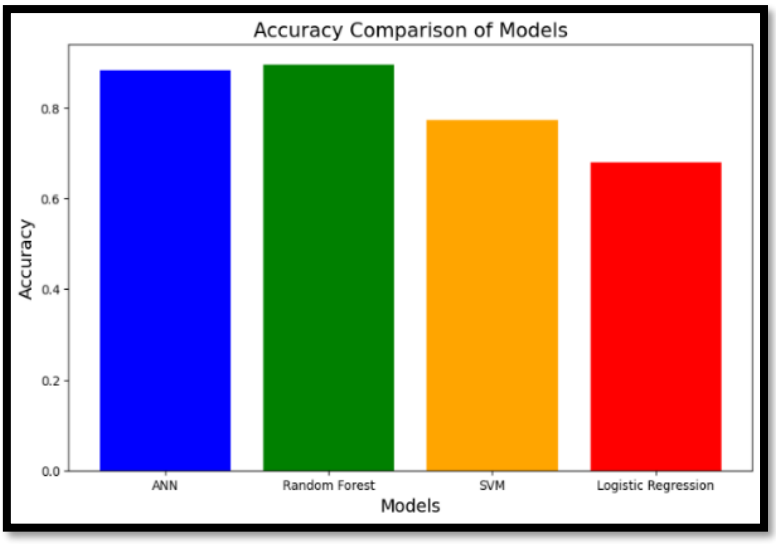


Figure 69 Accuracy Comparison (Self-created)

From bar chart in figure 69 and the results show that the Random Forest model performed definitively better than all other algorithms, achieving an accuracy of 89.59%. This performance surpassed that of the ANN model by a narrow margin as it achieved 88.22%. In contrast, the Support Vector Machine and Logistic Regression models exhibited low performances, attaining 77.23% and 67.99% accuracy, respectively. Based on such findings, the Random Forest model was chosen as the most suitable for deployment purposes; this was then exported as a pickle file for integration into the web application for real-time threat detection, thereby guaranteeing a reliable and accurate result.

```
Best model saved as best_model.pkl.
```

Figure 70 Model Deployment (Self-created)

The Random Forest model outperforms than other models so trained Random Forest (Figure 70) is deployed as pickle file, and it is applied in the web application for real-time threat detection.

6 Network Security Policies

6.1 Introduction

As increasing number of sophisticated cyber threats resulting in the critical need of having strong network and security policies is vital to protecting the infrastructure. Network security policies are sets of rules, configurations, and security controls that safeguard networks, systems, and data from unauthorized access, breaches, and cyberattacks. Policing these translates to a blueprint of secure network operations to the level of integrity, confidentiality, and availability of the involved data (Karthikeyan and Thenmozhi, 2024). Data security in AI-powered threat detection would require a network-level security policy to ensure that sensitive datasets are secured, cloud-hosted applications are well protected, and malicious intrusions are prevented. Adopting Cloud solutions like AWS, Cloud platforms offer advanced security mechanisms such as Virtual Private Clouds (VPCs), Identity and Access Management (IAM), built-in firewalls, encryption, and automated backup solutions that enhance overall security resilience (Karthikeyan and Thenmozhi, 2024).

A multi-layered security strategy helps the organization reduce risks, support least-privilege access, and meet industry compliance. The corresponding AWS-based security policy implemented, comprises networking VPC as per multi-tenancy and segmentation, EC2 as a secure computing environment, Amazon S3 for encrypted storage, IAM for access management, and AWS Backup for disaster recovery, which constructs a secure, elastic infrastructure base for AI-driven threat detection.

6.2 Virtual Private Cloud (VPC)

A Virtual Private Cloud (VPC) is a network that is logically isolated from other networks running in a public cloud service provider's infrastructure later. Organizations can specify the specific details of their network environment within VPC, including the subnets in use, the routing settings, these security settings, and more. AWS VPC (Amazon Web Services Virtual Private Cloud) is a virtual network that limits access to cloud resources (Sahoo et al., 2024). A VPC allows their designers to create a dedicated virtual network similar to an on-premises data center; however, they can downsize or expand instantly, making VPCs more cost-effective and dependable than standard data centers.

6.2.1 Key Features of AWS VPC Implementation

6.2.1.1 Network Isolation and Customization

- AWS VPC empowers organizations to isolate and segment workloads with custom IP address ranges, subnets, and routing tables (Sahoo et al., 2024).
- This way no unauthorized access is possible because communication between the subnets is restricted according to a security policy.
- Public and private subnets also assist in keeping critical resources such as databases and backend servers separate from the internet, reducing the attack point.

6.2.1.2 Enhanced Security with Security Groups and Network ACLs

- Security Groups (SGs) is virtual firewalls that filter traffic coming in and out of your instance.
- Network Access Control Lists (NACLs) add an extra layer of security at the subnet level by defining allow/deny rules for incoming and outgoing traffic.
- These controls are designed to implement a zero-trust security model, where we know exactly who accesses the cloud environment.

6.2.1.3 Elastic IP for Persistent Connectivity

- AWS Elastic IPs (EIPs) maintain a single public IP address for cloud instances even if the instances are restarted or swapped out.
- This is especially critical for hosting web applications, APIs, and AI-based threat detection systems because they rely on constant interaction with the users and other systems (Sahoo et al., 2024).

6.2.1.4 Secure Communication via VPN and Direct Connect

- Organizations can create a Virtual Private Network (VPN) that further connects private on-premises infrastructure with AWS VPC and employs encryption in transit (Sahoo et al., 2024).
- AWS Direct Connect is a dedicated network connection to AWS that reduces latency and increases security for sensitive workloads.

6.2.1.5 Traffic Monitoring and Logging with AWS VPC Flow Logs

- VPC Flow Logs provide granularity into authenticating virtual networks which can help security teams to observe network activities, detect abnormal data and investigate potential threats.
- This is especially useful for Threat detection powered by AI, as machine learning models can learn from traffic systems to detect anomalies/potential bad behavior in real-time.

6.2.1.6 Why VPC is Implemented in This Project

AWS VPC has been used in the project of AI-Powered Threat Detection for Networks to give a secure and isolated cloud environment for deploying AI models and web applications. The key reasons for implementing VPC include:

- Network Segmentation: With VPC, we can separate AI processing, web application and database layer to restrict security risks.
- Controlled access: With the help of security groups and IAM roles, they block unwanted access and reduce the chances of any cyberattacks.
- Security from Threats: The VPC allows the detection and mitigation of security incidents through built-in logging and monitoring.
- Scalability and Flexibility: The architecture of the VPC is designed to dynamically scale AI models and web applications, ensuring real-time threat detection capabilities (Mane and Ainapure, 2021).

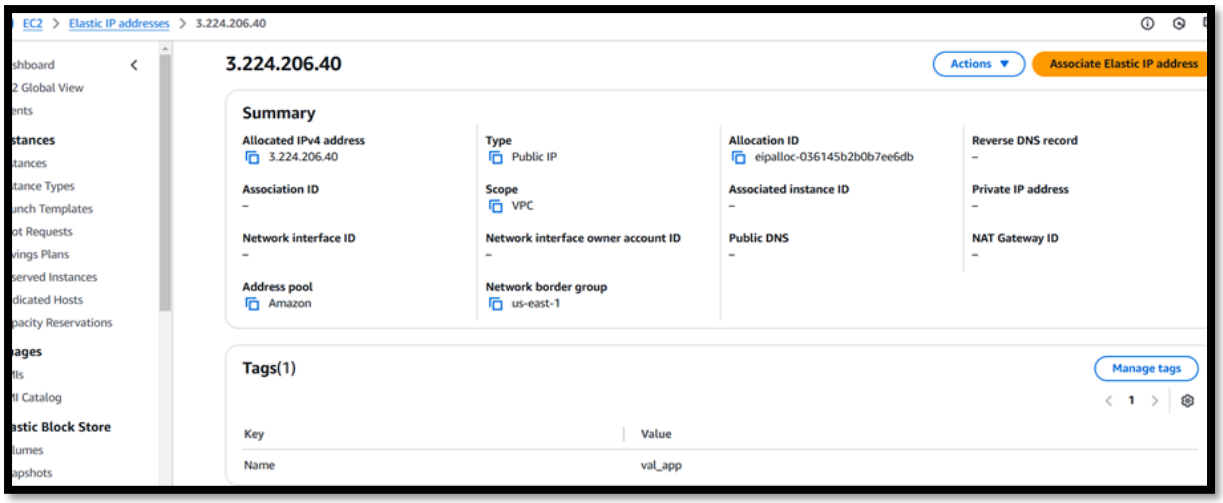


Figure 71 Summary of AWS Elastic IP address (Self-created)

The above image (Figure 71) shows the AWS Elastic IP address with summary of IP address including the type of IP address, allocation id, scope, address pool, and network of elastic IP address. This elastic IP address we can assign to ec2 instance which we will create.

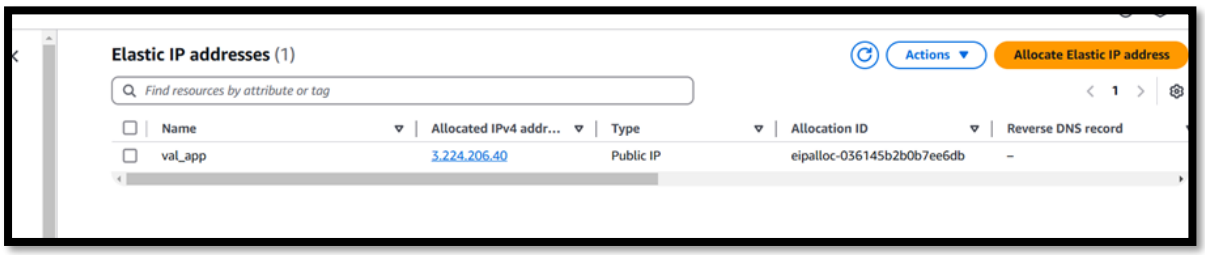


Figure 72 AWS Elastic IP address (Self-created)

The above image (Figure 72) shows the AWS Elastic IP address with the name of val_app. We can assign the elastic IP to ec2 instance.

6.3 EC2 Instances

Amazon EC2 (Elastic Compute Cloud) is a flexible cloud computing service allowing users to run virtual machines in the cloud, providing scalable compute capacity in the cloud for hosting applications, running services, and executing workloads (Niranjanamurthy et al., 2020). EC2 instances are the foundational building blocks of cloud infrastructure and can be launched with various instance types, storage options and security settings. With respect to AI-powered threat detection for networks, EC2 instances serve as the web application and backend AI models hosts, allowing for the processing, analysis, and response to cyber threats.

6.3.1 Role of EC2 Instances in AI-Powered Threat Detection

6.3.1.1 Hosting the Web Application and AI Backend

- The EC2 instances used in this project have been configured to provide hosting for the company forum and AI-based threat detection.
- The frontend web app is hosted on EC2, enabling employees to communicate in a secure environment.
- Text-based messages are instantly analyzed, detects, and pertinent actions are taken by the backend AI model operating on EC2 (Niranjanamurthy et al., 2020).

6.3.1.2 Optimized Compute Power for AI and ML Models

- The EC2 instance type is selected as t2.small, which has enough compute resources to get started with inference, real-time text analysis, and security monitoring.
- For heavier ML workloads, various EC2 instances optimized for true GPU performance, such as p3 or g4 instances, can be used to facilitate training and running of AI models.
- CPU and Memory resources are provisioned for this instance optimized for processing large volumes of text data and network log activity.

6.3.1.3 Security and Access Control

- The EC2 instance is secured with AWS Security Group which restricts the access of users and services to authorized ones.
- This prevents unauthenticated remote logins, SSH is limited only to certain IP addresses.
- IAM roles are attached so that the instance can interact securely with Amazon S3/AWS Backup and other services in the cloud.

6.3.1.4 Elastic IP and Persistent Connection

- An Elastic IP is assigned to the EC2 instance, allowing a consistent public IP address that enables the web application with AI detection system to be accessed without interruption.
- Therefore, this setup is crucial to ensure secure and reliable communication between employees with the help of AI model and also the system administrator monitoring threats.

6.3.1.5 Monitoring and Auto Scaling for Performance and Reliability

- AWS CloudWatch monitors the CPU, memory, and network usage of the EC2 instance.
- As the work load increases, we can set up Auto Scaling Groups (ASG) to automatically launch more EC2 instances to accommodate more traffic and threat analysis (Niranjanamurthy et al., 2020).
- This will keep the system responsive and prevent any downtimes due to high demand in computational power.

6.3.1.6 Data Backup and Disaster Recovery

- AWS Backup policies are configured to automated snapshots of the EC2 instance so the system can be restored when recovery is needed after a system failure or cyber-attack.
- This allows the system to be restored quickly without losing valuable log or AI model data.

6.3.2 Why EC2 is used in this project

The use of EC2 instances is essential for the AI-powered threat detection system offered in this project because they provide a flexible, scalable, and secure environment for the developed application. The biggest reasons to go with EC2 are as follows:

- On-demand compute powers - EC2 process allows AI workloads in a scalable and cost-effective manner.

- Security & Access Control - AWS comes equipped with robust security utility, firewalls IAM policies, and encryption (Niranjanamurthy et al., 2020).
- Data integrity - Regular backups protect the organization from data loss in the event of a disaster.
- Scalability - The system can be scaled up or down according to real-time business threats.

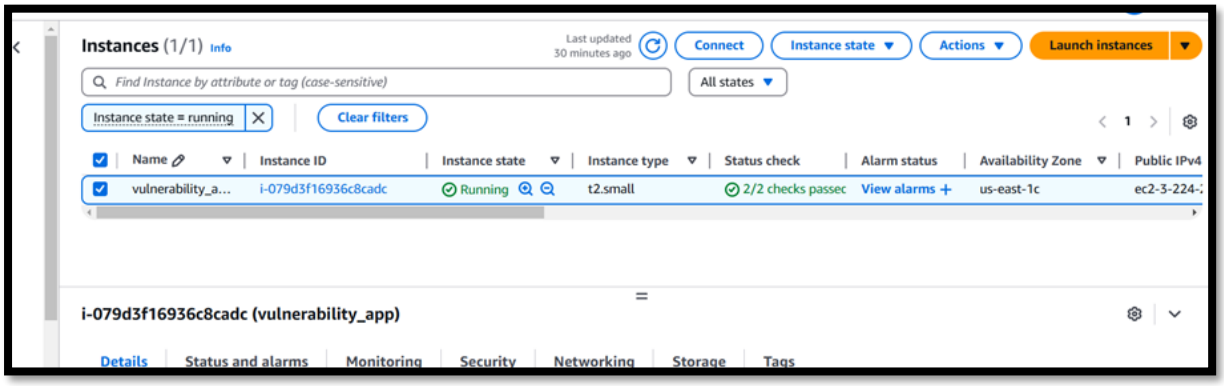


Figure 73 AWS EC2 instance management console (Self-created)

The above image (Figure 73) shows the AWS EC2 instance management console. It highlights a single running instance (i-079d3f16936c8cadc) named vulnerability_app, with the instance type t2.small. The instance is located in the us-east-1c availability zone, has passed its 2/2 status checks, and is associated with a public IPv4 address (3.224.206.40). Additional details are accessible under tabs like monitoring, security, and networking.

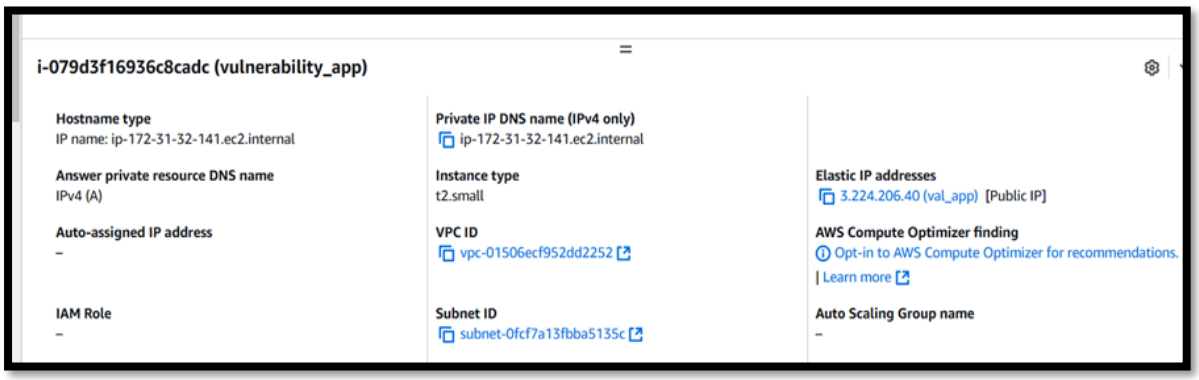


Figure 74 AWS EC2 instance for vulnerability_app (Self-created)

The above image (Figure 74) provides details about an AWS EC2 instance with ID i-079d3f16936c8cadc, labeled as vulnerability_app. It shows key attributes such as the instance type (t2.small), private IP DNS name, public elastic IP address (3.224.206.40), VPC ID, and Subnet ID. Additionally, it mentions options for AWS Compute Optimizer findings and lacks an associated IAM Role or Auto Scaling Group.

6.4 Amazon S3 for Secure Data Storage

Amazon S3 stands for Amazon Simple Storage Service, allowing customers to securely store and retrieve any amount of data, from anywhere on the web. In the application of AI-enabled network threat detection, Amazon S3 acts as the repository for the datasets, AI models, logs, and backup files, ensuring that all data is securely stored and can be accessed when needed (Bundela, Dhanda and Gupta, 2024).

6.4.1 Role of Amazon S3 in AI-Powered Threat Detection

6.4.1.1 Storage of Datasets for AI Model Training

Amazon S3 buckets securely store the Kaggle dataset used to develop and validate AI models (Bundela, Dhanda and Gupta, 2024).

AI models can consume massive amounts of text-based cyber threat detection data, such as network logs, textual messages, and labeled threat classifications.

Amazon S3 enables efficient data retrieval, allowing the AI system to learn continuously and improve the accuracy of threat detection.

6.4.1.2 Secure Storage for Model Files

Trained AI models for real-time threat detection are kept in Amazon S3, allowing version control and easy deployment.

One final benefit of storing model files in S3 is that they are they are protected, backed up, and accessible for future updates and improvements.

This helps ensure that the AI models can be available and resilient even in the face of an outage by providing a multi-region redundancy (Bundela, Dhanda and Gupta, 2024).

6.4.1.3 Logging and Incident Response Data

Logs generated by the threat detection system are those for anomalous user activities, flagged messages and detected security threats.

These logs are placed on Amazon S3 to analyze inspects for forensic analysis, performing compliance audits, and having better decisions made from AIs.

By combining Amazon Athena and AWS Glue with S3, security logs can be queried and analyzed at scale.

6.4.1.4 Automated Backup & Disaster Recovery

The Amazon S3 is configured with automated backups to avoid data loss and to ensure business continuity (Bundela, Dhanda and Gupta, 2024).

We apply the Versioning and Lifecycle Management policies to store important data and reduce storage costs.

Stored data can be promptly restored from backup snapshots in case of a cyberattack or accidental deletion, minimizing disruption.

6.4.1.5 Access Control and Security Policies

To enforce, the policies are applied at bucket level (S3 Bucket Policies) and at individual access levels IAM Roles (IAM Users) to limit unauthorized access to ensure proper access and users and services can read/write to the stored data.

Databases are encrypted at rest (using SSE-S3, SSE-KMS, SSE-C) and SSL/TLS encryption protects data in motion.

AWS Identity and Access Management (IAM) policies ensure that only select AI models, applications, and administrators have access to critical files.

6.4.1.6 Integration with AWS Services for Enhanced Security

AWS Macie is used on the sensitive data which is inside the S3, it detects security risks such as data leaks or sensitive data exposure.

Amazon GuardDuty analyzes S3 access patterns to identify anomalies, unauthorized access attempts or cyber threats.

AWS CloudTrail entries also log all requests to access an S3 bucket; therefore, have audit data for compliance and security audit.

6.4.2 Why Amazon S3 is Used in This Project

- ✓ Amazon S3 is selected as the primary storage solution which is secure, durable and integrates seamlessly with AI and security tools. The key advantages include:
- ✓ Scalability: S3 stores an unlimited amount of structured and unstructured data, ideal for AI and cybersecurity applications.
- ✓ High Durability and Availability: 99.999999999% (11 nines) of data durability, effectively eliminating the risk of data loss.
- ✓ Security and Compliance: Offers end-to-end encryption, access controls, and adherence to GDPR, HIPAA, and ISO standards.
- ✓ Cost-Efficiency: Whether with its standard, infrequent access or Glacier, AWS gives businesses the flexibility to lower costs with tiered access.
- ✓ Fluid Integration: Facilitates the automation of downstream data workflows, as well as AI model deployment with AWS Lambda, EC2, CloudWatch, and ML services.

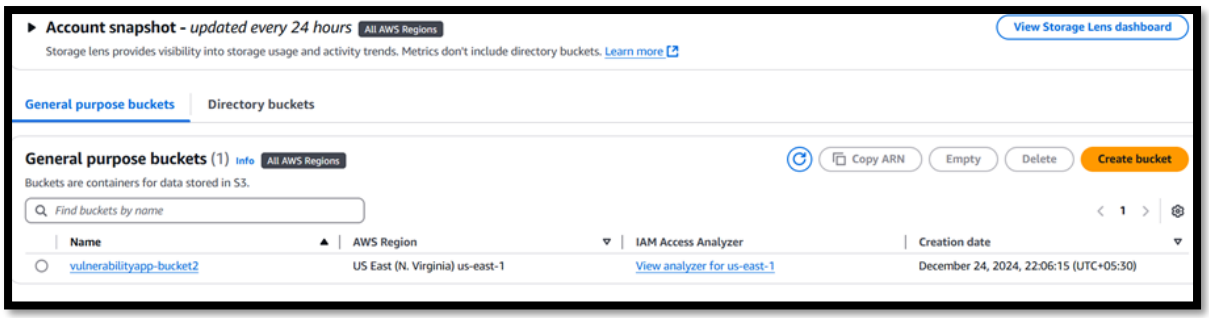


Figure 75 Amazon S3 dashboard (Self-created)

The above image (Figure 75) displays the Amazon S3 dashboard showing a bucket named `vulnerabilityapp-bucket2` in the `us-east-1` (US East, N. Virginia) region. The bucket's creation date is December 24, 2024, at 22:06:15 . Options like "Copy ARN," "Empty," "Delete," and "Create bucket" are available for managing buckets.

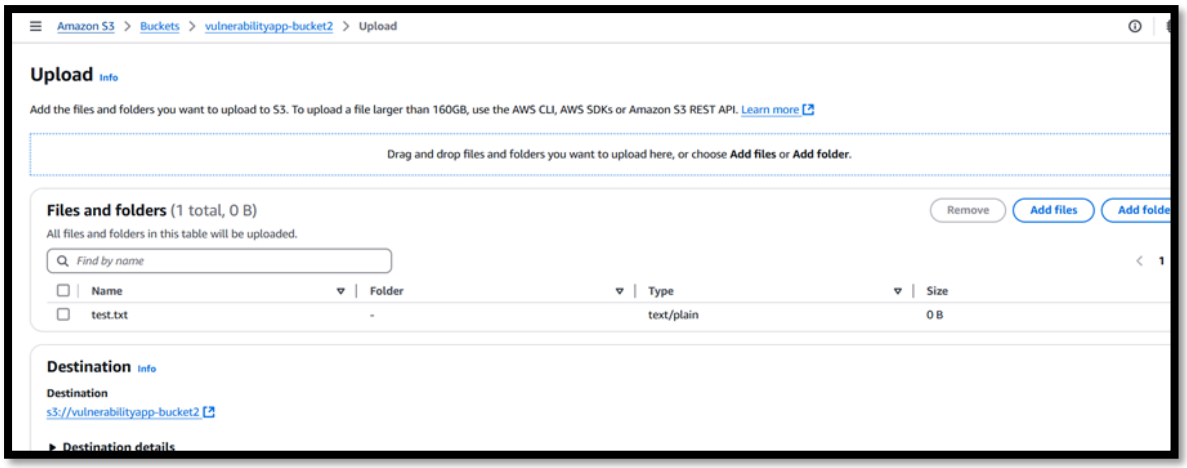


Figure 76 Amazon S3 upload interface for adding files (Self-created)

The above image (Figure 76) shows the Amazon S3 upload interface for adding files to the `vulnerabilityapp-bucket2`. A file named `test.txt` (0 bytes, plain text) is selected for upload. Users can

drag and drop files, add folders, or manage selections using the provided buttons. The destination for the upload is clearly displayed.

6.5 AWS Backup for Disaster Recovery

As the world becomes increasingly cyber-aware, organizations are implementing a variety of security solutions to combat potential threats, none more so than the revolutionary new AI-powered threat detection systems. AWS Backup offers a central, policy-driven, automated way to manage backups of AWS resources such as Amazon S3, EC2 instances, databases, and other services, and is essential for disaster recovery. The use of AWS Backup in this project enables a comprehensive disaster recovery strategy so that important data, AI models and security logs are safe from accidental deletions, cyberattacks and hardware failures (Bayazitov et al., 2024).

One of the key benefits including automation is one of the key benefits of AWS Backup. This project implements a backup plan to run at predefined intervals to store all AI models, threat detection logs, datasets in a secure storage. This automation reduces the chances of human error and ensures that the most up-to-date versions of important files are always available for recovery. AWS Backup has a centralized dashboard that enables security teams to monitor backup status, configure policies and restore data in case of an incident seamlessly (Bayazitov et al., 2024).

Multi-region backups support via AWS Backup Requirements in AWS Backup. This means that in the event of a cyberattack or natural disaster compromising one AWS data center, backups can be recovered from another location. This improves the resilience of the system and reduces impact time, allowing operability of AI-based threat detection with significant disruptions.

Retention policies and lifecycle management are also key features for AWS Backup. On this project, using backup rules that retain multiple copies of the AI models, logs, and configurations for a certain amount of time, while older backups are ages into low-cost storage solutions, cohort storage management, such as Amazon S3 Glacier (Bayazitov et al., 2024). This helps keep costs low for storage without sacrificing regulatory compliance around retention and auditing of data.

Security AWS Backup is heavily focused on security. AWS Key Management Service (AWS KMS) is leveraged to enforce end-to-end encryption in transit and at rest for backups. IAM roles and access policies guide that only those persons can modify or restore backups, which minimizes the possibility of unauthorized access or data tampering. These include integration with AWS CloudTrail, which logs backup activity, enabling an administrator to see who changed or deleted a backup in the same AWS account, and detect potential security incidents.

Point-in-time recovery (PITR) for databases is another must-have feature of AWS Backup, as it provides data related to network security policies, AI threat analysis, and identified incidents with the ability to be restored to a previous state when corruption or ransomware attacks occur (Sohail and Tabet, 2023). Such a feature is important in cybersecurity applications to keep the data integrity through rapidly recovering to the last known correct state, thereby minimizing downtime and mitigating loss of critical data.

By incorporating AWS Backup, this project benefits from a highly-available, resilient, and secure storage layer for the AI-based threat-detection system, which acts as an essential piece of the disaster recovery plan for the project. So, using multi-region storage, encryption, automated retention policies, and real-time monitoring, AWS Backup builds resilience, reduces downtime, and secures mission-critical cybersecurity data from accidental and malicious loss. Along with integration with other AWS security tools to make it more robust, AWS Backup provides effectiveness and reliability in disaster recovery strategy for the cloud-based AI powered Security systems (Sohail and Tabet, 2023).

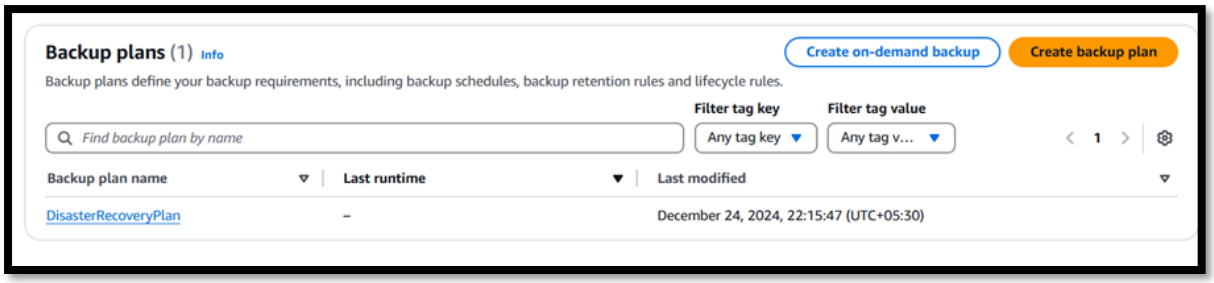


Figure 77 Backup plans section in AWS Backup (Self-created)

The above image (Figure 77) displays the "Backup plans" section in AWS Backup, showing one backup plan named "DisasterRecoveryPlan." This plan includes details such as the last modified date and time (December 24, 2024, 22:15:47 and no recorded last runtime. Options to create an on-demand backup or a new backup plan are also available.

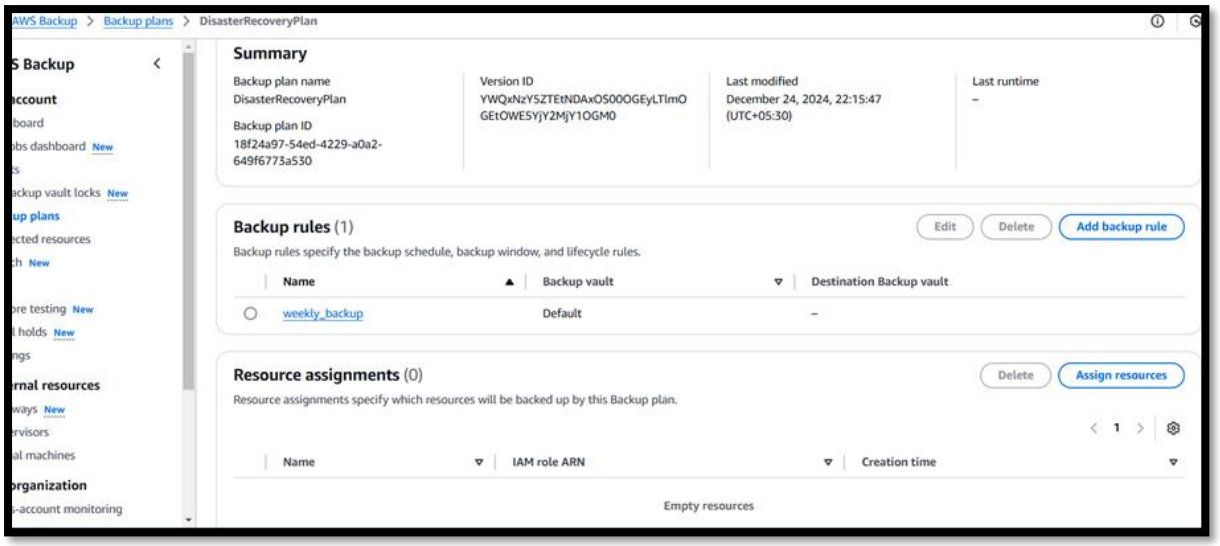


Figure 78 AWS backup plan named as DisasterRecoveryPlan (Self-created)

The above image (figure 78) shows the details of a backup plan named "DisasterRecoveryPlan" in AWS Backup. It includes a summary with the backup plan ID, version ID, and last modified time. The plan has one backup rule, "weekly_backup," stored in the default backup vault, but no resource assignments are currently configured.

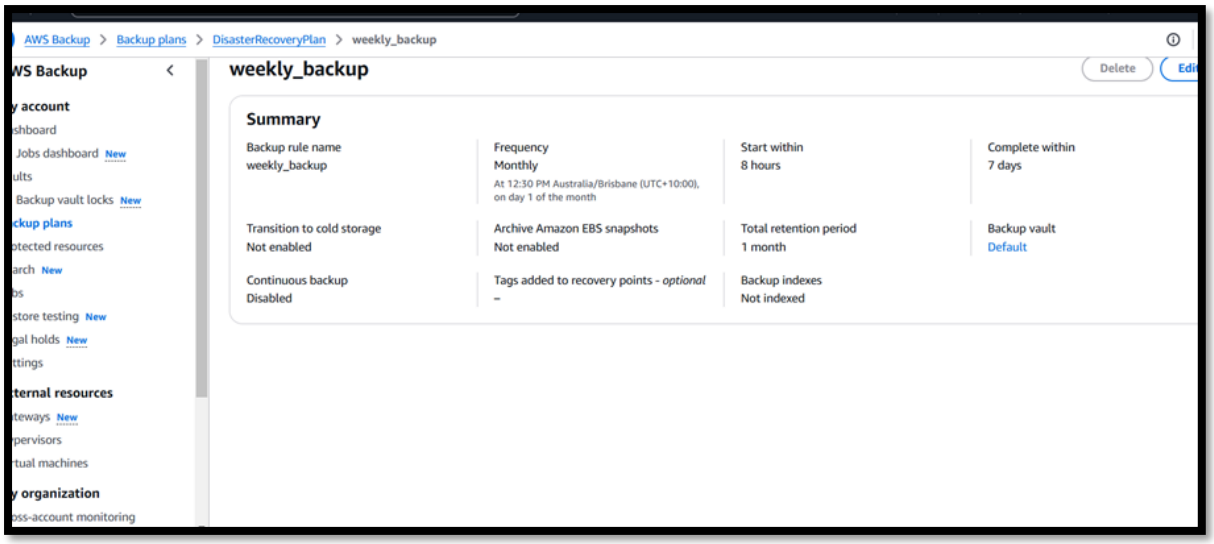


Figure 79 Backup plan named weekly_backup in the AWS Backup service(Self-created)

The above image (Figure 79) displays the details of a backup plan named weekly_backup in the AWS Backup service. The plan has a monthly frequency, starting at 12:30 PM Brisbane time on the first day of the month, with a retention period of one month. Features like transitioning to cold storage, continuous backup, and EBS snapshot archiving are not enabled. The backups are stored in the Default backup vault and are not indexed.

7 Risk Assessment

7.1 Introduction

This Risk Assessment Document provides an overview of the potential risks involved in the implementation of the AI Threat Detection system within a network environment. It assesses potential security risks, operational difficulties, and mitigation methods to maintain strong network defense. These risks include Data breaches, system failures, compliance risks, bias in AI models, and misconfigurations in network security infrastructure.

7.2 Risk Identification and Assessment

Risk Category	Risk Description	Likelihood	Impact	Mitigation Strategy
Data Breach	Unauthorized access to sensitive threat detection logs and employee communication data (Aboukadri, Ouaddah and Mezrioui, 2024).	High	High	Implement end-to-end encryption, IAM role-based access control, and multi-factor authentication (MFA), Regular security audits (Aboukadri, Ouaddah and Mezrioui, 2024).
AI Model Bias	AI system may produce false positives or negatives in threat detection due to biased training data (Maddireddy, 2022).	Medium	High	Use diverse datasets, perform continuous model training, and validate against unbiased test cases. Implement human-in-the-loop monitoring (Maddireddy, 2022).
Cloud Misconfiguration	Incorrect AWS security settings (e.g., open S3 buckets, weak IAM policies) leading to data exposure.	High	High	Follow AWS best security practices, automate security policy enforcement using AWS Config, and conduct periodic vulnerability assessments (Akinbolaji, 2024).
Downtime or Service Disruption	Cloud infrastructure failure affecting real-time threat detection services.	Medium	High	Use AWS Auto Scaling, Multi-Region Deployment, and Disaster Recovery (AWS Backup) for high availability (Akinbolaji, 2024).

Compliance and Legal Risks	Failure to comply with GDPR, CCPA, or other regulations related to data privacy.	Medium	High	Implement data retention policies, audit logging, and compliance checks. Regular security reviews (Prasad, 2024).
Cyberattacks (DDoS, Phishing, Ransomware)	Targeted attacks disrupting AI model functionality or data integrity (Chakraborty et. al., 2023).	High	High	Deploy AWS WAF, use DDoS protection via AWS Shield, and implement automated security response mechanisms (Chakraborty et. al., 2023).
Insider Threats	Employees misusing access privileges to modify or leak threat detection data.	Low	High	Enforce role-based access control (RBAC), monitor logs using AWS CloudTrail, and apply least privilege principle (Chakraborty et. al., 2023).

7.3 Risk Mitigation Plan

The Risk Mitigation Plan is as follows:

- **Security Hardening:** All the AWS services including VPC, EC2, S3, IAM and AWS Backup to be strictly adhered to security.
- **AI Model Validation:** Continuously test and improve threat detection algorithms to mitigate bias and reduce false positives.
- **Data Protection:** Use AWS IAM and KMS encryption to encrypt sensitive logs and implement stringent access control policies (Chakraborty et. al., 2023).
- **Incident response plan:** Evaluate risk acceptance and write a real-time incident response plan with security teams with AWS CloudWatch and AWS Security Hub alerts
- **Conduct Regular Security Audits:** Periodic penetration testing, compliance check, and cloud security assessment
- **User Awareness Training:** Educate all employees on the best practices for cybersecurity, such as preventing phishing attacks and mitigating insider threats.

8 Recommended Security Controls

In today's fast-changing cyber security scene, ensuring the security, dependability, and resilience of AI-powered online apps is crucial. AI-driven threat detection systems require network security, threat protection, threat monitoring, incident response, and compliance governance. The system uses AWS access control, tight threat security, and machine learning-based anomaly detection of threats.

8.1 Network Security Policies and Access Control

A strong network threat security policy is the main cyberdefense. Critical services are isolated and IAM roles control user access in VPC segmentation. RBAC ensures least privilege access for admins, employees, and users. Admin and staff accounts should be enforced MFA for security. Set up Security Groups and ACLs to restrict access(Amazon Web Services (AWS), 2025a). By ensuring that only allowed users can access vital data and functionality, these processes improve system security.

8.2 Data Protection and Encryption

Web applications with cyberthreat data must maintain threat security. AWS KMS end-to-end encryption avoids data leakage and unauthorised access. Amazon S3 storage should be protected with Server-Side Encryption (SSE), ensuring encrypted threat logs and sensitive files. TLS/SSL encryption should be enforced for all network communications, ensuring user data confidentiality. Masking and tokenising sensitive user data improves security and reduces breach risk(Amazon Web Services (AWS), 2025b). These encryption technologies ensure data confidentiality and integrity, while ensuring industry security.

8.3 Threat Detection and Real-Time Monitoring

Cyber threats require real-time monitoring and AI-driven threat identification. AWS GuardDuty should monitor AWS accounts for illegal access and unusual network activity. WAF prevents SQL injection, XSS, and DDoS threats. Random Forest, SVM, and Logistic Regression ML models detect suspicious activity and risky content in intelligent threat analysis. NLP can identify dangerous or malicious user-generated text messages on the platform before they reach end-users(Tirtakusuma et al., 2024).Security teams can quickly notice, respond to, and fix security issues with event logging and SIEM tools like AWS CloudTrail and Amazon Security Hub.

8.4 Incident Response and Automated Threat Mitigation

A systemic incident response plan ensures that security threats are swiftly identified, contained, and fixed. AWS Lambda functions can remediate cyber threats by automatically triggering security policies and blocking malicious users. Security patching is automated by AWS Systems Manager, ensuring that vulnerabilities are patched before attackers can exploit them. AWS Backup must be utilised to construct a disaster recovery and backup strategy with regular snapshots and backups to prevent data loss. High-availability multi-region replication can help the system recover quickly from errors. Incident response strategies protect the security from cyber threats while ensuring system availability and threat resilience(Loukasmäki, 2023).

8.5 Security Compliance and Governance

For GDPR, CCPA, and ISO 27001 compliance, the system needs continuous security monitoring and governance regulations. Amazon Inspector and other penetration testing tools can uncover flaws before attackers do. To reduce cloud hazards, AWS Service Control Policies (SCPs) should be enforced. To reduce phishing and social engineering, companies should train employees in cyber security(Cochran, 2024). To ensure security installations comply with best practices and legislation, use automated auditing tools.

8.6 Conclusion

The system uses AI-powered security controls and AWS infrastructure to detect cyber threats, automate responses, and monitor continually. Cybersecurity includes network segmentation, encryption, access control, and AI-driven anomaly detection. Automatic compliance monitoring, security audits, and extensive backups boost platform resilience. These security methods defend the application against modern cyber threats, ensuring a secure, scalable, and user-friendly environment for threat users.

9 Black Box Testing for the Web Application

Test Case ID	Test Case Description	Test Steps	Expected Result	Actual Result	Status
TC001	Validate Homepage Navigation	Navigate to the homepage. Check the layout, links to "About", "Contact", "Community", and "Admin/Employee".	The homepage loads with all links functional and properly redirecting to respective pages.	Working as expected	Pass
TC002	Validate Login for Registered User	Enter valid email, password, and OTP on the login page.	Successful login, user is redirected to their profile page.	Working as expected	Pass
TC003	Validate Login for Blocked User	Attempt to log in with credentials of a blocked user.	Display message: "Your account was blocked!"	Working as expected	Pass
TC004	Validate Signup for New User	Enter valid details (username, email, etc.) on the signup page.	Signup successful, user receives a unique User ID.	Working as expected	Pass
TC005	Validate Forgot Password Functionality	Enter valid email for password recovery and verify OTP. Create a new password.	OTP is received via email. Password reset is successful.	Working as expected	Pass
TC006	Validate Admin Dashboard Access	Log in as admin using admin credentials.	Admin dashboard loads with options to manage users, employees, and view visual analytics.	Working as expected	Pass
TC007	Validate Threat Detection on Community Page	Post a normal message on the community forum.	Message is posted successfully.	Working as expected	Pass
TC008	Validate Threat Detection on Community Page (Malicious Input)	Post a malicious message (e.g., containing threats or spam).	Message flagged as a threat and reported to the admin.	Working as expected	Pass
TC009	Validate Admin Blocking User for Threat Messages	Admin blocks a user from the dashboard for posting a malicious message.	User is blocked, and further login attempts show the "Your account was blocked!" message.	Working as expected	Pass
TC010	Validate Contact Query Threat Detection	Submit a normal query through the contact form.	Query is processed and forwarded to admin email.	Working as expected	Pass

TC011	Validate Contact Query Threat Detection (Malicious Query)	Submit a malicious query through the contact form.	Query is flagged as a threat and reported to the admin for further action.	Working as expected	Pass
TC012	Validate Employee Dashboard	Log in as an employee and view threat data visuals.	Employee dashboard loads with functionality to block users and view threat data visuals.	Working as expected	Pass
TC013	Validate Admin Editing User Details	Admin edits user details from the dashboard.	User details are updated successfully.	Working as expected	Pass
TC014	Validate Admin Assigning Employee ID	Admin assigns Employee ID and occupation to newly signed-up employees.	Employee receives email with Employee ID and occupation details.	Working as expected	Pass
TC015	Validate Profile Picture Upload	User uploads a profile picture on their profile page.	Profile picture uploads successfully and displays correctly.	Working as expected	Pass
TC016	Validate Logout Functionality	User clicks the logout button.	User is redirected to the login page.	Working as expected	Pass
TC017	Validate Visualization of Threat Data	Admin views bar charts and data visualizations on the "Show Visuals" page.	Visuals display threat data accurately.	Working as expected	Pass
TC018	Validate Admin/Employee Login with Incorrect Credentials	Enter incorrect credentials on the login page.	Displays error message: "Invalid credentials!"	Working as expected	Pass
TC019	Validate Multi-Role Access	Test access for admin, employee, registered user, and unregistered user.	Each role has appropriate access to their respective functionalities.	Working as expected	Pass
TC020	Validate System Scalability (High User Load)	Simulate multiple users accessing the application simultaneously.	Application handles load without downtime or performance degradation.	Working as expected	Pass

10 AWS Deployment

Step 1: Login into AWS Management Console then Open Amazon S3 and then Create Bucket

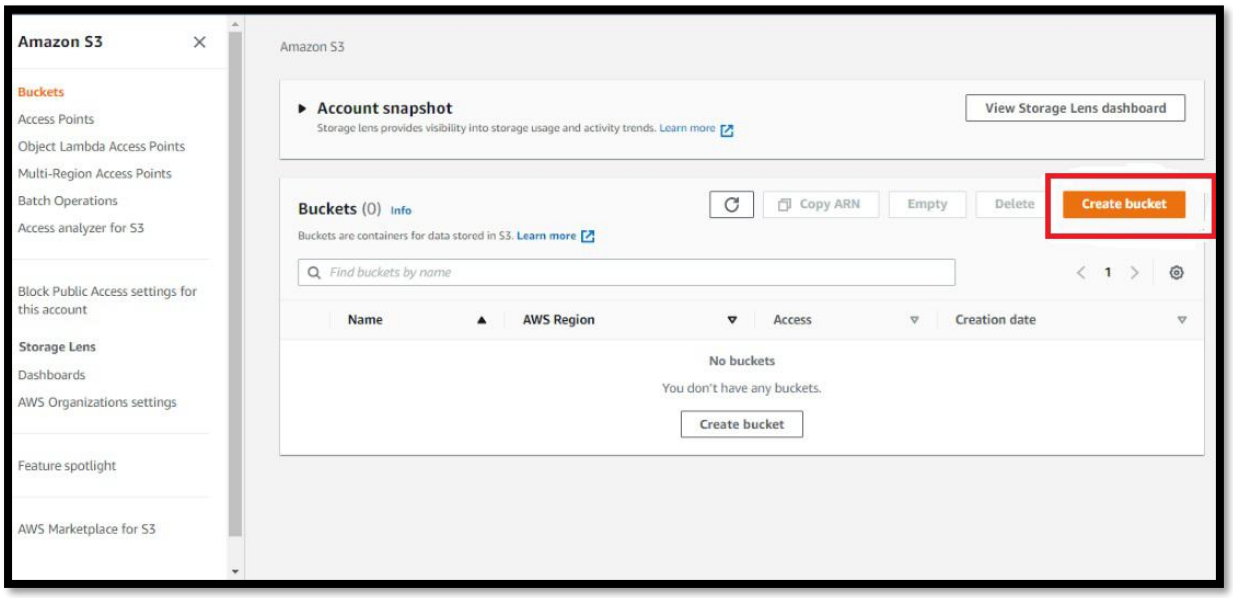


Figure 80 Login into AWS (Self-created)

Click on the “Create bucket” button.

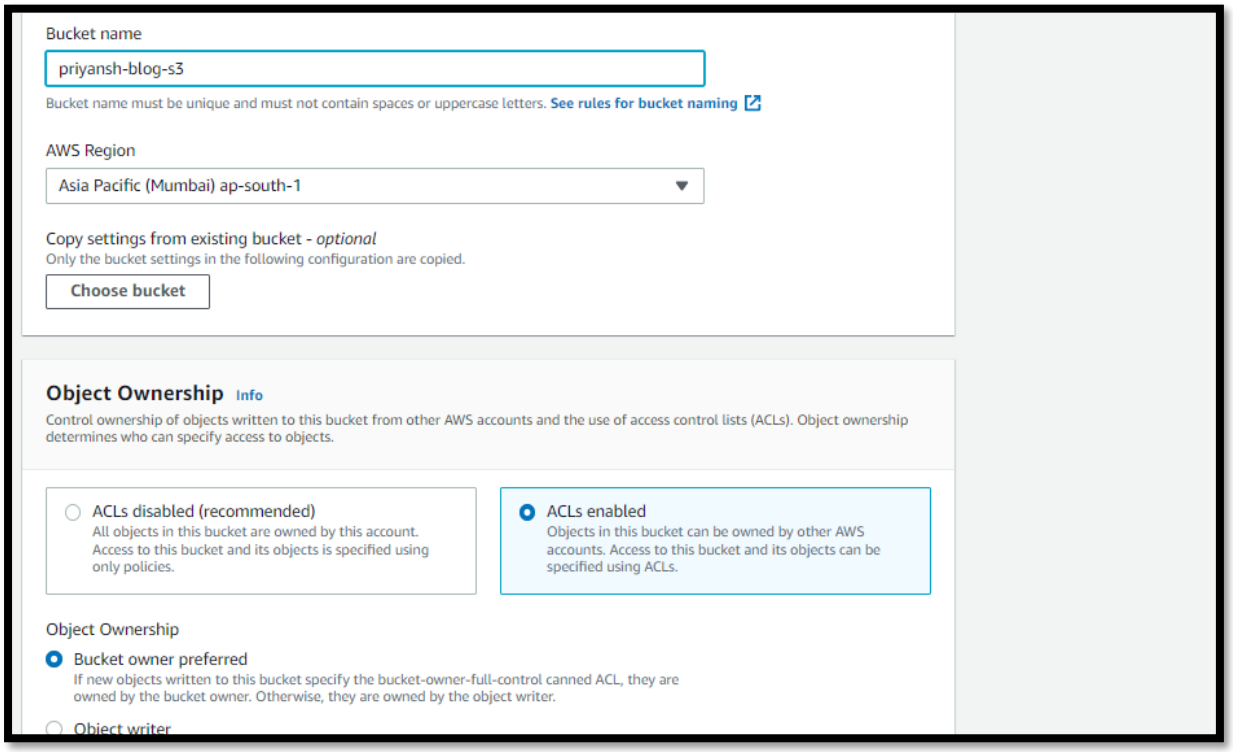


Figure 81 Create bucket (Self-created)

Give Bucket Name, AWS Region and Object Ownership.

Step 2: Block Public Access settings for the bucket

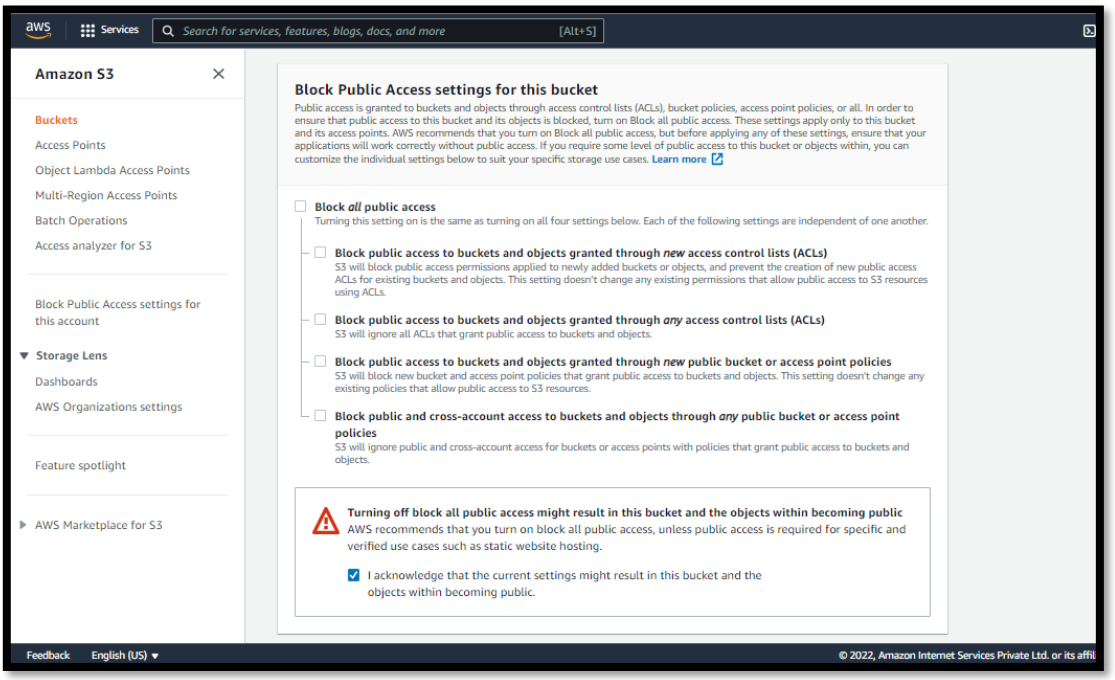


Figure 82 Block Public Access(Self-created)

Checked mark the “Block all public access”.

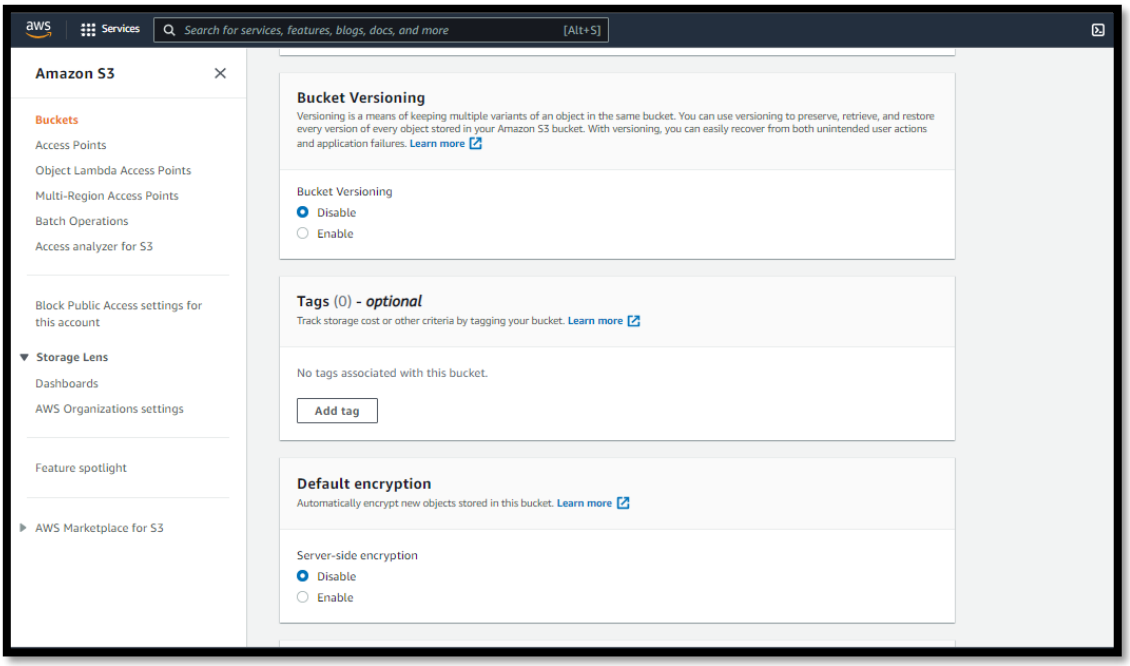


Figure 83 Disable Bucket Versioning (Self-created)

Make Bucket Versioning “Disable”, Add Tags if required and set “Disable” for Default encryption

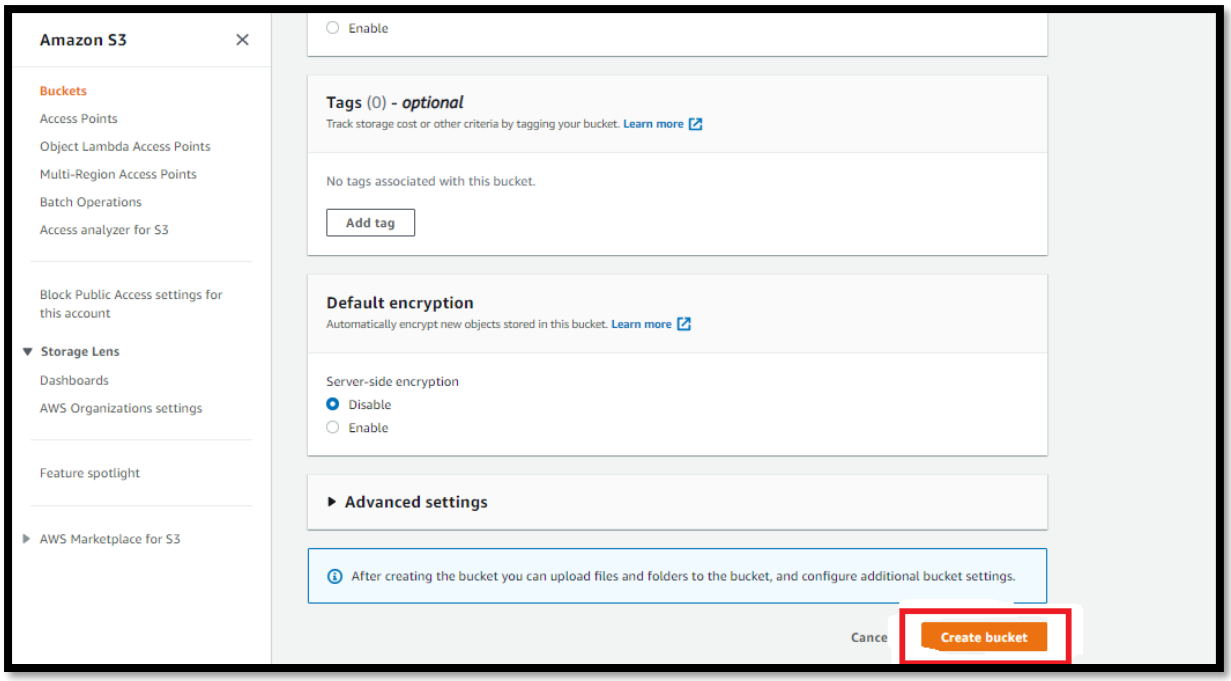


Figure 84 Create Bucket (Self-created)

Click on the “Create bucket” button.

Step 3: Upload code files

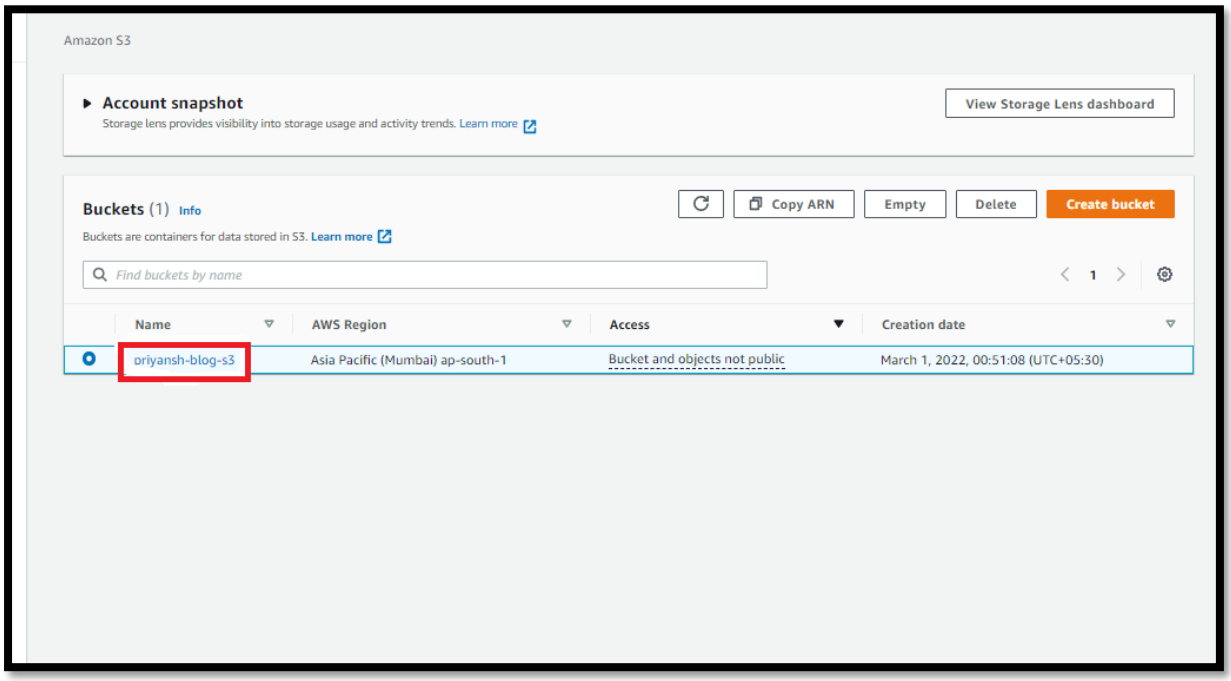


Figure 85 Created bucket (Self-created)

Select Bucket and Click on the created bucket.

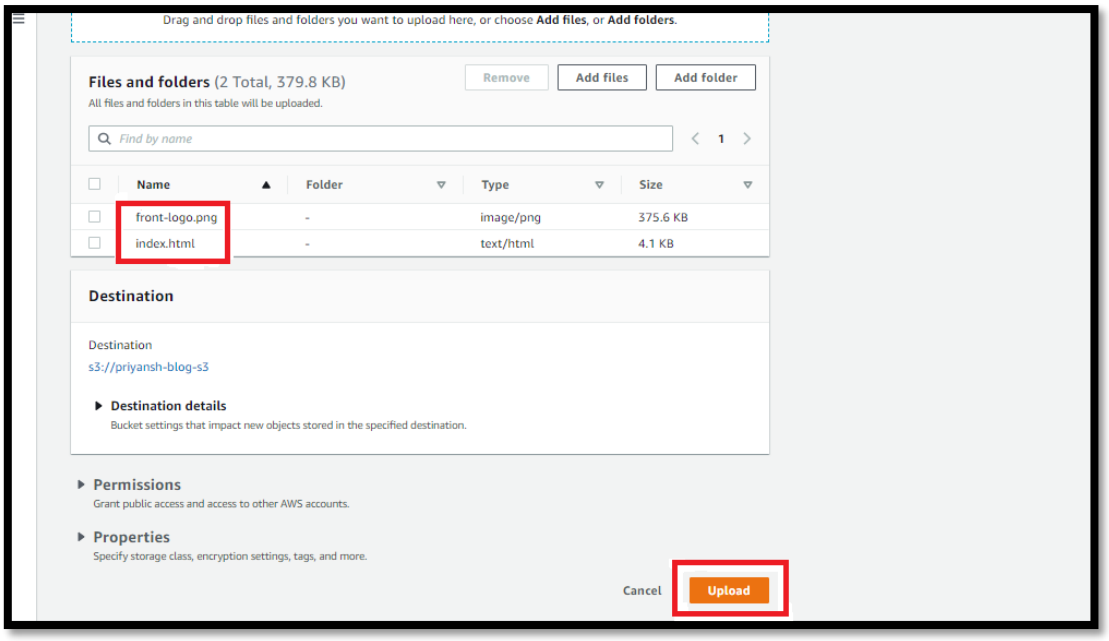


Figure 86 Upload folder (Self-created)

Click on “Upload” then select folder or files to upload. Then select HTML code from local file. At last, after uploading click on “Close”.

Step 4: After the files have been properly uploaded, choose Permissions and proceed as directed.

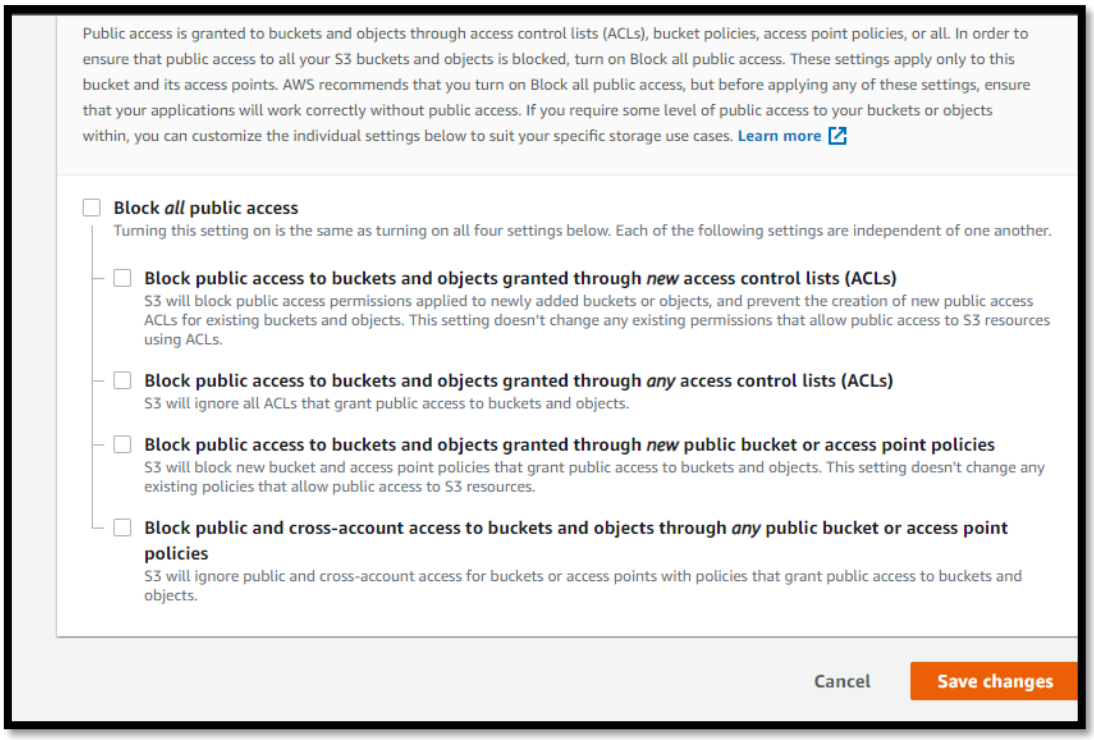


Figure 87 Block public access (Self-created)

In “Block all public access”,

1. Under Bucket Policy, click Edit.
2. Uncheck, Turn off all public access.
3. After saving the modifications, type "confirm."

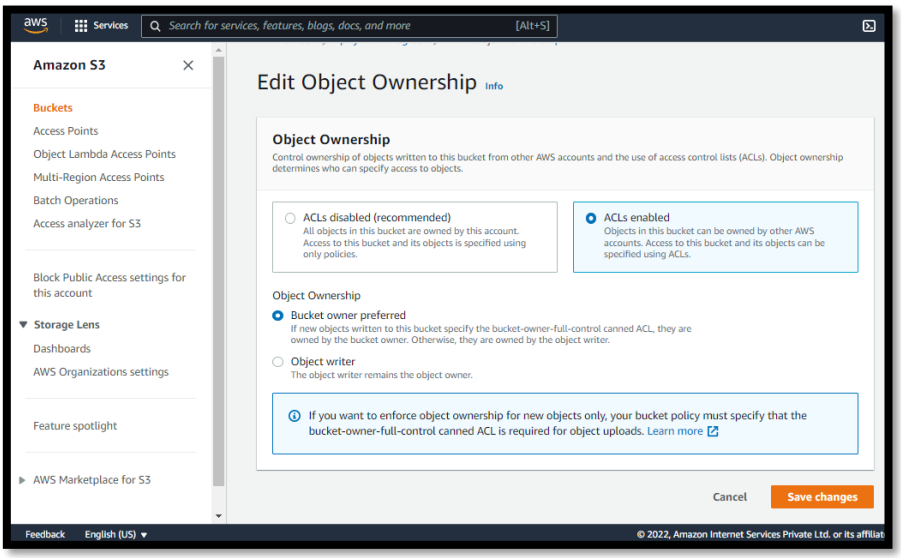


Figure 88 Enable ownership (Self-created)

In “Ownership of the Object”,

1. Select Edit.
2. Select "ACLs Enabled."
3. Verify I admit that it has been restored.
4. Select "Save Changes."

Step 5: Make public Object

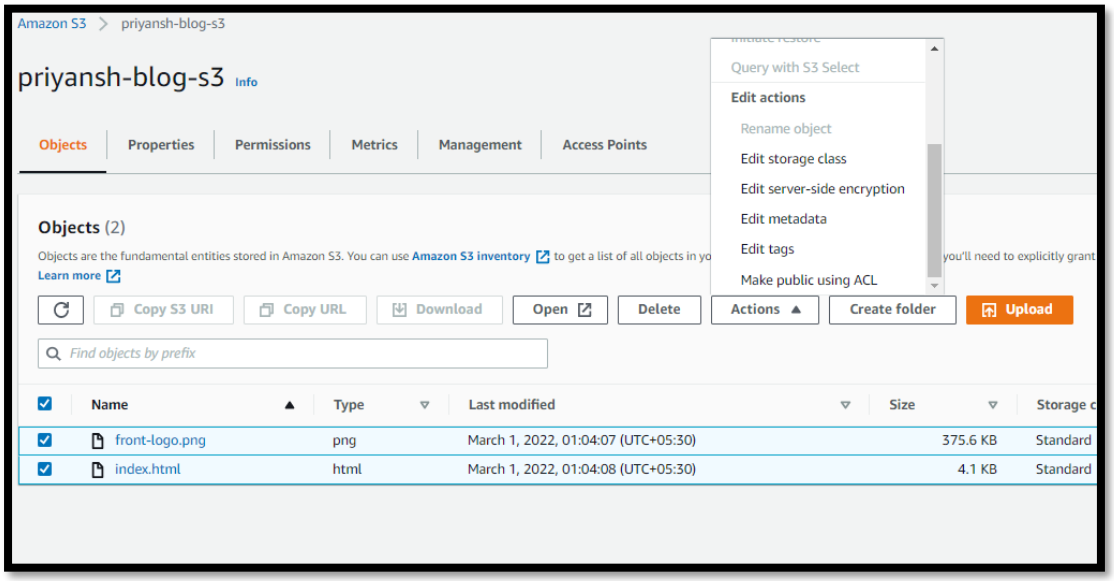


Figure 89 Make public access (Self-created)

- Now, Click on Objects.
- Choose "All Objects."
- Click Actions now.
- Make Public Using ACL is the option.
- Click Close and Make Public now.

Step 6: Copy the Object URL

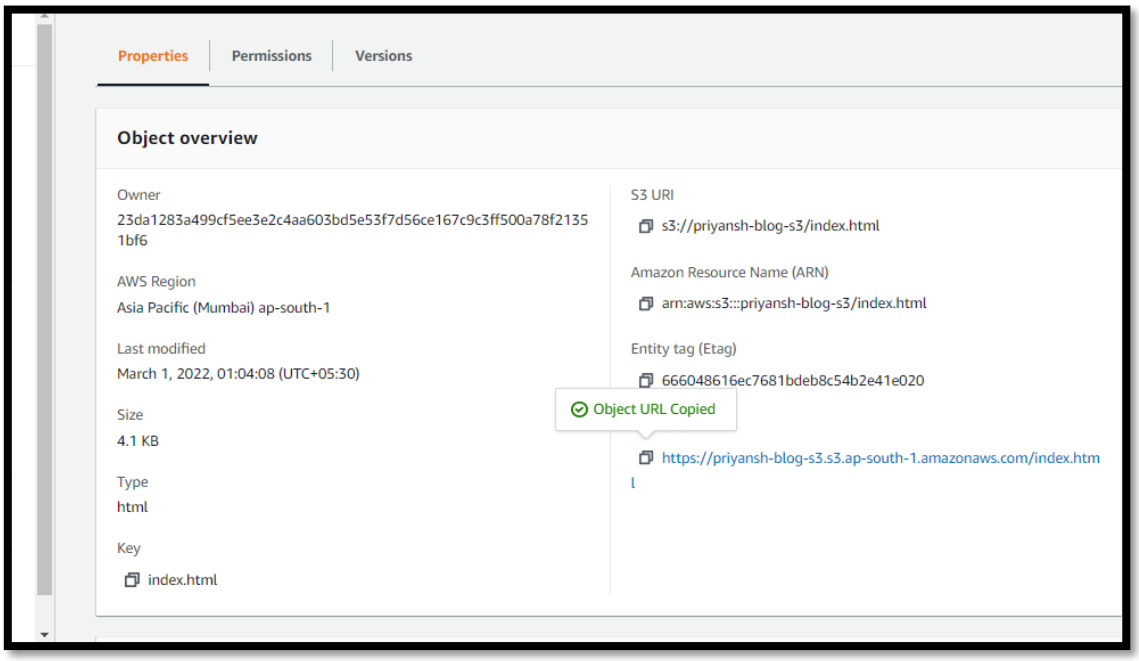


Figure 90 Creating URL link (Self-created)

Step 7: Paste the URL in any browser and check that the Website runs properly.

11 Setup Guidelines & Implementation

11.1 Libraries Packages Installation for AI Threat Detection Using Python

Install necessary libraries via pip command

```
pip install os kagglehub pandas zipfile nltk wordcloud matplotlib seaborn scikit-learn keras pickle
```

Figure 91 Installing library packages (Self-created)

- os: Interacts with the operating system, managing paths and files.
- kagglehub: Used for working with Kaggle datasets in the environment.
- pandas: For data manipulation and analysis (e.g., handling CSV files).
- zipfile: For extracting files from zip archives.
- nltk: Natural Language Toolkit for text processing (e.g., stopwords removal, stemming).
- wordcloud: To generate word cloud visualizations.
- matplotlib & seaborn: For data visualization.
- sklearn (scikit-learn): Provides tools for machine learning (e.g., model selection, evaluation, preprocessing).
- keras: For deep learning models using TensorFlow backend.
- pickle: For saving and loading Python Objects (e.g., machine learning models).

Download the required datasets for stopwords and lemmatization

```
import nltk
nltk.download('stopwords')
nltk.download('wordnet')
```

Figure 92 Importing Library packages (Self-created)

Import all libraries required in google collab using import command

```
import os
import kagglehub
import pandas as pd
import zipfile
import warnings
import numpy as np
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from wordcloud import WordCloud
import matplotlib.pyplot as plt
import seaborn as sns
from collections import Counter

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_extraction.text import TfidfVectorizer
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import Adam
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score
import pickle
```

Figure 93 Importing Library packages (Self-created)

Google Colab Setup

- Upload Dataset to Google Drive.
- Training Model in Google Colab.
- Save the Model Using pickle.
- Download the Saved Model.

11.2 Setup guidelines implementation of Web Application

Install necessary libraries for the Flask application

```
pip install flask flask_sqlalchemy flask-login flask-mail joblib nltk werkzeug
```

Figure 94 Installing library packages (Self-created)

- Flask: A lightweight Python web framework to build web applications.
- Flask-SQLAlchemy: An extension for Flask to add the capabilities of SQLAlchemy, a powerful ORM to interact with a database.
- Flask-Login: An extension for Flask that provides session management and other functionalities, for example, user authentication.
- Flask-Mail: A Flask extension to send email from within the application.
- Joblib: A Python library for easy and efficient serialization of big Python objects, commonly used to save/restore machine learning models.
- NLTK: A comprehensive library for natural language processing tasks such as tokenization, lemmatization, and stopwords removal.
- Werkzeug: A library that provides utilities for constructing and handling HTTP requests and responses in web applications.

Install the necessary libraries for AI threat detection

```
pip install kagglehub pandas zipfile nltk wordcloud matplotlib seaborn scikit-learn keras pickle
```

Figure 95 Installing library packages (Self-created)

Visual Studio Code Setup for Flask Web Application

- Create the project folder (vulnerability_app).
- Create a virtual environment

```
python -m venv venv
```

Figure 96 Creating environment (Self-created)

- Activate the virtual environment.

```
venv\Scripts\activate
```

Figure 97 activating environment (Self-created)

- Create the Flask app (app.py) to handle user authentication , AI threat detection and interactive web interface.
- Create all HTML files and CSS for the frontend.
- Upload Trained Pickle Files in Flask application.
- Check directory structure with all files

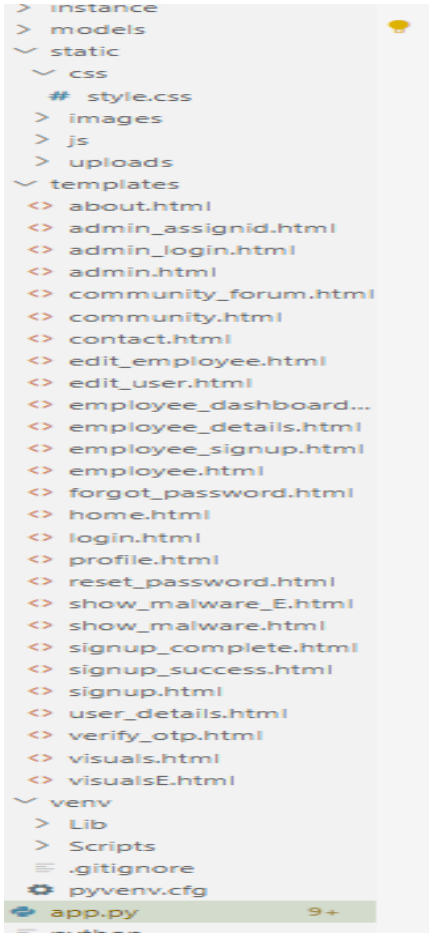


Figure 98 Creating folder (Self-created)

- Run the flask application using below code:

```
python app.py
```

Figure 99 Run code (Self-created)

- Access the app by pasting " http://127.0.0.1:5000/" in browser.

11.3 Setup guidelines implementation of AWS for Cyber Security Policies

1. **IAM – Identify and Access Management**
   - Enable IAM Best Practices
     - ✓ Instead of using the root user, create IAM roles with certain rights.
     - ✓ Make sure that every user has MFA that is multi-factor authentication.
     - ✓ To validate policies, activate AWS IAM Access Analyzer.
     - ✓ Use IAM roles and policies to implement least privilege access.
   - Monitor Activities
     - ✓ Check IAM roles, users, and policies on a regular basis.
     - ✓ Turn on AWS CloudTrail to monitor and audit access.
2. **Network Security**
   - Use VPC (Virtual Private Cloud) for Isolation
     - ✓ Set up private subnets for private information.
     - ✓ To limit traffic, set up network ACLs and VPC security groups.
     - ✓ To stop malicious traffic, use AWS WAF (Web Application Firewall).

- Enable Encryption in Transit
  - ✓ For secure connections, use TLS/SSL.
  - ✓ Make that the load balancer and API gateway are using HTTPS.
- DDoS Protection
  - ✓ Turn on AWS Shield to mitigate DDoS attacks.
  - ✓ Make use of AWS Global Accelerator to direct traffic over secure AWS edges.

3. Secure Application Deployment

- Enable Web Application Security
  - ✓ To stop typical attacks such SQL injection and XSS, use AWS WAF.
  - ✓ For authentication, turn on Amazon API Gateway using AWS Cognito.
- Enable Secure SDLC (Software Deployment Life Cycle)
  - ✓ Use AWS CodePipeline to create CI/CD pipelines.
  - ✓ Use Amazon Inspector to check for security flaws.
- Implement AWS Secrets Manager
  - ✓ Check that Database credentials are safely stored and managed.

4. Compliance & Governance

- Enforce Security Policies
  - ✓ For centralized policy administration, use AWS Organizations.
  - ✓ To ensure security compliance, implement AWS Service Control Policies (SCPs).
- Continuous Compliances Audits
  - ✓ Turn on AWS Audit Manager to do ongoing security audits.
  - ✓ Run AWS Trusted Advisor often to get security advice.

5. Data Protection and Encryption

- Enable Data Encryption
  - ✓ For encryption keys, use KMS (Key Management Service) provided by AWS.
  - ✓ Use S3 SSE-KMS (Server-Side Encryption KMS) to encrypt S3 buckets.
  - ✓ For EC2 instances, enable EBS Volume Encryption.
- Implement Data Backup and Recovery
  - ✓ Automate snapshot backups using AWS Backup.
  - ✓ Turn on cross-region replication in order to recover from disasters.

6. Security Monitoring & Incident Response

- Log All Activities
  - ✓ Turn on API call logging for AWS CloudTrail.
  - ✓ For compliance audits, use AWS Config.
- Real-Time Threat Detection
  - ✓ Use Amazon GuardDuty to identify anomalies.
  - ✓ To consolidate security findings, use AWS Security Hub.
- Automated Incident Response
  - ✓ Configure AWS Lambda functions to react to security alarms automatically.
  - ✓ To conduct security investigations, activate Amazon Detective.

7. Incident Management & Response

- Create an Incident Response Plan
  - ✓ Create security playbooks for situations involving AWS.
  - ✓ For automatic runbooks, use AWS Systems Manager.
- Automate the Remediation of Threats
  - ✓ Use AWS Lambda to implement auto-remediation.
  - ✓ For ongoing monitoring, combine AWS Security Hub with AWS Config.

11.4 Setup guidelines implementation of AWS for Deployment

1. **Select the Appropriate AWS Deployment Plan**
    - For small application, select “Single Server Deployment”
    - For better scalability, select “Multi-Tier Architecture”
    - For modern cloud-native apps, select “Containerization & Serverless”
    - Mostly used architecture is “Multi-Tier Architecture” with a compute later, database layer and load balancer.
2. **Configure AWS infrastructure**
    - For secure access, use AWS IAM
        - ✓ For permission control, create IAM roles and policies.
        - ✓ Turn on MFA, or multi-factor authentication.
    - Configure Amazon VPC
        - ✓ For security, create public and private subnets.
        - ✓ To limit access, use Network ACLs and Security Groups.
    - Use AWS Route 53 for DNS Management
        - ✓ Set up DNS routing and register a domain.
3. **Select a Deployment Compute Option**
    - Amazon EC2
        - ✓ Elastic Compute Cloud
        - ✓ For a conventional server-based deployment, start up EC2 instances.
        - ✓ Auto Scaling Groups should be set up for high availability.
    - AWS Elastic Beanstalk
        - ✓ PaaS for simplified deployment
        - ✓ Automatically scales, monitors, and controls infrastructure.
    - AWS Lambda
        - ✓ Serverless Compute
        - ✓ Install event-driven microservices without having to worry about server management.
    - Amazon ECS/EKS
        - ✓ Containers & Kubernetes
        - ✓ To install Docker containers, use ECS - Elastic Container Service.
        - ✓ For container orchestration based on Kubernetes, use EKS - Elastic Kubernetes Service.
4. **Install and Maintain Databases**
    - Amazon S3 for Storage
        - ✓ Safely store backups and static assets.
    - To implement NoSQL Database, use Amazon DynamoDB
        - ✓ Perfect for serverless databases with good performance.
    - To implement Relational Database, use Amazon RDS
        - ✓ Make use of Aurora, PostgreSQL, or MySQL with scalability and backups that happen automatically.

5. **For Automated Deployment implement CI/CD Pipeline**
    - For automated CI/CD workflow, use AWS CodePipeline
    - To build and test the application code, use AWS CodeBuild
    - For automatically deploy applications, use AWS CodeDeploy –
    - Examples:

Develop & Commit Code – GitHub or CodeCommit
Build & Test – AWS CodeBuild
Deploy to AWS – AWS Elastic Beanstalk, ECS or EC2

6. **Enable the option for auto scaling and load balancing**
    - Amazon ELB – Elastic Load Balancer
        - ✓ Traffic should be divided across several EC2 instances.
        - ✓ For routing based on HTTP/HTTPS, use an application load balancer.
    - ASG – Auto Scaling Groups
        - ✓ Modify capacity automatically in response to demand.

7. **Secure and Track the Deployment**
    - Enable Logging & Monitoring
        - ✓ Utilize Amazon CloudWatch for metrics and logs.
        - ✓ Turn on AWS CloudTrail to monitor API activity.
    - Secure AWS Environment
        - ✓ To defend against online attacks, use AWS WAF.
        - ✓ Turn on AWS Shield to prevent DDoS attacks.
        - ✓ Use AWS KMS to encrypt data.
8. **Optimize the Cost and Performance**
    - Use Amazon CDN - CloudFront to Deliver Content More Quickly
    - Turn on Auto Scaling to Cut Expenses
    - For cost optimization, use AWS Compute Savings Plans.

9. **Backup and Disaster Recovery**
    - Automate snapshots with AWS Backup.
    - Multi-AZ deployments should be set up for high availability.
    - For disaster recovery, enable cross-region replication for S3 and RDS.

10. **Validate and Test the Deployment**
    - To test load, use AWS Performance Insights.
    - For recommendations on cost and security, implement AWS Trusted Advisor.
    - Perform Vulnerability Assessment Penetration Testing

References

Aboukadri, S, Ouaddah, A & Mezrioui, A 2024, "Machine learning in identity and access management systems: Survey and deep dive," *Computers & Security*, 139103729, https://doi.org/10.1016/j.cose.2024.103729.

Abuali, KM, Nissirat, L & Al-Samawi, A 2023, "Advancing Network Security with AI: SVM-Based Deep Learning for Intrusion Detection," *Sensors*, 23(21):8959, https://doi.org/10.3390/s23218959.

Akinbolaji, TJ 2023a, *Advanced integration of artificial intelligence and machine learning for Real-Time threat detection in cloud computing environments*, https://www.irejournals.com/formatedpaper/1704258.pdf.

Akinbolaji, TJ 2023b, *Advanced integration of artificial intelligence and machine learning for Real-Time threat detection in cloud computing environments*, https://www.irejournals.com/paper-details/1704258.

Alam, B 2024, *SVM Python – Easy Implementation of SVM algorithm*, https://hands-on.cloud/svm-python-tutorial/.

Al-Sanjary, OI, Roslan, MAB, Helmi, RAA & Ahmed, AA 2020, "Comparison and detection analysis of network traffic datasets using K-Means clustering algorithm," *Journal of Information & Knowledge Management*, 19(03):2050026, https://doi.org/10.1142/s0219649220500264.

Alshomrani, M, Albeshri, A, Alturki, B, Alallah, FS & Alsulami, AA 2024, "Survey of Transformer-Based Malicious Software Detection Systems," *Electronics*, 13(23):4677, https://doi.org/10.3390/electronics13234677.

Alzaabi, FR & Mehmood, A 2024, "A review of recent advances, challenges, and opportunities in malicious insider threat detection using machine learning methods," *IEEE Access*, 1230907–30927, https://doi.org/10.1109/access.2024.3369906.

Amazon Web Services (AWS) 2025a, *Use Amazon S3 with Amazon EC2 instances - Amazon Elastic Compute Cloud*, https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AmazonS3.html.

Amazon Web Services (AWS) 2025b, *Use the AWS VSS solution to restore data for your instance - Amazon Elastic Compute Cloud*, https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/application-consistent-snapshots-restore.html.

Amazon Web Services (AWS) 2025c, *Security and Compliance - Overview of Amazon web services*, https://docs.aws.amazon.com/whitepapers/latest/aws-overview/security-and-compliance.html.

Amazon Web Services (AWS), 2025, *Security Best Practices - Security Best Practices for Manufacturing OT*, https://docs.aws.amazon.com/whitepapers/latest/security-best-practices-for-manufacturing-ot/security-best-practices.html.

Amazon Web Services (AWS) 2025d, *Protecting Data with encryption - Amazon Simple Storage Service*, https://docs.aws.amazon.com/AmazonS3/latest/userguide/UsingEncryption.html.

Amazon Web Services, Inc. 2024a, *Amazon Virtual Private Cloud User Guide*,.

Amazon Web Services, Inc. 2024b, *Amazon Elastic Compute Cloud Developer Guide*, https://docs.aws.amazon.com/pdfs/ec2/latest/devguide/ec2-dg.pdf.

Anas Brital | Random Forest algorithm explained ., https://anasbrital98.github.io/blog/2021/Random-Forest/.

Anton, SDD, Sinha, S & Schotten, HD 2019, "Anomaly-based Intrusion Detection in Industrial Data with SVM and Random Forests," *2022 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, https://doi.org/10.23919/softcom.2019.8903672.

Ao, Y, Li, H, Zhu, L, Ali, S & Yang, Z 2018, "The linear random forest algorithm and its advantages in machine learning assisted logging regression modeling," *Journal of Petroleum Science and Engineering*, 174776–789, https://doi.org/10.1016/j.petrol.2018.11.067.

Arazzi, M, Arikkat, DR, Nicolazzo, S, Nocera, A, A, RRK, P, V & Conti, M 2023, "NLP-Based Techniques for Cyber Threat Intelligence," *arXiv (Cornell University)*, https://arxiv.org/abs/2311.08807.

Azam, Z, Islam, MdM & Huda, MN 2023, "Comparative analysis of intrusion detection systems and Machine Learning-Based model analysis through Decision Tree," *IEEE Access*, 1180348–80391, https://doi.org/10.1109/access.2023.3296444.

Bayazitov, D & Kozhakhmet, K 2024, "Leveraging Amazon Web Services for cloud storage and AI algorithm Integration: A Comprehensive analysis," *Applied Mathematics & Information Sciences*, 18(6):1235–1246, https://doi.org/10.18576/amis/180606.

Bezerra, A, Silva, I, Guedes, LA, Silva, D, Leitão, G & Saito, K 2019, "Extracting Value from Industrial Alarms and Events: A Data-Driven Approach Based on Exploratory Data Analysis," *Sensors*, 19(12):2772, https://doi.org/10.3390/s19122772.

Bhadouria, A, Bathla, R & Arora, G 2024, "Fortifying digital frontiers: Enhancing login page security against emerging cyber threats," *2022 10th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*, 1–7, https://doi.org/10.1109/icrito61523.2024.10522357.

Bundela, R, Dhanda, N & Gupta, KK 2024, "Identification and analysis of security issues in cloud computing," *IEEE - Online*, 1685–1690, https://doi.org/10.1109/icdt61202.2024.10489443.

Chaganti, KC & S&P Global 2024, *Leveraging Generative AI for Proactive threat intelligence: Opportunities and risks*,.

Chakraborty, A, Biswas, A & Khan, AK 2023, "Artificial intelligence for cybersecurity: Threats, attacks and mitigation," in *Intelligent systems reference library*, pp. 3–25, https://doi.org/10.1007/978-3-031-12419-8_1.

Chavlis, S & Poirazi, P 2021, "Drawing inspiration from biological dendrites to empower artificial neural networks," *Current Opinion in Neurobiology*, 701–10, https://doi.org/10.1016/j.conb.2021.04.007.

Cochran, KA 2024, "Legal and compliance considerations in cybersecurity," in *Apress eBooks*, pp. 431–463, https://doi.org/10.1007/979-8-8688-0432-8_15.

Corallo, A, Lazoi, M & Lezzi, M 2019, "Cybersecurity in the context of industry 4.0: A structured classification of critical assets and business impacts," *Computers in Industry*, 114103165, https://doi.org/10.1016/j.compind.2019.103165.

Dhadhania, A, Bhatia, J, Mehta, R, Tanwar, S, Sharma, R & Verma, A 2023, "Unleashing the power of SDN and GNN for network anomaly detection: State-of-the-art, challenges, and future directions," *Security and Privacy*, 7(1):, https://doi.org/10.1002/spy2.337.

Edozie, E, Shuaibu, AN, Sadiq, BO & John, UK 2025, "Artificial intelligence advances in anomaly detection for telecom networks," *Artificial Intelligence Review*, 58(4):, https://doi.org/10.1007/s10462-025-11108-x.

GeeksforGeeks 2024, *Artificial Neural Networks and its Applications*, https://www.geeksforgeeks.org/artificial-neural-networks-and-its-applications/.

Ghiasi, MM & Zendehboudi, S 2020, "Application of decision tree-based ensemble learning in the classification of breast cancer," *Computers in Biology and Medicine*, 128104089, https://doi.org/10.1016/j.compbiomed.2020.104089.

Goh, H-A, Ho, C-K & Abas, FS 2022, "Front-end deep learning web apps development and deployment: a review," *Applied Intelligence*, 53(12):15923–15945, https://doi.org/10.1007/s10489-022-04278-6.

Gürfidan, R, Ersoy, M & Kilim, O 2023, "AI-Powered cyber attacks Threats and measures," in *Springer eBooks*, pp. 434–444, https://doi.org/10.1007/978-3-031-31956-3_37.

Habbal, A, Ali, MK & Abuzaraida, MA 2023, "Artificial Intelligence Trust, Risk and Security Management (AI TRiSM): Frameworks, applications, challenges and future research directions," *Expert Systems With Applications*, 240122442, https://doi.org/10.1016/j.eswa.2023.122442.

Hess, AS & Hess, JR 2019, "Logistic regression," *Transfusion*, 59(7):2197–2198, https://doi.org/10.1111/trf.15406.

Huang, S, Cai, N, Pacheco, PP, Narrandes, S, Wang, Y & Xu, W 2018, "Applications of Support Vector Machine (SVM) learning in cancer genomics," *Cancer Genomics & Proteomics*, 15(1):, https://doi.org/10.21873/cgp.20063.

Irungu, J, Graham, S, Girma, A & Kacem, T 2023, "Artificial intelligence techniques for SQL injection attack detection," *ICIIT '23*, 38–45, https://doi.org/10.1145/3591569.3591576.

Jackson-Barnes, S 2022, *Multi cloud vs hybrid cloud*, January 25, 2025, https://www.orientsoftware.com/blog/multi-cloud-vs-hybrid-cloud/.

Jeffrey, N, Tan, Q & Villar, JR 2023, "A review of Anomaly detection Strategies to detect Threats to Cyber-Physical Systems," *Electronics*, 12(15):3283, https://doi.org/10.3390/electronics12153283.

Kadiyala, P & Kumar, KA 2022, "A Deep Learning–Based Malware and Intrusion Detection framework," *Chapter 17*, 367–380, https://doi.org/10.1002/9781119857686.ch17.

Kala, EM 2023, "The impact of Cyber security on business: How to protect your business," *Open Journal of Safety Science and Technology*, 13(02):51–65, https://doi.org/10.4236/ojsst.2023.132003.

Karthikeyan & Thenmozhi 2024, "Fortifying the cloud: navigating data security challenges and pioneering Future-Ready solutions," *International Research Journal on Advanced Science Hub*, 6(10):277–301, https://doi.org/10.47392/irjash.2024.039.

Kasri, W, Himeur, Y, Alkhazaleh, HA, Tarapiah, S, Atalla, S, Mansoor, W & Al-Ahmad, H 2025, "From Vulnerability to Defense: The role of large language models in enhancing Cybersecurity," *Computation*, 13(2):30, https://doi.org/10.3390/computation13020030.

Kibria, HB & Matin, A 2022, "The severity prediction of the binary and multi-class cardiovascular disease − A machine learning-based fusion approach," *Computational Biology and Chemistry*, 98107672, https://doi.org/10.1016/j.compbiolchem.2022.107672.

Komaragiri, VB & Edward, A 2022, "AI-Driven vulnerability management and automated threat mitigation," *International Journal of Scientific Research and Management (IJSRM)*, 10(10):980–998, https://doi.org/10.18535/ijsrm/v10i10.ec05.

Louati, F, Ktata, FB & Amous, I 2024, "Enhancing Intrusion Detection Systems with Reinforcement Learning: A Comprehensive Survey of RL-based Approaches and Techniques," *SN Computer Science*, 5(6):, https://doi.org/10.1007/s42979-024-03001-1.

Loukasmäki, H 2023, *Cyber Incident Response in Public Cloud: Implications of modern cloud computing characteristics for cyber incident response*, Jamk University of Applied Sciences, https://www.theseus.fi/bitstream/handle/10024/803156/Thesis_Loukasm%C3%A4ki_Henri.pdf?sequence=2.

Malaiyappan, JNA, Prakash, S, Bayani, SV & Devan, M 2024, "Enhancing cloud Compliance: a machine learning approach," *Deleted Journal*, 2(2):, https://doi.org/10.62127/aijmr.2024.v02i02.1036.

Mane, AS & Ainapure, BS 2021, "Private cloud configuration using Amazon web services," in *Lecture notes in networks and systems*, pp. 839–847, https://doi.org/10.1007/978-981-16-0882-7_75.

Marri, R, Varanasi, S & Chaitanya, SVK 2024, "Integrating Security Information and Event Management (SIEM) with Data Lakes and AI: Enhancing Threat Detection and Response," *Deleted Journal*, 6(1):151–165, https://doi.org/10.60087/jaigs.v6i1.239.

Mbah, NGO & Evelyn, NAN 2024, "AI-powered cybersecurity: Strategic approaches to mitigate risk and safeguard data privacy," *World Journal of Advanced Research and Reviews*, 24(3):310–327, https://doi.org/10.30574/wjarr.2024.24.3.3695.

Miorelli, R, Kulakovskyi, A, Chapuis, B, D'Almeida, O & Mesnil, O 2021, "Supervised learning strategy for classification and regression tasks applied to aeronautical structural health monitoring problems," *Ultrasonics*, 113106372, https://doi.org/10.1016/j.ultras.2021.106372.

Mothukuri, V, Khare, P, Parizi, RM, Pouriyeh, S, Dehghantanha, A & Srivastava, G 2021, "Federated-Learning-Based anomaly detection for IoT security attacks," *IEEE Internet of Things Journal*, 9(4):2545–2554, https://doi.org/10.1109/jiot.2021.3077803.

Niranjanamurthy, M, Amulya, MP, Niveditha, NM & Dayananda, P 2020, "Creating a custom virtual private cloud and launch an elastic Compute Cloud (EC2) instance in your virtual private cloud," *Journal of Computational and Theoretical Nanoscience*, 17(9):4509–4514, https://doi.org/10.1166/jctn.2020.9106.

Nwachukwu, NC, Durodola-Tunde, NK & Akwiwu-Uzoma, NC 2024, "AI-driven anomaly detection in cloud computing environments," *International Journal of Science and Research Archive*, 13(2):692–710, https://doi.org/10.30574/ijsra.2024.13.2.2184.

Nwoye & Nwagwughiagwu 2024, "AI-Driven Anomaly detection for proactive cybersecurity and data breach prevention," *International Journal of Engineering Technology Research & Management*, Vol-08 Issue 11, https://ijetrm.com/issues/files/Nov-2024-21-1732196009-NOV37.pdf.

Oliynyk, D, Mayer, R & Rauber, A 2023, "I know what you trained last summer: a survey on stealing machine learning models and defences," *ACM Computing Surveys*, 55(14s):1–41, https://doi.org/10.1145/3595292.

Pabbath Reddy, AR & National Institutes of Health 2021, "THE ROLE OF ARTIFICIAL INTELLIGENCE IN PROACTIVE CYBER THREAT DETECTION IN CLOUD ENVIRONMENTS," *NeuroQuantology*, 19–19(12):764–773, https://www.researchgate.net/publication/378693448.

Panarin, R 2023, "The role of artificial intelligence in cybersecurity," *Custom Software Development Company*, https://maddevs.io/blog/artificial-intelligence-in-cybersecurity/.

Popov, A 2023, "Feature engineering methods," in *Elsevier eBooks*, pp. 1–29, https://doi.org/10.1016/b978-0-323-85955-4.00004-1.

Prasad, AN 2024, "Regulatory compliance and risk management," in *Apress eBooks*, pp. 485–624, https://doi.org/10.1007/979-8-8688-1023-7_8.

Pruneski, JA, Williams, RJ, Nwachukwu, BU, Ramkumar, PN, Kiapour, AM, Martin, RK, Karlsson, J & Pareek, A 2022, "The development and deployment of machine learning models," *Knee Surgery Sports Traumatology Arthroscopy*, 30(12):3917–3923, https://doi.org/10.1007/s00167-022-07155-4.

Rácz, A, Bajusz, D & Héberger, K 2021, "Effect of dataset size and Train/Test split ratios in QSAR/QSPR Multiclass Classification," *Molecules*, 26(4):1111, https://doi.org/10.3390/molecules26041111.

Rajput, H 2018, *Simplifying logistic regression*, https://dzone.com/articles/machinex-simplifying-logistic-regression.

Sahoo, S, Swain, A, Mohapatra, SK, Pattanaik, S & Senapati, A 2024, "Utilizing AWS Cloud-Enhanced VPN and VPC integration to improve data security and access control in library management systems," *IEEE - Online*, 188–194, https://doi.org/10.1109/icoici62503.2024.10696648.

Sajid, M, Malik, KR, Almogren, A, Malik, TS, Khan, AH, Tanveer, J & Rehman, AU 2024, "Enhancing intrusion detection: a hybrid machine and deep learning approach," *Journal of Cloud Computing Advances Systems and Applications*, 13(1):, https://doi.org/10.1186/s13677-024-00685-x.

Silvestri, S, Islam, S, Papastergiou, S, Tzagkarakis, C & Ciampi, M 2023, "A machine learning approach for the NLP-Based analysis of cyber threats and vulnerabilities of the healthcare ecosystem," *Sensors*, 23(2):651, https://doi.org/10.3390/s23020651.

Singh, A & Paul, S 2020, *Hands-On Python Deep learning for the web: Integrating Neural Network Architectures to Build Smart Web Apps with Flask, Django, and TensorFlow*,.

Sohail, M & Tabet, S 2023, "Data privacy and ransomware impact on Cyber-Physical Systems data protection," in *CRC Press eBooks*, pp. 115–134, https://doi.org/10.1201/9781003262527-7.

Stutz, D, De Assis, JT, Laghari, AA, Khan, AA, Andreopoulos, N, Terziev, A, Deshpande, A, Kulkarni, D & Grata, EGH 2024, "Enhancing security in cloud computing using artificial intelligence (AI)," *Applying Artificial Intelligence in Cybersecurity Analytics and Cyber Threat Detection*, 179–220, https://doi.org/10.1002/9781394196470.ch11.

Subasi, A, Molah, E, Almkallawi, F & Chaudhery, TJ 2017, "Intelligent phishing website detection using random forest classifier," *2017 International Conference on Electrical and Computing Technologies and Applications (ICECTA)*, 1–5, https://doi.org/10.1109/icecta.2017.8252051.

Symeonidis, S, Effrosynidis, D & Arampatzis, A 2018, "A comparative evaluation of pre-processing techniques and their interactions for twitter sentiment analysis," *Expert Systems With Applications*, 110298–310, https://doi.org/10.1016/j.eswa.2018.06.022.

Tariq, AHIE, Tariq, MBIE & Lu, S 2024, "Hybrid AI-Driven techniques for enhancing ZeroDay Exploit Detection in Intrusion Detection System (IDS)," *AIoTC*, 156–160, https://doi.org/10.1109/aiotc63215.2024.10748333.

Tirtakusuma, DI, Sukaridhoto, S, Budiarti, RPN, Mulyo, D, Alfarezi, F, Aminputra, BB & Hakim, OS 2024, "Real-time monitoring using AWS cloud platform for agriculture soil," *IEEE International Symposium on Consumer Technology*, 510–516, https://doi.org/10.1109/isct62336.2024.10791164.

Villegas-Ch, W, Govea, J & Ortiz-Garces, I 2024, "Developing a Cybersecurity Training Environment through the Integration of OpenAI and AWS," *Applied Sciences*, 14(2):679, https://doi.org/10.3390/app14020679.

Wang, B-X, Chen, J-L & Yu, C-L 2022a, "An AI-Powered Network Threat Detection System," *IEEE Access*, 1054029–54037, https://doi.org/10.1109/access.2022.3175886.

Wang, B-X, Chen, J-L & Yu, C-L 2022b, “An AI-Powered Network Threat Detection System,” *IEEE Access*, 1054029–54037, https://doi.org/10.1109/access.2022.3175886.

Wang, B-X, Chen, J-L & Yu, C-L 2022c, “An AI-Powered Network Threat Detection System,” *IEEE Access*, 1054029–54037, https://doi.org/10.1109/access.2022.3175886.

Zhang, WC and J 2024, *Elevating Security operations: The role of AI-Driven Automation in enhancing SOC efficiency and efficacy*, https://journals.sagescience.org/index.php/jamm/article/view/128.