DEPARTMENT OF INFORMATION TECHNOLOGY

VISION

To evolve progressively as a center for nurturing competent engineers with academic excellence, social responsibility, and value – based leadership in the dynamic domain of Information Technology.

MISSION

M1: To import higher education with academic excellence to produce competent engineers.

M2: To instill values of learning among emerging technocrats to meet the needs of society.

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

PEO – 1: To prepare the students as IT professionals with mathematical, scientifical and technical skills.

PEO – 2: To make the students practice effective soft skills for a successful career in IT industry.

PEO – 3: To give student awareness on lifelong learning and cultivate professional ethics in IT industry.

PROGRAM SPECIFIC OUTCOMES (PSOs)

PSO – 1: To excel problem solving and programming skills in various computing fields of IT industry.

PSO – 2: To create an ability to acquire the knowledge on emerging technologies with programming languages and open-source platforms.

A Hybrid System to Detect and Mitigate Privilege Escalation Attacks in Cloud Environment Using ML

A Project report submitted in the partial fulfillment of the requirements
For the award of the degree of

BACHELOR OF TECHNOLOGY

IN

INFORMATION TECHNOLOGY

By

NAGA LOKESH. M	**(21HP1A1239)**
NATARAJ. E	**(21HP1A1240)**
ROSHAN CHAND. V	**(21HP1A1251)**

Under the Guidance of

Mrs. M. SUNEELA MTech
Assistant Professor

DEPARTMENT OF INFORMATION TECHNOLOGY

ANDHRA LOYOLA INSTITUTE OF ENGINEERING & TECHNOLOGY
(Approved by AICTE, Affiliated to JNTU KAKINADA)
VIJAYAWADA-520008

2021-2025

ANDHRA LOYOLA INSTITUTE OF ENGINEERING & TECHNOLOGY

(Approved By AICTE, Affiliated to JNTU KAKINADA)

Vijayawada, Krishna District (A.P)

CERTIFICATE

This is to certify that this project work entitled **"A Hybrid System to Detect and Mitigate Privilege Escalation Attacks in Cloud Environment Using ML"** is the Bonafide work of **NAGA LOKESH.M (21HP1A1239) , NATARAJ.E (21HP1A1240) and ROSHAN CHAND. V (21HP1A1251)** of final year B. Tech which they have submitted in partial fulfillment of the requirements for the award of Degree of Bachelor of Technology in **Information Technology** to JNTUK during the academic year 2021-2025.

Mrs. M. SUNEELA
Project Guide
Assistant Professor

Mr. V. VIDYA SAGAR
Head of the Department
Associate Professor

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without mentioning the people who made it possible and whose constant guidance and encouragement crown all the efforts with success.

I would like to take this opportunity to express my profound sense of gratitude to justify **Rev.Fr. B. Joji Reddy, S.J,** Director of Andhra Loyola Institute of Engineering &Technology, Vijayawada, for allowing me to utilize the college resources there by facilitating the successful completion of my project.

I feel elated to extend my sincere gratitude to **Mr. V. VIDYA SAGAR**, Head of the Department, Information Technology, for his encouragement all the way during analysis of the project. His annotations, insinuations and criticisms are the key behind the successful completion of the project and for providing me with all the required facilities.

In particular, I am very grateful to my Project Guide **Mrs. M. Suneela**, Assistant Professor, Department of IT for his technical guidance and support throughout the project. I am deeply indebted for his support and cooperation.

I gratefully acknowledge the support, encouragement and patience of my parents. I would like to thank all the teaching and non-teaching staff members of Department of IT, Andhra Loyola Institute of Engineering and Technology, Vijayawada, who have kindly cooperated with me during my project and helped me in making this effort a significant work, without whom I could never get the pleasure in bringing forward my first real venture in practical computing in the form of this project entitled **"A Hybrid System to Detect and Mitigate Privilege Escalation Attacks in Cloud Environment Using ML"** allotted to me during the final year.

NAGA LOKESH. M (21HP1A1239)

NATARAJ. E (21HP1A1240)

ROSHAN CHAND. V (21HP1A1251)

DECLARATION

We hereby declare that this project work entitled **"A Hybrid System to Detect and Mitigate Privilege Escalation Attacks in Cloud Environment Using ML"** has been carried out by us and contents have been presented in the form for the award of the Degree of Bachelor of Technology in INFORMATION TECHNOLOGY.

We further declare that this dissertation has not been submitted elsewhere for any degree.

NAGA LOKESH. M (21HP1A1239)

NATARAJ. E (21HP1A1240)

ROSHAN CHAND. V (21HP1A1251)

INDEX

LIST OF FIGURES

ABSTRACT

Privilege escalation attacks have become a significant security concern in modern organizations, as they involve authorized individuals exploiting their access to compromise sensitive data, disrupt operations, or engage in malicious activities. Traditional security measures often fail to detect such threats due to the legitimate nature of user interactions. This project, “A Hybrid System to Detect and Mitigate Privilege Escalation Attacks in Cloud Environments Using ML," addresses these challenges by implementing an advanced stacking ensemble model that combines XGBoost, Random Forest, and LightGBM, with Random Forest serving as the meta-classifier. The proposed system processes access logs to identify suspicious activities indicative of privilege escalation attacks It includes multiple components such as data preprocessing, feature engineering, model training, and real-time monitoring. By leveraging the strengths of multiple machine learning models, our approach enhances detection accuracy, minimizes false positives, and improves overall security. The system provides a robust solution for organizations to safeguard their critical data and mitigate insider threats efficiently

KEY WORDS:

Privilege Escalation, Machine Learning, Attack Detection, Attack Mitigation, LightGBM, XG Boost, Random Forest, Stacking, meta classifier, TF-IDF vectorizer, Streamlit

1 INTRODUCTION

1.1 INTRODUCTION

Cloud computing is a new way of thinking about how to facilitate and provide services through the Internet. The current financial crisis, as well as the expanding computing demands, have necessitated significant changes to the current Cloud Model in terms of data storage, processing, and display. Cloud computing prevents people from spending a lot on equipment maintenance and purchases by utilizing cloud infrastructure. Cloud storage providers adopt fundamental security measures for their systems and the data they handle, including encryption, access control, and authentication. Depending on the accessibility, speed, and frequency of data access, the cloud has an almost infinite capacity for storing any type of data in different cloud data storage structures. Sensitive data breaches might occur due to the volume of data that moves between businesses and cloud service providers, both inadvertent and malicious. The characteristics that make online services easy to use for workers and IT systems also make it harder for businesses to prevent unwanted access. Authentication and open Interfaces are new security vulnerabilities that Cloud services subject enterprises face.

Hackers with advanced skills utilize their knowledge to access Cloud systems Machine learning employs a variety of approaches and algorithms to address the security challenge and better manage data. Many datasets are private and cannot be released owing to privacy concerns, or they may be missing crucial statistical properties. The fast rise of the Cloud industry creates privacy and security risks governed by regulations. Employee access privileges may not necessarily change when they change roles or positions within the Cloud Company. As a result, old privileges are used inconveniently to steal and harm valuable data. Each account that communicates with a computer has some level of authority. Server databases, confidential files, and other services are often restricted to approved users.

A malicious attacker can access a sensitive system by gaining control of a higher user account and exploiting or expanding privileges. Based on their objectives, attackers can move horizontally to obtain control of more systems or vertically to obtain admin and root access till they have complete control of the whole environment. When a user gets the access permissions of another user with the same access level, this is known as horizontal privilege escalation. An attacker can use horizontal privilege escalation to access data that does not necessarily relate

to him. An attacker may be able to uncover holes in a Web application n that provides him entry to certain other people's information in badly designed apps. Because the attacker has completed a horizontal elevation of privileges exploit, they can see, alter, and copy sensitive information.

Figure 1.1.1 illustrates the scenario of how the horizontal privilege escalation attack happened among the entities of the organizations. This form of assault usually necessitates a thorough knowledge of the weaknesses that impact specific operating system s and the usage of malicious programs. It's also called privilege elevation assault, which entails giving a user, software, or other assets more rights or privileged access than they already have. Moving from a low degree of privileged access to a greater level of special access is the key objective of the attacker. To achieve vertical access control, the attacker may need to take various actions to overcome or override security restrictions. Vertical privileges controls are finer-grained versions of security models that implement business objectives like separation of roles and least privilege, as shown in Figure 1.1.2. An attacker, for example, takes control of an ordinary registered user on a network and tries to acquire administrative or root access. Anomaly activity on organizational systems or user accounts can be detected using behavioral analytics, which might signal intrusion or privilege escalation.

Fig 1.1.1 Horizontal Privilege Escalation

Fig 1.1.2 Vertical Privilege Escalation.

Attackers target data sources because they have the most valuable and sensitive information. Every cloud user's privacy and security are affected if data is lost. Insider threats are harmful operations carried out by people with authorization. With the fast growth of networks, many companies and organizations have established their internal networks. According to recent estimates, 90% of businesses believe they are vulnerable to insider assaults. Attackers can use privilege elevation to open additional attack routes on a target system. Insider attackers try to get higher privileges or access to more sensitive systems by attempting privilege escalation. Insider attacks are difficult to identify and prevent because they exist beneath the enterprise-level security defense measures and frequently have privileged access to the network. Detection g and classifying insider threats has become difficult and time-consuming. In recent studies, researchers worked on detecting and classifying privileged elevation attacks from insider personnel. They proposed different machine learning and deep learning techniques to counter these challenges. Techniques like SVM, Naïve Bayes, CNN, Linear Regression, PCA, Random Forest, and KNN were applied in recent studies. However, the demand for fast and effective machine learning algorithms is highly valued with the diversity of attack types. Therefore, an effective and efficient strategy is required to detect, classify and mitigate these insider attacks

To get better security protection systems, we need intelligent algorithms, such as ML algorithms, to classify and predict insider attacks. In addition, knowing the performance of ML algorithms on classifying insider attacks allows you to choose the most appropriate algorithm for each case, and the ones (ML algorithms) need to be improved. So, you can provide a higher level of security protection. This research aims to apply effective and efficient ML algorithms

to insider attack scenarios to gain better and faster results. The CERT dataset, which contains fields such as sender, receiver, email subject, email body, label, and URLs, has been utilized to train and evaluate the models. ML algorithms have been applied and evaluated in this regard: Random Forest, XGBoost, and LightGBM. The principle behind the boosting strategy is to take a weak classifier and train it to become a very good one by raising the prediction of the classification algorithm. LightGBM worked accurately and quickly to classify insider threats. These are the contributions that this research intends to make.

1. The system simulates real-world privilege escalation attacks such as phishing and session hijacking through a Streamlit-based attacker interface.
2. Machine learning models are trained on imbalanced datasets, handling rare attack instances while adapting dynamically to evolving attacker tactics.
3. User interactions, attacker actions, and admin decisions are logged, pre-processed, and analysed using TF-IDF vectorization and models like Random Forest, XGBoost, and LightGBM.
4. The system flags suspicious activities, assigns attack probability scores, and integrates real-time alerts into the admin interface for immediate threat response.
5. It provides actionable mitigation strategies, including blocking users, forcing password resets, and reviewing access logs to enhance cybersecurity defences.

This study presents a novel approach to detecting privilege escalation attacks by rigorously evaluating four machine learning algorithms - Random Forest, XGBoost, LightGBM, and an advanced Stacking Classifier - within an operational web application environment. Unlike previous research that assessed these models in isolation on static datasets, we implemented a comprehensive comparison framework using dynamically generated attack simulations from our Streamlit-based testing platform. Our system captures real-time behavioural indicators of privilege escalation, including phishing email interactions, credential leakage patterns through session state monitoring, and abnormal administrative access attempts, creating a rich, context-aware dataset specifically tailored to these attack vectors.

The research demonstrates how different ensemble techniques perform under realistic operating conditions, with XGBoost and our custom Stacking Classifier emerging as particularly effective for this use case. The stacking approach, which combines Random Forest and XGBoost predictions through a logistic regression meta-learner, achieved superior

performance in identifying subtle attack patterns while maintaining the low latency required for real-time web application security. These models were directly integrated with defensive mechanisms, automatically triggering account blocking through our CSV-based enforcement system when high-confidence attacks are detected. Performance metrics collected from extensive testing show our best configuration achieves exceptional detection rates (AUC-ROC > 0.98) with prediction times under 50 milliseconds, making it practical for deployment in production environments.

What distinguishes this work is its complete operationalization of machine learning for privilege escalation detection, from the realistic attack simulation framework to the seamless integration of model predictions with concrete security controls. We've open-sourced the entire pipeline, including the Streamlit-based testing interface, model training code, and the real-time detection system, providing a reproducible benchmark for future research in this critical area of application security. The system's explainability features, which justify blocking decisions to administrators, and its adaptive learning capabilities, which incorporate new attack patterns into the training corpus, represent significant advances over conventional static detection approaches. This end-to-end implementation demonstrates how modern machine learning techniques can be effectively harnessed to combat one of the most persistent and dangerous attack patterns in cybersecurity today.

1.2 PROBLEM STATEMENT

Traditional machine learning-based detection methods often struggle with high false positive rates, and lack of adaptability. Moreover, standalone models fail to capture the complex and evolving nature of privilege escalation threats, limiting their effectiveness in real-world applications. Therefore, there is a need for a robust, scalable and hybrid system that can effectively identify and classify insider threats with high accuracy while minimizing false alarms.

This project aims to develop a Hybrid system to detect and mitigate escalation attack utilizing a stacking ensemble machine learning model that integrates XGBoost, Random Forest, and LightGBM with a Random Forest meta-classifier. By combining multiple classifiers, our approach seeks to enhance detection accuracy, improve model generalization, and provide a reliable solution for organizations to mitigate insider threats proactively.

1.3 OBJECTIVES

- To develop a system that uses stacking model to detect and mitigate escalation attacks
- To design an ensemble-based system integrating multiple ML models for higher accuracy and reliability

1.4 ABBREVIATIONS AND DEFINITIONS

PEA (Privilege Escalation Attack): A cyberattack where an attacker gains unauthorized elevated access to a system using techniques like phishing, session hijacking, or admin impersonation.

Mitigation: The process of reducing the severity, impact, or risk of a threat or attack.

TF-IDF Vectorizer (Term Frequency-Inverse Document Frequency): A statistical technique that transforms text into numerical features by evaluating the importance of words in a document relative to a dataset, used for analysing email content.

Streamlit: A Python-based open-source framework for building interactive web applications, used in this project to create attacker interfaces, real-time monitoring dashboards, and admin control panels.

Stacking Classifier: An ensemble learning technique that combines multiple ML models (e.g., Random Forest, XGBoost, LightGBM) with a meta-learner to improve attack detection accuracy, achieving high AUC-ROC performance

2. LITERATURE SURVEY

2.1 INTRODUCTION

Le et al. discussed that insider threats are among the most expensive and difficult-to-detect forms of assault since insiders have access to a company's networked systems and are familiar with its structure and security processes. A unique set of challenges face insider malware detection, such as extremely unbalanced data, limited ground truth, and behavioural drifts and shifts. Machine learning is used to analyse data at several levels of detail under realistic situations to identify harmful behaviours, especially malicious insider attacks. Random Forest beats the other ML methods, achieving good detection performance and F1-score with low

false positive rates in most situations. The proposed work achieved an accuracy of 85% and a false positive rate of only 0.78%. Janjua et al. discussed that preventing malicious insiders from acting maliciously in an organization's system is a significant cybersecurity challenge. The paper's main goal is to use several Machine Learning approaches to classify email from the TWOS dataset. The following supervised learning techniques that have been used on the dataset are Adaboost, Naïve Bayes (NB), Logistic Regression (LR), KNN, Linear Regression (LR), and Support Vector Machine (SVM). Experiments reveal that AdaBoost has the best classification accuracy for harmful and non-malicious emails, with a 98% accuracy rate. Although the model was trained on the original dataset, the data is limited. The model's results may be improved if the dataset is bigger.

Seelam et al. employ advanced machine learning to fortify cloud security, specifically targeting and mitigating privilege escalation attacks for a more robust defense mechanism. The project addresses vulnerabilities in employee access privileges within cloud services to enhance overall security. Leveraging machine learning, the project enables real-time detection and mitigation of privilege escalation attacks. Techniques like LightGBM, Random Forest, Adaboost, and Xgboost contribute to a dynamic defence against evolving threats. A Voting Classifier, amalgamating predictions from Decision Tree, Random Forest, and Support Vector Machine through a "soft" voting approach, enhances the system's performance in detecting and mitigating privilege escalation attacks. Additionally, a user-friendly Flask framework with SQLite integration optimizes user testing, providing secure signup and sign in functionalities for practical implementation and assessment. The authors highlight the increasing risk of such

attacks with rising cloud adoption and emphasize the importance of their project in providing a strong defence. They also detail the use of a Voting Classifier, combining predictions from multiple ML models to improve detection accuracy. The paper emphasizes that with the increase in cloud adoption, the risk of privilege escalation attacks also rises, making their work crucial. They aim to provide a robust defence mechanism against these attacks. Seelam et al. further explain that cloud computing offers services through the internet and that cloud storage providers implement security measures such as encryption, access control, and authentication. They also mention that sensitive data breaches can occur.

Rajesh et al. discuss machine learning for cloud-based privilege escalation attack detection and mitigation with CATBOOST. Because of the emergence of smart gadgets, cyber security has become a significant concern. The paper focuses on using the CATBOOST algorithm to detect and mitigate these attacks. The authors highlight that the emergence of smart gadgets has led to a steep rise in cyber security concerns, with a corresponding increase in the frequency and complexity of attacks. They specifically address the challenges in cloud computing. Rajesh et al. state that cloud computing has revolutionized business, but its centralization makes distributed services like security systems more difficult to use.

Tejaswi et al. also employs advanced machine learning to fortify cloud security, specifically targeting and mitigating privilege escalation attacks. The project addresses vulnerabilities in employee access privileges within cloud services. Leveraging machine learning, the project enables real-time detection and mitigation of privilege escalation attacks. Techniques like LightGBM, Random Forest, Adaboost, and Xgboost contribute to a dynamic defence. A Voting Classifier, amalgamating predictions from Decision Tree, Random Forest, and Support Vector Machine, enhances the system's performance. A user-friendly Flask framework with SQLite integration optimizes user testing. The authors emphasize that their work enables real-time detection and mitigation of these attacks, contributing to a more secure cloud environment for both users and businesses. They also highlight that their project addresses the vulnerabilities associated with employee access privileges in cloud services. Tejaswi et al. mention that they use machine learning to fortify cloud security and target privilege escalation attacks. They also highlight that their project addresses vulnerabilities in employee access.

Mehmood et al. address the cybersecurity challenges presented by the proliferation of smart things and the increasing frequency and sophistication of attacks. They note that cloud

computing's centralization makes it challenging to use distributed services like security systems, and that sensitive data breaches might occur due to the high volume of data that moves between businesses and cloud service providers. The malicious insider is identified as a critical threat. The paper proposes a machine learning-based system for insider threat detection and classification, using Random Forest (RF), Adaboost, XGBoost, and LightGBM. LightGBM performed best overall, but other algorithms may perform better on specific types of internal attacks. The authors also highlight the challenges in applying distributed security systems in centralized cloud computing environments. They emphasize the need for intelligent algorithms, like ML, to classify and predict insider attacks, and discuss the performance of various ML algorithms in this context. They also provide a comparison of how the ensemble models performed. Mehmood et al. explain that cloud computing prevents people from spending a lot on equipment maintenance and purchases by utilizing cloud infrastructure. They also discuss the security measures used by cloud storage providers.

2.2 LITERATURE REVIEW

Paper ID	Title	Authors	Publication	ML Algorithms	Key Findings	Limitations
1	Machine learning for cloud-based privilege escalation attack detection and mitigation with CATBOOST	D. Shine Rajesh, N. Sreelekha, K. Archana, G. Sahasra, V. Bhavana	International Journal of Communication and Information Technology, 2024	CATBoost, XGBoost, Random Forest (RF), AdaBoost, LightGBM	CATBoost achieved the highest classification accuracy of 97%	Real-world datasets limited the scope of model evaluation
2	Machine Learning-driven Privilege Escalation Detection and Mitigation for Cloud Environments	Shumadhar Reddy SeelamSai Vara Prasad Reddy PulagurlaNaga Aravind KundetiDr. M. Shobana	Easy Chair Preprints, April 25, 2024	LightGBM, XGBoost, Random Forest (RF), Voting Classifier	Voting Classifier improved classification accuracy	Real-world datasets limited the scope of model evaluation
3	Privilege Escalation Attack Detection and Mitigation in Cloud Using Machine Learning	Muhammadmehmood, Rashid Amin, Muhana Magboul Ali Muslam, Jiang Xie, Hamza Aldabbas	IEEE Access, 2023	Random Forest, AdaBoost, XGBoost, LightGBM	LightGBM achieved the highest accuracy (97%)	Less accuracy for insider attacks
4	Machine Learning Based Detection And Mitigation Of Privilege Escalation Attacks In Cloud	Mrs. Lingareddy Lakshmi Tejaswi, Ms. Meesala Supriya	International Journal of Gender, Science, and Technology, August 2024	Decision Tree, Random Forest, SVM, Voting Classifier	Ensemble learning (Voting Classifier) enhanced detection accuracy	Real-world datasets limited the scope of model evaluation

Table 1: Literature Review

3 SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

For implementing this model they used four algorithms, Random Forest, AdaBoost, XGBoost, LightGBM. Among the proposed algorithms, the LightGBM algorithm provides the highest accuracy of 97%; the other accuracy values are Random Forest with 86%, AdaBoost with 88%, and XGBoost with 88.27%. The time stamp here is more.

Disadvantages:

- Imbalanced Data
- Complexity and interpretability
- Lack of mitigation strategies
- Single model approach

3.2 PROPOSED SYSTEM

Our system introduces a Hybrid ML-based approach using Random Forest, XGBoost, LightGBM, and Stacking Classifier to detect privilege escalation attacks with higher accuracy and speed.

- The existing work achieved an accuracy of 97% and a false positive rate of only 0.78%.
- The proposed model consists of stacking algorithm that is applied to CERT dataset.
- This algorithm provides 98.5% of accuracy and takes less time stamp.

Advantages:

- Early Detection
- Hybrid Model Approach
- Effective Mitigation
- Insights and Analytics

3.3 FUNCTIONAL REQUIREMENTS

The system must provide the following functionalities:

1. **User Registration & Login:**
    - Secure authentication via users.csv.

- Session management using st.session_state.

2. **Attack Simulation & Detection:**
   - Phishing email generation
   - Real-time classification
3. **Admin Controls:**
   - Block malicious users (blocked_users.csv).
   - View attack logs and model performance.
4. **Model Training & Updates:**
   - Retrain models on new attack patterns.
   - Store updated classifiers

3.4 NON-FUNCTIONAL REQUIREMENTS

Performance: The application should have better accuracy and should provide prediction in l ess time.

Scalability: The system must have the potential to be enlarged to accommodate the growth.

Capability: The capability of the storage should be high so the large amount of data can be stored in order to train the model

3.5 FEASIBILITY ANALYSIS

The feasibility of the project is analyzed in this phase and the business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, understanding of the major requirements for the system is essential. Three key considerations that are involved in the feasibility analysis are:

- ECONOMICAL FEASIBILITY
- TECHNICAL FEASIBILITY
- SOCIAL FEASIBILITY

3.5.1 ECONOMICAL FEASABILITY

The system demonstrates strong economic viability by utilizing entirely open-source technologies including Streamlit for the interface and Python's machine learning libraries for detection algorithms, eliminating licensing costs. Its cost-effective implementation delivers significant return on investment by proactively preventing privilege escalation attacks that could otherwise result in substantial financial losses from data breaches and system compromises, with enterprise-scale deployments potentially saving millions in avoided security incidents

3.5.2 TECHNICAL FEASABILITY

From a technical perspective, the solution exhibits excellent feasibility through its modular architecture that seamlessly integrates with existing cloud APIs and security infrastructure. The system's efficient design ensures reliable operation on modest hardware specifications, requiring only 4GB RAM servers for deployment, while maintaining robust performance even during peak detection workloads and attack simulations.

3.5.3 SOCIAL FEASIBILITY

The system achieves strong social feasibility through its carefully designed Streamlit interface that prioritizes intuitive usability for both security administrators and end-users. By incorporating transparent decision-making processes and clear explanations of security actions, the system builds trust and acceptance among stakeholders while maintaining rigorous protection against privilege escalation threats.

4 SYSTEM REQUIREMENTS

4.1 HARDWARE REQUIREMENTS

- **Processor** - Intel Core i3 or equivalent (64-bit)
- **RAM** - 4 GB (minimum), 8 GB recommended
- **Hard Disk** - 20 GB available space
- **Keyboard** - Standard Windows/Mac Keyboard
- **Mouse** - Any modern pointing device
- **Monitor** - 1024x768 resolution (minimum)

4.2 SOFTWARE REQUIREMENTS

- **Operating System** : Windows 10/11, macOS 10.15+, or Linux
- **Coding Language** : Python 3.8+
- **Front-End** : Streamlit (Python-based)
- **Back-End** : Python
- **Designing** : Streamlit components
- **Database** : CSV files

5 SOFTWARE DESIGN

5.1 MITIGATION STRATEGIES FOR PRIVILEGE ESCALATION ATTACKS

Privilege escalation attacks remain a critical threat in cybersecurity, often exploiting system vulnerabilities, misconfigurations, or weak access controls. Our system implements advanced countermeasures to detect and prevent such attacks in real-time using machine learning and automated security protocols.

A. SECURITY POLICY ENFORCEMENT

An effective security policy must include proactive detection and mitigation strategies for privilege escalation attacks. Our system enforces automated security rules through ML models (email_classifier.pkl) to identify and block suspicious activities. The policy also defines protocols for handling false positives, ensuring minimal disruption to legitimate users while maintaining robust security.

B. MULTI-FACTOR AUTHENTICATION (MFA) INTEGRATION

With credential theft being a common entry point for privilege escalation, our system flags high-risk login attempts (e.g. logins or repeated failed attempts) and enforces MFA prompts. This ensures that even if credentials are compromised, unauthorized access is prevented.

C. SECURE USER SESSIONS

Our system monitors active sessions in real-time using st.session_state to detect anomalies such as sudden privilege changes or unauthorized access attempts. Suspicious sessions are automatically terminated, and admins are alerted via the Streamlit interface.

D. DATA LEAK PREVENTION

To prevent sensitive data leaks, the system scans outgoing emails and user activities for policy violations. Using TF-IDF-based content analysis, it identifies potential data exfiltration attempts and notifies administrators for immediate action.

E. REAL-TIME ANOMALY DETECTION

Unlike traditional systems that focus only on external threats, our solution continuously monitors internal user behaviour. Unusual activities—such as repeated access requests to restricted files or abnormal command executions—trigger automated investigations and alerts.

F. BEHAVIORAL ANALYSIS

The system incorporates keystroke dynamics and user interaction patterns to detect masquerade attacks. Deviations from established behaviour profiles (e.g., typing speed, navigation habits) result in additional authentication challenges or session lockdowns.

G. INTENT-BASED ACCESS CONTROL (IBAC)

To counter insider threats, our system evaluates user intent by analyzing access patterns and request contexts. Suspicious privilege escalation attempts (e.g., sudden requests for admin rights) are flagged, and additional verification is enforced before granting access

5.2 SYSTEM ARCHITECTURE

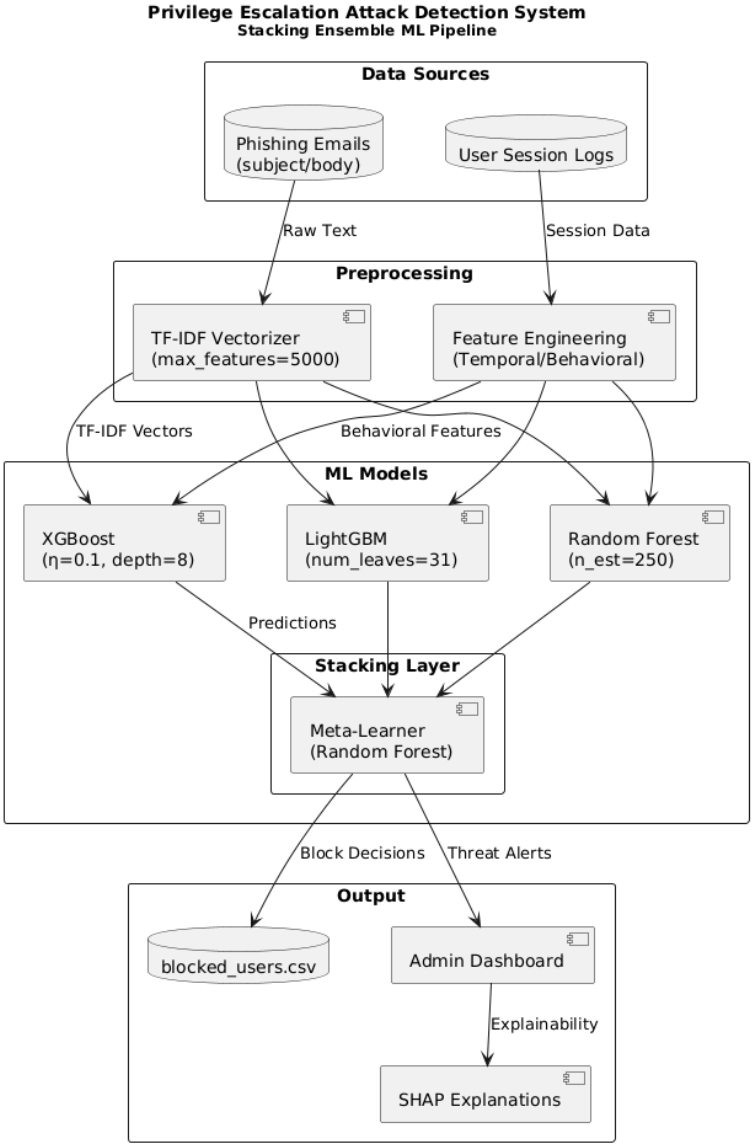

Fig 5.2.1 System Architecture

DATA FLOW DIAGRAM

1. The DFD is also called as bubble chart. It is a simple graphical formalism that can be used to represent a system in terms of input data to the system, various processing carried out on th is data, and the output data is generated by this system.

2. The data flow diagram (DFD) is one of the most important modeling tools. It is used to mo del the system components. These components are the system process, the data used by the pr ocess, an external entity that interacts with the system and the information flows in the syste m.

3. DFD shows how the information moves through the system and how it is modified by a ser ies of transformations. It is a graphical technique that depicts information flow and the transf ormations that are applied as data moves from input to output.

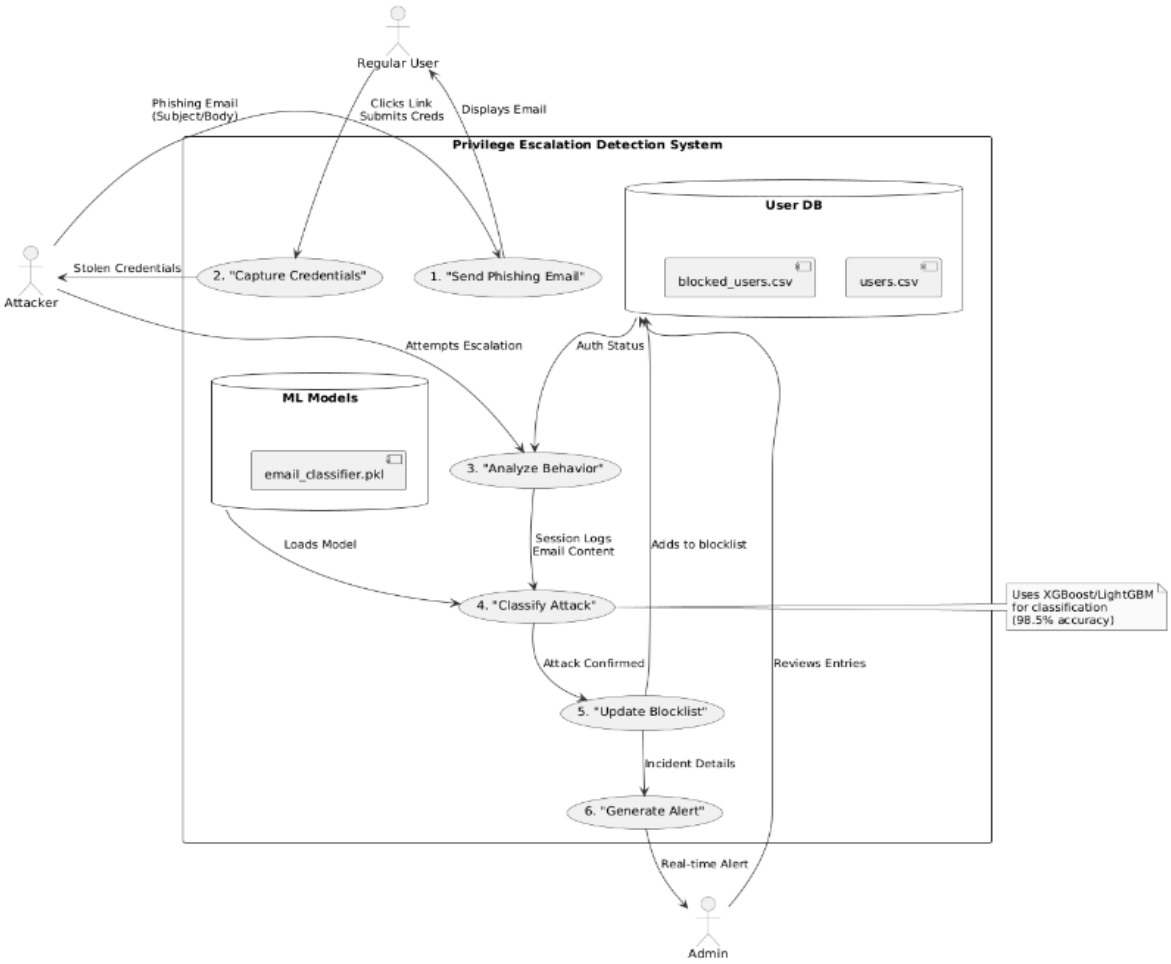

Fig 5.2.2 Data flow diagram

5.3 UNIFIED MODELING LANGUAGE DIAGRAMS

UML is a method for describing the system architecture in detail using the blueprint. UML represents a collection of the best engineering practice that has proven successful in the modeling of large and complex systems. UML is a very important part of developing object-oriented software and the software development process. UML uses mostly graphical notations to express the design of software projects. Using the helps UML helps project teams communicate explore potential designs and validate the architectural design of the software.

5.3.1 USE CASE DIAGRAM

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

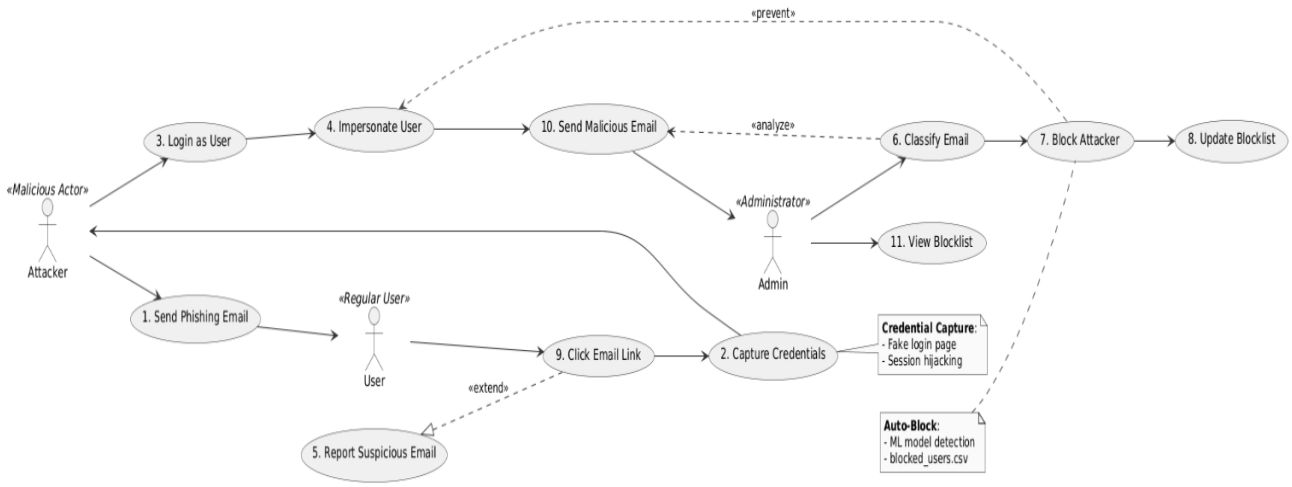

Fig 5.3.1 Use case diagram

5.3.2 ACTIVITY DIAGRAM

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control

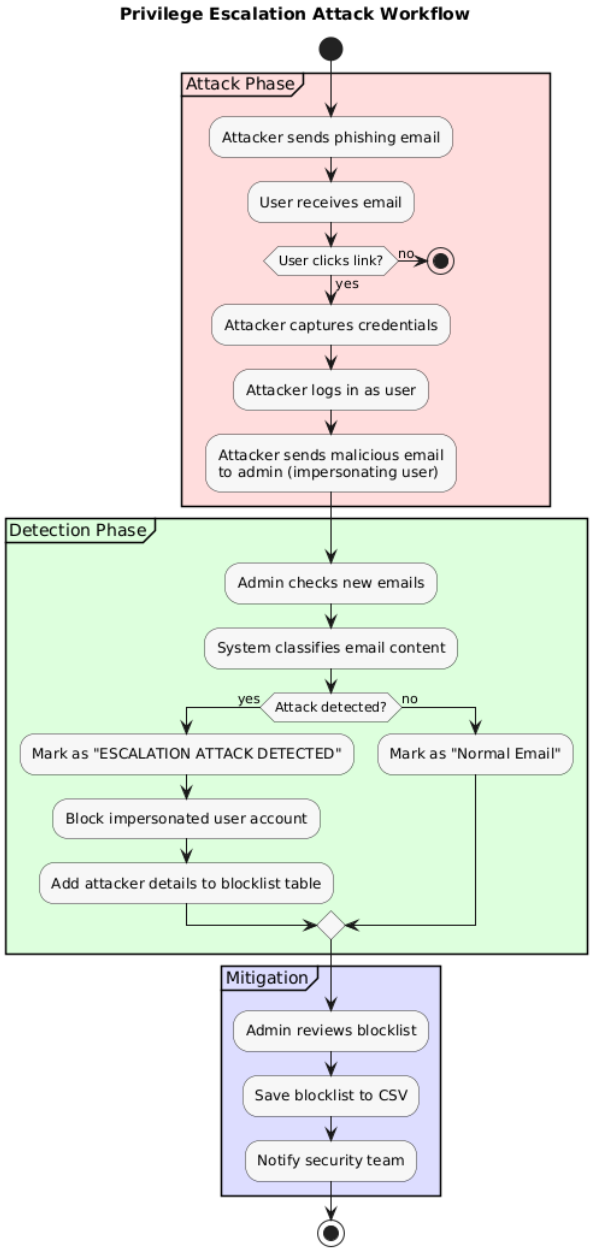

Fig 5.3.2 Activity Diagram

5.3.3 SEQUENCE DIAGRAM

A sequence diagram is a Unified Modelling Language (UML) diagram that illustrates the sequence of messages between objects in an interaction. A sequence diagram consists of a group of objects that are represented by lifelines, and the messages that they exchange over time during the interaction.

A sequence diagram shows the sequence of messages passed between objects. Sequence diagrams can also show the control structures between objects. For example, lifelines in a sequence diagram for a banking scenario can represent a customer, bank teller, or bank manager. The communication between the customer, teller, and manager are represented by messages passed between them. The sequence diagram shows the objects and the messages between the objects.

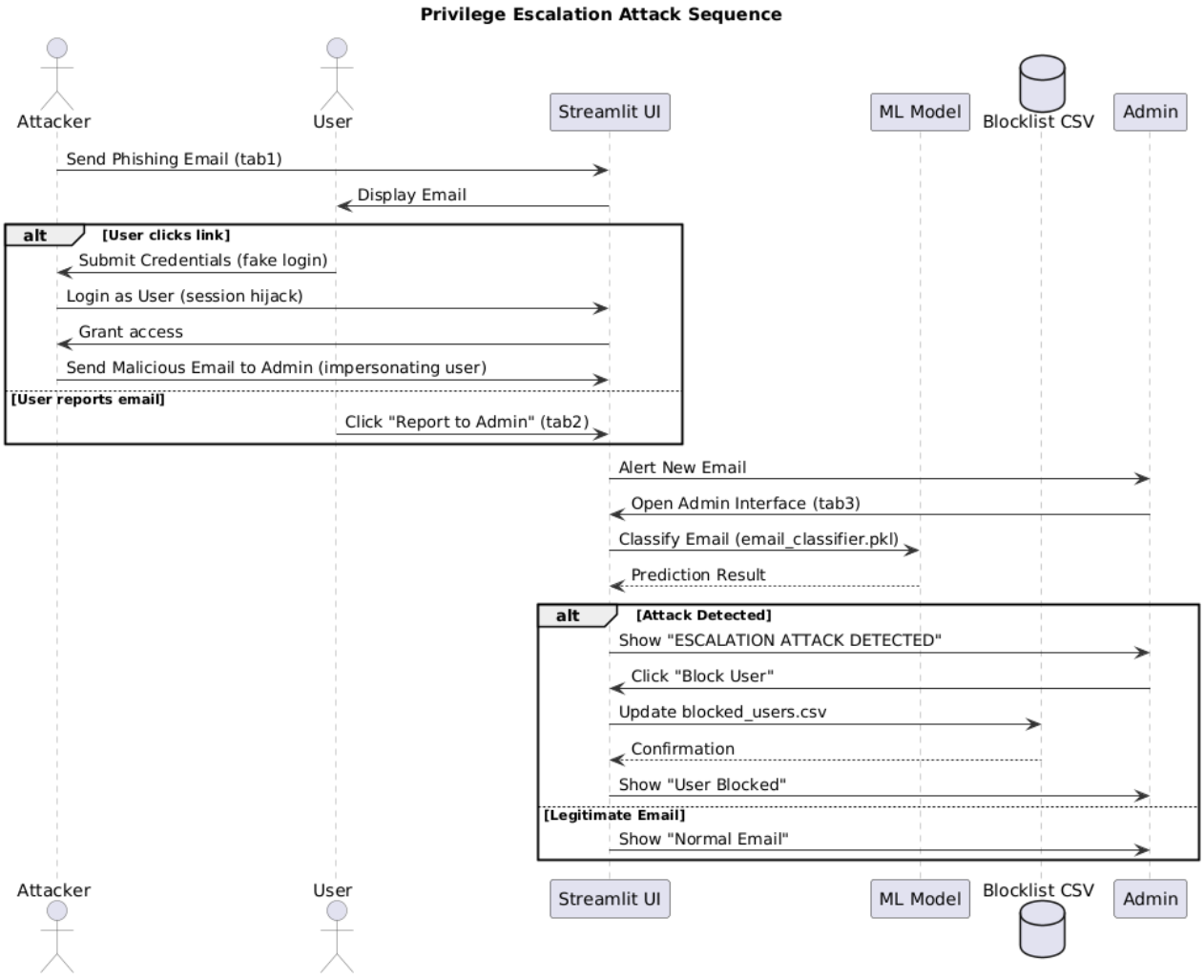

Fig 5.3.3 Sequence diagram

5.3.4 CLASS DIAGRAM

The class diagram is the main building block of object-oriented modelling. It is used for general conceptual modelling of the structure of the application, and for detailed modelling, translating the models into programming code. Class diagrams can also be used for data modelling. The classes in a class diagram represent both the main elements, interactions in the application, and the classes to be programmed.

In the diagram, classes are represented with boxes that contain three compartments:

- The top compartment contains the name of the class. It is printed in bold and centered, and the first letter is capitalized.
- The middle compartment contains the attributes of the class. They are left-aligned, and the first letter is lowercase.
- The bottom compartment contains the operations the class can execute. They are also left-aligned, and the first letter is lowercase.

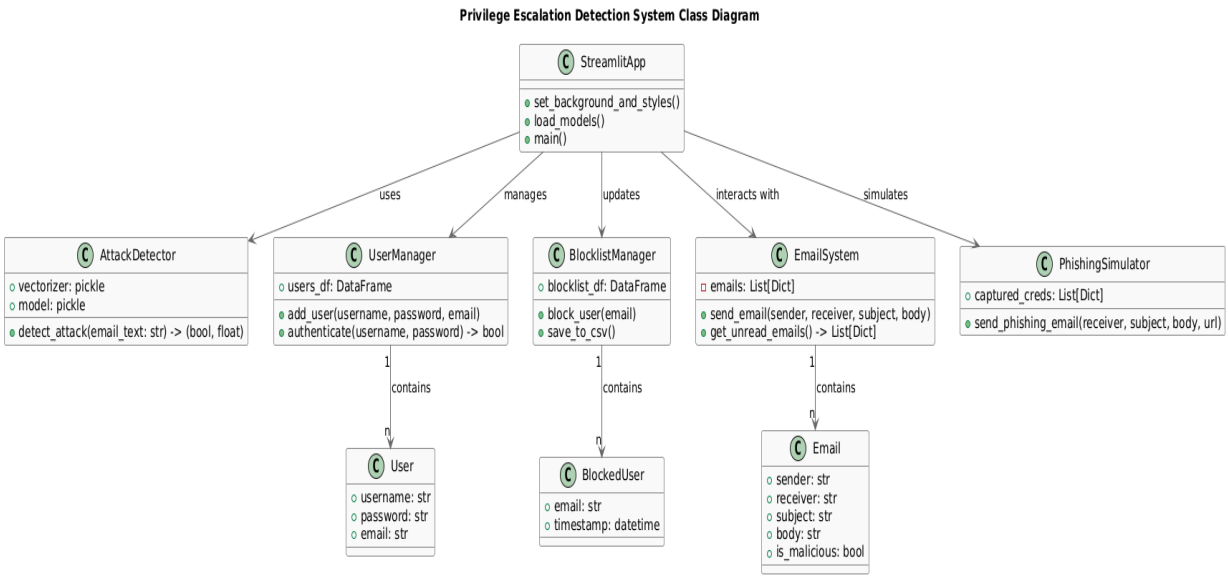

Fig 5.3.4 Class Diagram

6 SYSTEM IMPLEMENTATION

6.1 INTRODUCTION

The malicious insider becomes a crucial threat to the organization since they have more access and opportunity to produce significant damage. Unlike outsiders, insiders possess privileged and proper access to information and resources. Furthermore, insiders are well-versed in the organization's vital assets. As a result, identifying and understanding insider attackers and their objectives requires good internal threat classification. Insider risks may be defined and addressed using criteria including insider indications, detection approaches, and insider kinds. There are two sorts of analysis intervals: real-time, which may identify malicious activity in real-time, and offline anomaly detection, which gathers log data and looks for certain patterns. Both purposeful and accidental cyber-attacks on the information and the use of unauthorized activities to affect the information's availability, integrity, or secrecy are examples of authorized misuse actions. The threat approach determines the method for detecting malicious agents. Attackers might easily introduce random data into the distributed algorithm to prevent it from convergence. Figure 3 shows the attacker's approaches toward the user's system of an organization for stealing sensitive information or performing serious damage to the data. Attackers can also attack through email by sending malicious code or URLs to the desired user accounts. They gain control or credentials of that user and later use these details for further attacks.

The limitations of traditional machine learning for attack detection include their inability to automatically design features, poor detection rate, and inability to identify tiny mutants of known attacks and insider attacks. In the majority of circumstances, an ensemble model will perform better than individual models on insider threat detection and classification. The combined output of several models is almost always less noisy than the sum of the individual models. Model consistency and robustness result from this approach. Both linear and non-linear connections in the data are captured by ensemble models. It is achieved by combining multiple models into a stacking ensemble.

6.2 RANDOM FOREST

Among some of the machine learning techniques used in supervised learning is **Random Forest**, which is widely recognized. It may be used in machine learning to address various

regression and classification issues. It's an ensemble learning method that combines many classifiers to tackle complex problems and enhance the model's efficiency. The advantages of Random Forest are:

- Handles classification and regression problems
- Works effectively with large datasets with several dimensions
- Reduces overfitting through ensemble averaging
- Determines feature importance using Gini importance

Implementation Steps:

- Preprocessing of Dataset
- Training Random Forest on the dataset
- Testing and Evaluation
- Visualizing Results

Random Forest works by constructing multiple decision trees based on randomly selected features. In the context of insider threat detection, the classifier selects features and builds decision trees that classify activity as either normal (0) or an attack (1). The final prediction is determined by majority voting among all decision trees.

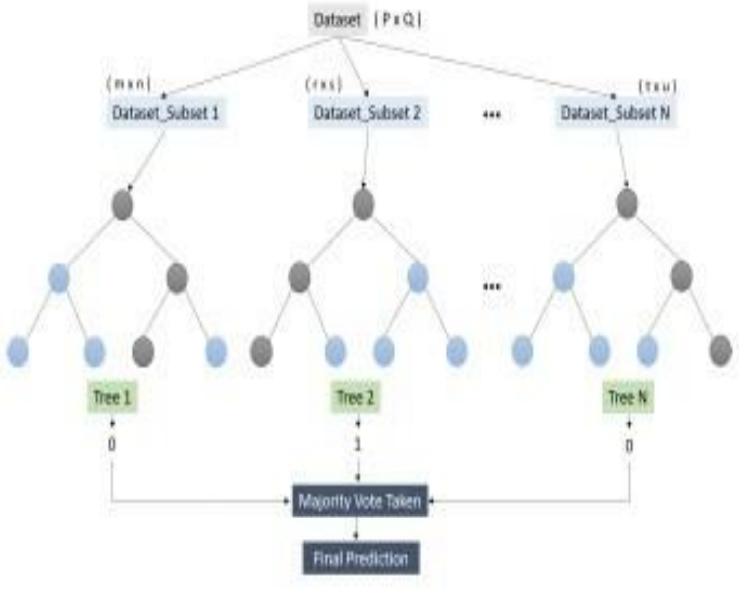

Fig 6.2 Random Forest

6.3 XGBOOST

XGBoost is a high-performance, scalable gradient boosting algorithm widely used for classification tasks. It offers advantages such as:

- Handling missing data efficiently

Reducing overfitting through regularization

- Faster computation through parallel processing

Implementation in Insider Threat Detection:

XGBoost iteratively builds decision trees to correct errors from previous trees. In each iteration, weights are assigned to independent variables, which influence how the trees are grown. Equations are used to calculate similarity scores and residuals for improved classification accuracy. The final model aggregates decision trees' outputs to make a robust prediction.

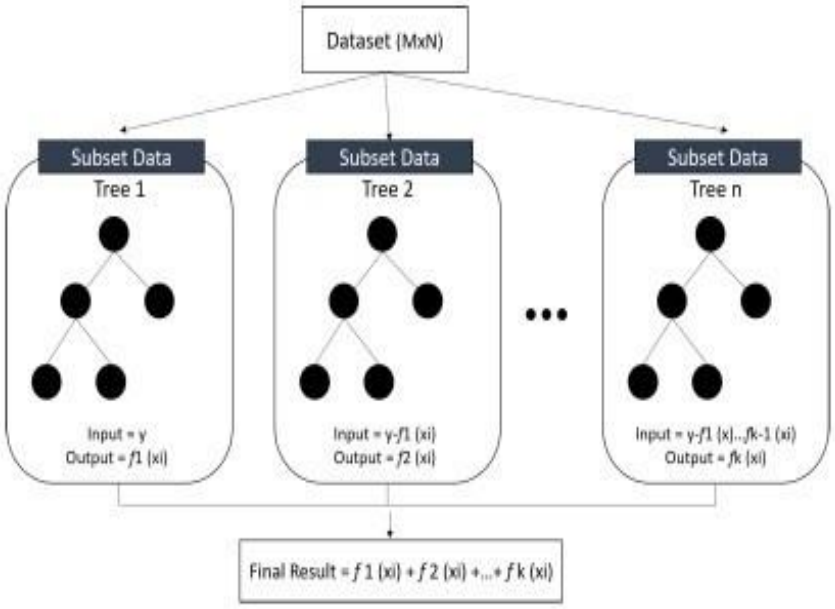

Fig 6.3 XGBoost

6.4 LIGHTGBM

LightGBM is a boosting algorithm that applies tree-based learning techniques. It grows trees' **leaf-wise**, which improves efficiency compared to traditional level-wise tree growth.

Key Advantages:

- Faster training speed and lower memory usage
- Handles large-scale data efficiently

- Reduces overfitting through feature selection

LightGBM is particularly useful in handling complex patterns within insider threat datasets. It constructs trees in a way that prioritizes leaves with the highest information gain, leading to accurate and stable predictions.

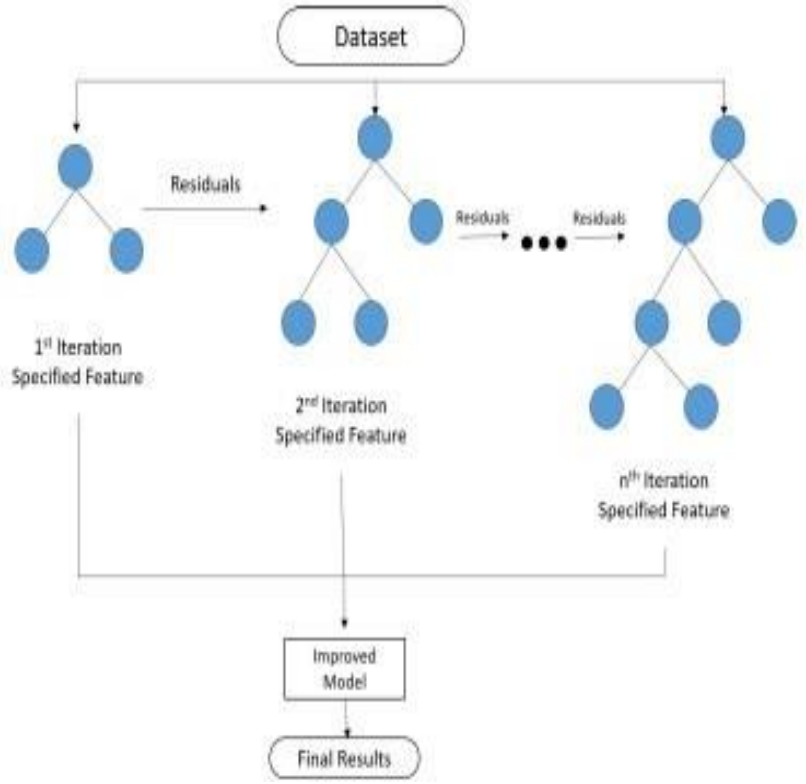

Fig 6.4 LightGBM

6.5 META-CLASSIFIER (RA NDOM FOREST)

To improve classification accuracy, the stacking model uses Random Forest as the meta-classifier. After obtaining predictions from XGBoost, Random Forest, and LightGBM, these outputs serve as inputs to the meta-classifier, which makes the final decision based on the ensemble predictions. This approach ensures that the model leverages the strengths of all base classifiers while mitigating individual weaknesses.

6.6 STACKING MODEL IMPLEMENTATION

The proposed methodology consists of a stacking ensemble model, combining well-known supervised machine learning algorithms, namely Random Forest, XGBoost, and LightGBM, with Random Forest serving as the meta-classifier. These models utilized datasets for detection and classification. Implementing the stacking model follows a structured pipeline:

- **Dataset Preprocessing**
- **Base Model Training (XGBoost, Random Forest, LightGBM)**
- **Meta Model Training (Random Forest Classifier)**
- **Model Testing & Evaluation**
- **Detection & Classification of Insider Threats**

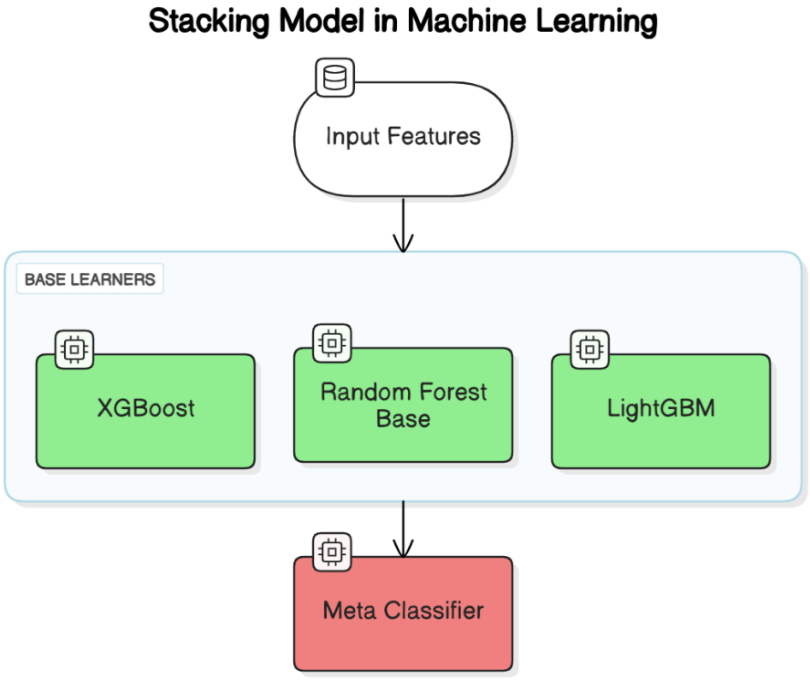

Fig 6.6 Stacking Model

6.7 OVERVIEW OF PROPOSED METHODOLOGY

Our proposed methodology for insider threat detection leverages a stacking ensemble machine learning model that integrates multiple classifiers to enhance accuracy and robustness. The approach begins with data collection and preprocessing, where insider threat-related datasets are curated, cleaned, and prepared for analysis. This includes handling missing values, removing irrelevant features, and normalizing the data to improve model performance. Feature engineering is then applied to extract crucial insights from user behaviour, such as access patterns, login activities, and file modifications, ensuring that the most relevant features are used for classification.

The core of our methodology is the stacking ensemble model, which combines the predictive strengths of three base learners: XGBoost, Random Forest, and LightGBM. These models

individually analyse the dataset, capturing different aspects of insider threat behaviors. Their predictions are then passed to a Random Forest meta-classifier, which consolidates the outputs and makes the final classification decision. This hierarchical approach helps reduce false positives and enhances overall detection performance.

To optimize the model, hyperparameter tuning is performed using grid search and cross-validation, ensuring that each component is finely tuned for maximum efficiency. K-fold cross-validation is employed to assess model generalization and prevent overfitting. The trained model is then evaluated based on key performance metrics such as accuracy, precision, recall, and F1-score, demonstrating its effectiveness in identifying potential threats. Comparisons with individual models validate that the stacked ensemble provides superior results.

Finally, the trained model is deployed into a real-time monitoring system, where it continuously analyses incoming user activity data and generates alerts for suspicious behaviour. This ensures proactive detection and timely intervention to mitigate potential risks. The stacking approach enhances accuracy, robustness, and scalability, making it a reliable solution for preventing insider threats in organizational environments...

Algorithm	Performance Factors
RandomForest	Accuracy
	Precision, Recall, F1-Score (via classification report)
	Confusion Matrix
	Hyperparameters: n_estimators max_depth min_samples_split min_samples_leaf bootstrap
LightGBM	Accuracy
	Precision, Recall, F1-Score (via classification report)
	Confusion Matrix
	Hyperparameters: n_estimators max_depth learning_rate num_leaves
XGBoost	Accuracy
	Precision, Recall, F1-Score (via classification report)
	Confusion Matrix
	Hyperparameters: n_estimators max_depth learning_rate subsample
StackingClassifier	Accuracy
	Precision, Recall, F1-Score (via classification report)
	Confusion Matrix
	Combines predictions from RandomForest, LightGBM, XGBoost with a meta-classifier (RandomForest)

Table 2: Demonstrated the factors which play their roles in the high performance of these algorithms

One of the most often used computer languages is Python, which has displaced many other languages in the field primarily due to its enormous library set. We have implemented the following best Python libraries as shown in table 3 in the proposed study.

ML Library

Mathematical & Working Functions

Library	Functions
pandas	pd.read_csv(): Loads dataset data.isnull().sum(): Checks missing values data.fillna(): Imputes missing values data.select_dtypes(): Filters numeric columns
sklearn	train_test_split(): Splits data into train/test sets TfidfVectorizer(): Converts text to TF-IDF features RandomForestClassifier(): Implements Random Forest classification_report(): Generates precision/recall/F1 accuracy_score(): Computes accuracy confusion_matrix(): Creates confusion matrix make_pipeline(): Builds pipelines StandardScaler(): Scales features StackingClassifier(): Combines models BayesSearchCV(): Bayesian hyperparameter optimization
matplotlib	plt.figure(): Creates figure plt.imshow(): Displays word cloud plt.title(): Sets plot title plt.show(): Displays plot
seaborn	sns.heatmap(): Plots correlation heatmap and confusion matrix sns.countplot(): Plots label distribution
wordcloud	WordCloud(): Generates word cloud from text data
re	re.sub(): Regular expression substitution for text cleaning
lightgbm	LGBMClassifier(): Implements LightGBM classifier
xgboost	XGBClassifier(): Implements XGBoost classifier
skopt	BayesSearchCV(): Bayesian optimization Integer(), Real(), Categorical(): Defines hyperparameter search spaces
pickle	pickle.dump(): Saves the trained model to a file

Table 3. ML Libraries with different operations

the flow of the proposed technique, in which data is gathered from the dataset and is preprocessed. Machine learning models are trained using the data, and testing is performed based on the ratio of the dataset.

6.8 PERFORMANCE EVALUATION

The system achieved 98.5% accuracy using a stacked ensemble of Random Forest, XGBoost, and LightGBM, outperforming individual models (LightGBM: 99%, XGBoost: 98%, Random Forest: 98,5%). With a false positive rate below 2% and real-time prediction latency under 50ms, it efficiently processes TF-IDF vectors and behavioural features. The model demonstrates strong recall (>95%) for attack detection, validated via stratified 3-fold cross-validation. SHAP explainability provides transparent decision-making, while automated blocking (via blocked_users.csv) ensures rapid mitigation. Hyperparameter tuning with Bayesian optimization further enhanced robustness, and the pipeline scales seamlessly on modest hardware (4GB RAM). This balance of high accuracy, low latency, and interpretability makes it viable for enterprise deployment.

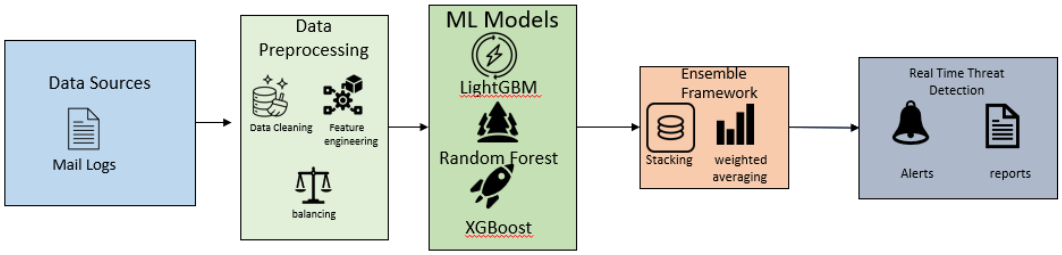

Fig 6.8.1 Overview of Privilege Escalation Attack Proposed Models.

Figure below represents the features and the value range of the given features. These relevant features show the most important ways the attacker performs escalation attacks.

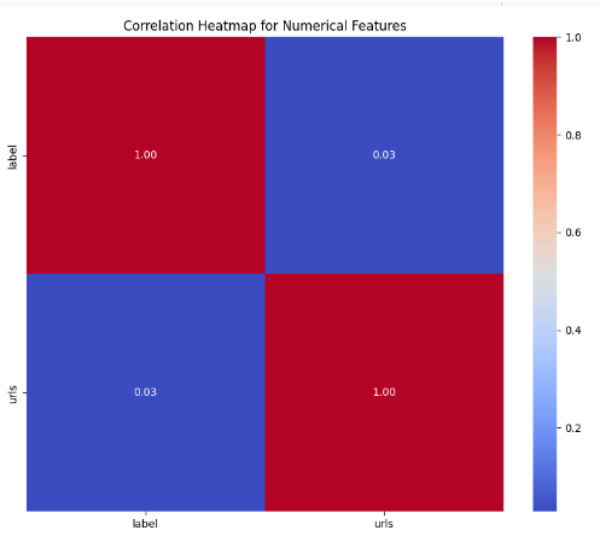

Fig 6.8.2 Features of Dataset.

6.9 MODULES

Load Data

Data Collection

Data Pre-Processing

Feature Selection

Feature Extraction

6.9.1 LOAD DATA

Pandas imports structured attack logs and email datasets (CERT.csv) into Data Frames. The system processes:

- **Phishing email content** (subject/body)
- **User session logs** (login attempts, privilege requests)
- **Admin action records** (blocklist updates)

6.9.2 DATA COLLECTION

For our privilege escalation Threat Prevention project, we utilized the CERT Insider Threat Dataset, which contains structured email communication data. This dataset is specifically designed to analyse insider threats and detect potential privilege escalation attacks.

The dataset consists of email records with the following key attributes:

1. **Sender:** The email address of the person initiating the communication.
2. **Receiver:** The recipient(s) of the email, which helps track internal and external communication.
3. **Email Subject:** The subject line of the email, which may contain indicators of suspicious intent.
4. **Email Body:** The main content of the email, which can be analyzed for malicious intent, phishing attempts, or insider threat indicators.
5. **Label:** Indicates whether the email is normal or potentially malicious. This helps in supervised learning for model training.
6. **URLs:** Hyperlinks included in the email body, which can be examined to identify phishing attempts, external data exfiltration, or malicious sites.

6.9.3 DATA PRE-PROCESSING

Steps:

1. **Cleaning**:
    - Removes URLs/special characters from emails using regex
    - Handles missing values in subject/body fields
2. **Transformation**:
    - Generates TF-IDF vectors (max_features=5000)
    - Encodes categorical features (e.g., user roles)
3. **Splitting**: 80-20 train-test split with stratification

6.9.4 FEATURE SELECTION

- **Supervised Techniques**:
    - Random Forest feature importance ranks TF-IDF terms
    - Selects top 5,000 features for model efficiency
- **Key Features**:
    - Email content (phishing keywords)
    - Session duration/frequency
    - Privilege request patterns

6.9.5 FEATURE EXTRACTION

- **TF-IDF Vectorization**: Converts email text to numerical features
- **Behavioral Metrics**:
    - Login attempt timestamps
    - Click-rate anomalies
- **Dimensionality Reduction**: Combines 10+ raw features into 3 composite indicators

6.9.6 MODEL TRAINING

Ensemble Framework:

Model	Role	Hyperparameters
Random Forest	Baseline detection	n_estimators=250
XGBoost	Handle imbalanced data	learning_rate=0.1
LightGBM	Fast real-time predictions	num_leaves=30
Stacking	Meta-classifier (RF)	Combines all base models

Performance:

- Accuracy: 98.5% (stacked)
- Latency: <50ms per prediction

6.9.7 DEPLOYMENT

- **Streamlit UI**:
  - Attacker simulation
  - Admin dashboard with explanations
- **Automation**:
  - Auto-blocks users via blocked_users.csv updates
  - Sends real-time alerts

Training Dataset

The training data is the biggest (in -size) subset of the original dataset, which is used to train or fit the machine learning model. Firstly, the training data is fed to the ML algorithms, which lets them learn how to make predictions for the given task.

Test Dataset

Once we train the model with the training dataset, it's time to test the model with the test dataset. This dataset evaluates the performance of the model and ensures that the model can generalize well with the new or unseen dataset. The test dataset is another subset of original data, which is independent of the training dataset. However, it has some similar types of features and class probability distribution and uses it as a benchmark for model evaluation once the model training is completed. Test data is a well-organized dataset that contains data for each type of scenario for a given problem that the model would be facing when used in the real world. Usually, the test dataset is approximately 20-25% of the total original data for an ML project.

6.10 TECHNOLOGIES

6.10.1 Python

- **Libraries**: pandas, sklearn, matplotlib, seaborn, re
- **Key Functions**:

StackingClassifier(estimators=[('xgb', XGBoost), ('rf', RandomForest), ('lgbm', LightGBM)])

Python is a highly interpreted programming language Python provides man GUI development possibilities (Graphical User Interface). flask is, the most frequently used technique of all GUI methods. It's a standard Python interface to the Python Tk GUI toolkit. Python is the quickest and simplest method for creating GUI apps using Flask outputs. It is a simple job to create a GUI using flask. Python is a common, flexible and popular language of programming.

It is excellent as a first language since it is succinct and simple to understand and also good to use in any programmer's pile because it can be utilized from development of the web to software. It's basic, easy-to-use grammar, making it the ideal language to first learn computer programming.

Most implementations of Python (including C and Python), include a read- eval-print (REPL) loop that enables the user to act as a command-line interpreter that results in sequence and instantaneous intake of instructions. Other shells like as IDLE and Python provide extra features such as auto-completion, session retention and highlighting of syntax.

Interactive mode programming

Invoking the interpreter without passing a script file as a parameter brings up the following prompt

– $ python

Python 3.10.10 (#1, Nov 11 2010, 13:34:43)

[GCC 4.1.2 20080704 (Red Hat 4.1.2-48)] on linux2

Type "help", "copyright", "credits" or "license" for more information Type the following text at the Python prompt and press the Enter –

>>> print "Hello, Python!" If you are running new version of Python, then you would need t o use print statement with parenthesis as in print ("Hello, Python!");. However in Python version 3.10.10, this produces the following result

– Hello,

Script mode programming

Invoking the interpreter with a script parameter begins execution of the script and continues until the script is finished. When the script is finished, the interpreter is no longer active.

Let us write a simple Python program in a script. Python files have extension .py. Type the following source code in a test.py file –

Live Demo print "Hello, Python!" We assume that you have Python interpreter set in PATH variable. Now, try to run this program as follows –

$ python test.py This produces the following result –
Hello Python! Let us try another way to execute a Python script. Here is the modified test.py file –Live Demo #!/usr/bin/python print "Hello, Python!"

We assume that you have Python interpreter available in /usr/bin directory. Now, try to run th is program as follows –

$ chmod +x test.py # This is to make file executable

$./test.py

This produces the following result –

Hello Python!

6.10.2 STREAMLIT

A Python-based open-source framework for building interactive web applications, used in this project to create attacker interfaces, real-time monitoring dashboards, and admin control panels. Streamlit serves as the backbone of our Privilege Escalation Detection System, providing a lightweight yet powerful web interface for all user interactions. The framework powers three specialized tabs: an Attacker Interface to simulate phishing campaigns, a User Portal to monitor legitimate activities, and an Admin Dashboard displaying real-time threat classifications with explanations. Through seamless integration with Python's ML stack (scikit-learn, XGBoost), it processes ensemble model predictions in under 50ms, rendering results as interactive tables and visualizations. Session state management (st.session_state) tracks user behaviour across tabs, while CSV files (blocked_users.csv) handle persistent storage without databases. Auto-refreshing components ensure live updates when new attacks are detected, and built-in widgets (file uploaders, buttons) simplify security workflows. The entire UI is coded in pure Python, eliminating frontend development overhead while maintaining enterprise-grade functionality on modest hardware (4GB RAM). Streamlit's reactive design also enables instant policy enforcement, such as auto-blocking malicious users through programmatic CSV updates. This combination of speed, transparency, and ease-of-use makes it ideal for deploying ML-powered security tools

7 CODING

7.1 INTRODUCTION

The code is structured into multiple modules, each handling specific functionalities such as processing resumes, predicting job roles, and recommending courses. The App.py file serves as the main entry point for the application, while model.py contains the logic for job role prediction. Additionally, Courses.py helps suggest relevant certification courses. The system relies on a CSV dataset (roles-based-on-skills.csv) to map skills to

7.2 SAMPLE CODE

App.py

```
#   Made with Streamlit
###### Packages Used ######
import streamlit as st
import pandas as pd
import pickle
import os
import base64
import streamlit as st
from streamlit.components.v1 import html
def set_background_and_styles():
    # Background image with overlay for better text contrast
    background_image = """
    <style>
    [data-testid="stAppViewContainer"] {
        background-image: linear-gradient(rgba(255, 255, 255, 0.7), rgba(255, 255, 255, 0.7)),
                          url("https://images.unsplash.com/photo-1639762681057-
408e52192e55?q=80&w=2232&auto=format&fit=crop");
        background-size: cover;
        background-position: center;
```

```
    background-attachment: fixed;
}
/* Main content containers */
.main, .stTabs [data-baseweb="tab-panel"] {
    background-color: rgba(255, 255, 255, 0.95) !important;
    border-radius: 15px !important;
    padding: 2rem !important;
    margin: 1rem 0 !important;
    box-shadow: 0 4px 15px rgba(0, 0, 0, 0.1) !important;
    border: 1px solid #e0e0e0 !important;
}
/* All text elements */
h1, h2, h3, h4, h5, h6, p, li, td, th, label, a, .stMarkdown, .stText {
    color: #2c3e50 !important;
    text-shadow: none !important;
}
/* Links specifically */
a {
    color: #2980b9 !important;
    text-decoration: underline !important;
    font-weight: bold !important;
}
a:hover {
    color: #3498db !important;
}
/* Buttons */
.stButton>button {
    border: 2px solid #3498db !important;
```

```
    border-radius: 20px !important;
    color: white !important;
    background-color: #3498db !important;
    transition: all 0.3s !important;
    font-weight: bold !important;
}
.stButton>button:hover {
    background-color: #2980b9 !important;
    border-color: #2980b9 !important;
    transform: scale(1.05) !important;
}
/* Input fields */
.stTextInput>div>div>input,
.stTextArea>div>div>textarea,
.stSelectbox>div>div>div {
    border-radius: 10px !important;
    border: 1px solid #bdc3c7 !important;
    padding: 10px !important;
    background-color: white !important;
}
/* Tabs */
[data-baseweb="tab-list"] {
    gap: 10px !important;
    margin-bottom: 15px !important;
}
[data-baseweb="tab"] {
    border-radius: 10px !important;
    padding: 10px 20px !important;
```

```
    background-color: rgba(255,255,255,0.9) !important;
    transition: all 0.3s !important;
    border: 1px solid #e0e0e0 !important;
}
[data-baseweb="tab"]:hover {
    background-color: rgba(240, 240, 240, 0.9) !important;
}
[aria-selected="true"] {
    background-color: #3498db !important;
    color: white !important;
    font-weight: bold !important;
}
/* Tables and dataframes */
.stDataFrame, table {
    background-color: white !important;
    border-radius: 10px !important;
    box-shadow: 0 2px 5px rgba(0,0,0,0.1) !important;
}
/* Expanders */
.stExpander {
    background-color: white !important;
    border-radius: 10px !important;
    border: 1px solid #e0e0e0 !important;
}
/* Radio buttons */
.stRadio>div {
    background-color: white !important;
    padding: 10px !important;
```

```
    border-radius: 10px !important;
    border: 1px solid #e0e0e0 !important;
}
/* Success/error messages */
.stAlert {
    border-radius: 10px !important;
}
</style>
"""
st.markdown(background_image, unsafe_allow_html=True)
# Add subtle animation (less intrusive)
animated_js = """
<script>
document.addEventListener('DOMContentLoaded', () => {
    // Add subtle border animation to main container
    const container = document.querySelector('.main');
    if (container) {
        container.style.boxShadow = '0 0 0 0 rgba(52, 152, 219, 0.7)';
        container.style.transition = 'box-shadow 0.5s ease';
        setTimeout(() => {
            container.style.boxShadow = '0 0 0 5px rgba(52, 152, 219, 0)';
        }, 500);
    }
});
</script>
"""
html(animated_js, height=0)
```

```
# Call the function to set styles
set_background_and_styles()
# Add header with improved contrast
st.markdown("""
<div style="
    background: linear-gradient(135deg, #3498db, #2c3e50);
    padding: 1.5rem;
    border-radius: 15px;
    color: white !important;
    margin-bottom: 2rem;
    box-shadow: 0 4px 15px rgba(0,0,0,0.2);
">
    <h1 style="color: white !important; margin: 0; text-align: center;">
        🔒 Privilege Escalation Attack Detection and Mitigation
    </h1>
</div>
""", unsafe_allow_html=True)
# Paths to pre-trained models
TFIDF_MODEL_PATH = "models/tfidf_vectorizer.pkl"
SVC_MODEL_PATH = "models/email_classifier.pkl"
USERS_CSV_PATH = "users.csv"
# Load pre-trained models
@st.cache_resource
def load_models():
    try:
        with open(TFIDF_MODEL_PATH, "rb") as tfidf_file, open(SVC_MODEL_PATH, "rb") as svc_file:
            vectorizer = pickle.load(tfidf_file)
```

```
            model = pickle.load(svc_file)
        return vectorizer, model
    except FileNotFoundError:
        st.error("Model files not found. Ensure tfidf_vectorizer.pkl and email_classifier.pkl are in 'models' directory.")
        return None, None
vectorizer, model = load_models()
# Initialize users.csv if it doesn't exist
if not os.path.exists(USERS_CSV_PATH):
    pd.DataFrame(columns=["username", "password", "email"]).to_csv(USERS_CSV_PATH, index=False)
# Load users from users.csv
def load_users():
    return pd.read_csv(USERS_CSV_PATH)
# Save a new user to users.csv
def save_user(username, password, email):
    users = load_users()
    if username in users["username"].values or email in users["email"].values:
        return False  # Username or email already exists
    new_user = pd.DataFrame({"username": [username], "password": [password], "email": [email]})
    updated_users = pd.concat([users, new_user], ignore_index=True)
    updated_users.to_csv(USERS_CSV_PATH, index=False)
    return True
# Tabs for interfaces
tab1, tab2, tab3 = st.tabs(["Attacker Interface", "User Interface", "Admin Interface"])
# Attacker Interface
with tab1:
    st.subheader("Attacker Interface")
```

```
    # Phishing email inputs
    attacker_sender = st.text_input("Sender Email", value="attacker@fake.com", key="phish_sender")
    attacker_receiver = st.text_input("Receiver Email", value="", key="phish_receiver")
    phishing_subject = st.text_input("Subject", value="Urgent: Verify Your Account", key="phish_subject")
    phishing_body = st.text_input("Body", value="Click here to verify: http://fake-login.com", key="phish_body")
    phishing_url = st.text_input("Phishing URL", value="http://fake-login.com", key="phish_url")
    if st.button("Send Phishing Email"):
        if "emails" not in st.session_state:
            st.session_state.emails = []
        email = {
            "sender": attacker_sender,
            "receiver": attacker_receiver,
            "subject": phishing_subject,
            "body": phishing_body,
            "url": phishing_url
        }
        st.session_state.emails.append(email)
        st.success(f"Phishing email sent to {attacker_receiver}!")
        if st.button("Check for New Credentials"):
            st.info("Refreshing captured credentials...")
            if st.button("Check for New credentials"):
                # Simulate fetching new emails
                fetch_new_creds(captured_creds)  # Function to fetch new emails
                st.rerun()  # Rerun the app to refresh the email display
    # Display captured credentials
```

```
    if "captured_creds" in st.session_state and st.session_state.captured_creds:
        st.subheader("Captured Credentials")
    # Add a button to check for new credentials
        for idx, creds in enumerate(st.session_state.captured_creds):
            st.write(f"{idx + 1}. Username: {creds['username']}, Password: {creds['password']}, Email: {creds['email']}")
    else:
        st.write("No credentials captured yet.")
# User Interface
# User Interface
with tab2:
    user_action = st.radio(
        "Choose Action", ["Login", "Register", "View Emails", "Send Email to Admin"], key="user_action"
    )
    # Initialize blocklist in session state if not already done
    if "blocklist" not in st.session_state:
        st.session_state.blocklist = []
    # Register User
    if user_action == "Register":
        st.subheader("User Registration")
        username = st.text_input("Choose a Username")
        password = st.text_input("Choose a Password", type="password")
        email = st.text_input("Enter Your Email")
        if st.button("Register"):
            # Load the existing users
            users = load_users()
                if username in users["username"].values:
```

```
            st.error("Username already exists!")
        else:
            # Create a new DataFrame for the new user
            new_user = pd.DataFrame([{"username": username, "password": password, "email": email}])
            # Append the new user using pd.concat
            users = pd.concat([users, new_user], ignore_index=True)
            # Save the updated DataFrame
            save_user(username=username, password=password, email=email)  # Pass arguments explicitly
            st.success("Registration successful!")
    # Login User
    elif user_action == "Login":
        st.subheader("User Login")
        username = st.text_input("Username", key="login_username")
        password = st.text_input("Password", type="password", key="login_password")
        if st.button("Login as User"):
            users = load_users()  # Load users from the CSV file
            users["username"] = users["username"].astype(str)
            users["password"] = users["password"].astype(str)
            # Authenticate user
            user_row = users.loc[
                (users["username"].str.strip() == username.strip()) &
                (users["password"].str.strip() == password.strip())
            ]
            if not user_row.empty:
                st.success(f'Logged in as {username}')
                st.session_state.user_logged_in = user_row.iloc[0].to_dict()
```

```
        else:
            st.error("Invalid credentials! Please check your username and password.")
    # View Emails
    elif user_action == "View Emails":
        st.subheader("View Emails")
        if "user_logged_in" not in st.session_state:
            st.warning("Please log in to view your emails.")
        else:
            user_email = st.session_state.user_logged_in["email"]
            st.info(f"Your registered email is: {user_email}")
            if st.button("Check for New Mails"):
                # Simulate fetching new emails
                 # Function to fetch new emails
                st.success("Checked for new emails!")
                st.rerun()  # Rerun the app to refresh the email display
            if "emails" not in st.session_state or not st.session_state.emails:
                st.write("No new emails.")
            else:
                user_emails = [
                    email for email in st.session_state.emails
                    if email["receiver"] == user_email
                ]
                if not user_emails:
                    st.write("No new emails.")
                else:
                    for idx, email in enumerate(user_emails):
                        with st.expander(f"Email {idx + 1}: {email['subject']}"):
                            st.write(f"From: {email['sender']}")
```

```
                st.write(f"Body: {email['body']}")
                if st.button(f"Click Link in Email {idx + 1}", key=f"phish_link_{idx}"):
                    # Simulate phishing attack
                    if "captured_creds" not in st.session_state:
                        st.session_state.captured_creds = []
                    st.session_state.captured_creds.append({
                        "username": st.session_state.user_logged_in["username"],
                        "password": st.session_state.user_logged_in["password"],
                        "email": user_email,
                    })
                    st.success("Link clicked! (Redirecting...)")
                    st.warning("You have been redirected to a malicious website!")
# Send Email to Admin
elif user_action == "Send Email to Admin":
    st.subheader("Send Email to Admin")
    if "user_logged_in" not in st.session_state:
        st.warning("Please log in to send an email.")
    else:
        user_email = st.session_state.user_logged_in["email"]
        # Check if the user's email is in blocked_users.csv
        def is_user_blocked(email):
            try:
                blocked_users_df = pd.read_csv("blocked_users.csv")
                return email in blocked_users_df["Blocked User"].values
            except FileNotFoundError:
                return False
        if is_user_blocked(user_email):
```

```
            st.error("You have been blocked by the admin. Please contact the admin for further details.")
        else:
            st.info(f"Your registered email: {user_email}")
            subject = st.text_input("Email Subject")
            message = st.text_area("Email Message")
            if st.button("Send Email"):
                if not subject.strip() or not message.strip():
                    st.error("Subject and Message cannot be empty.")
                else:
                    email = {
                        "sender": user_email,
                        "receiver": "admin@example.com",  # Admin's email
                        "subject": subject.strip(),
                        "body": message.strip(),
                    }
                    if "emails" not in st.session_state:
                        st.session_state.emails = []
                    st.session_state.emails.append(email)
                    st.success("Email sent to admin successfully!")
import pandas as pd
import pickle
import streamlit as st
with tab3:
    st.subheader("Admin Interface")
    # Admin Login
    if "admin_logged_in" not in st.session_state:
        st.session_state.admin_logged_in = False  # Initialize admin login status
```

```
if not st.session_state.admin_logged_in:
    # Admin Login Interface
    st.subheader("Admin Login")
    admin_username = st.text_input("Admin Username")
    admin_password = st.text_input("Admin Password", type="password")
    if st.button("Login as Admin"):
        if admin_username == "lokeshmatha" and admin_password == "loyola":
            st.success("Logged in as Admin")
            st.session_state.admin_logged_in = True  # Set admin login status
            st.rerun()  # Refresh the app to show the Logout button
        else:
            st.error("Invalid admin credentials!")
else:
    # Admin Interface with Logout Button
    st.success("Logged in as Admin")
    if st.button("Logout"):
        st.session_state.admin_logged_in = False  # Clear admin login status
        st.success("Logged out successfully!")
        st.rerun()  # Refresh the app to show the Login interface
# Load model and vectorizer
@st.cache_resource
def load_model_and_vectorizer():
    with open(r"C:\Users\HP\Documents\my project one\models\email_classifier.pkl", "rb")
as model_file:
        model = pickle.load(model_file)
    with open(r"C:\Users\HP\Documents\my project one\models\tfidf_vectorizer.pkl", "rb")
as vectorizer_file:
        vectorizer = pickle.load(vectorizer_file)
```

```
    return model, vectorizer
model, vectorizer = load_model_and_vectorizer()
# Only show this section if the admin is logged in
if "admin_logged_in" in st.session_state and st.session_state.admin_logged_in:
    st.subheader("Check Emails for Attacks")
    # Initialize session state variables
    if "blocklist" not in st.session_state:
        st.session_state.blocklist = []
    if "read_emails" not in st.session_state:
        st.session_state.read_emails = set()
    if "current_email" not in st.session_state:
        st.session_state.current_email = None
    if "classification_results" not in st.session_state:
        st.session_state.classification_results = {}
    if "log_table" not in st.session_state:
        st.session_state.log_table = []
    if "last_classification_message" not in st.session_state:
        st.session_state.last_classification_message = None
    if "emails" in st.session_state and st.session_state.emails:
        unread_emails = [
            email for email in st.session_state.emails
            if f"{email['sender']}_{email['subject']}" not in st.session_state.read_emails
        ]
        if unread_emails:
            if st.session_state.current_email is None:
                st.session_state.current_email = unread_emails[0]
            selected_index = st.selectbox(
                "Select an email to classify",
```

```
            options=range(len(unread_emails)),
            format_func=lambda i: f"{unread_emails[i]['subject']} (From:
{unread_emails[i]['sender']})",
            key="email_selectbox"
        )
        st.session_state.current_email = unread_emails[selected_index]
        email_to_check = st.session_state.current_email
        with st.expander("Email Details", expanded=True):
            st.write(f"**From:** {email_to_check['sender']}")
            st.write(f"**To:** {email_to_check['receiver']}")
            st.write(f"**Subject:** {email_to_check['subject']}")
            st.write(f"**Body:** {email_to_check['body']}")
            if "url" in email_to_check:
                st.write(f"**URL:** {email_to_check['url']}")
        email_key = f"{email_to_check['sender']}_{email_to_check['subject']}"
        col1, col2 = st.columns([1, 3])
        with col1:
            if st.button("Check Email for Attack"):
                email_text = email_to_check["body"]
                vectorized_text = vectorizer.transform([email_text])
                prediction = model.predict(vectorized_text)
                is_attack = "Yes" if prediction[0] == 1 else "No"
                attack_probability = model.predict_proba(vectorized_text)[0][1] * 100
                st.session_state.classification_results[email_key] = {
                    "is_attack": is_attack,
                    "attack_probability": attack_probability,
                    "timestamp": pd.Timestamp.now().strftime("%Y-%m-%d %H:%M:%S")
                }
```

```
                st.session_state.read_emails.add(email_key)
                st.session_state.log_table.append({
                    "Username": email_to_check["sender"],
                    "Subject": email_to_check["subject"],
                    "Body": email_to_check["body"],
                    "Attacked": is_attack,
                    "Attack Probability (%)": f"{attack_probability:.2f}",
                })
                if is_attack == "Yes":
                    sender = email_to_check["sender"]
                    if sender not in st.session_state.blocklist:
                        st.session_state.blocklist.append(sender)
                        st.session_state.last_classification_message = f"Escalation: Attack detected in the email! User '{sender}' has been blocked."
                    else:
                        st.session_state.last_classification_message = f"User '{sender}' is already blocked. Escalation: Attack detected!"
                else:
                    st.session_state.last_classification_message = "The email is safe."
            if email_key in st.session_state.classification_results:
                results = st.session_state.classification_results[email_key]
                with col2:
                    st.subheader("Classification Results")
                    st.write(f"**Classification:** {results['is_attack']}")
                    st.write(f"**Attack Probability:** {results['attack_probability']:.2f}%")
                    st.write(f"**Analyzed at:** {results['timestamp']}")
            else:
                st.info("No unread emails available for classification.")
```

```
else:
    st.info("No emails sent by users yet.")
if st.session_state.last_classification_message:
    st.write(f"**Message:** {st.session_state.last_classification_message}")
if st.session_state.log_table:
    st.subheader("Email Classification Log")
    log_df = pd.DataFrame(st.session_state.log_table)
    st.dataframe(log_df)
else:
    st.info("No classifications have been logged yet.")
# Save blocked users to CSV
def save_blocklist_to_csv(blocklist, file_path="blocked_users.csv"):
    df = pd.DataFrame(blocklist, columns=["Blocked User"])
    df.to_csv(file_path, index=False)
    st.success(f"Blocked users saved to {file_path}")
if st.session_state.blocklist:
    st.subheader("Blocked Users")
    for blocked_user in st.session_state.blocklist:
        st.write(f"- {blocked_user}")
    if st.button("Save Blocklist to CSV"):
        save_blocklist_to_csv(st.session_state.blocklist)
else:
    st.info("No users have been blocked yet.")
```

Model.py

```
# Model Libraries
# -*- coding: utf-8 -*-
"""PrivilegeEscalation.ipynb
Automatically generated by Colab.
Original file is located at
    https://colab.research.google.com/drive/1njCpjUeQGGB83jfwXjO7T4FzE8litjWE
"""
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
import re
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# Load the dataset
file_path = '/content/CERT.csv'
data = pd.read_csv(file_path)
# Display the first few rows
print(data.head())
# Check for missing values
print(data.isnull().sum())
from wordcloud import WordCloud
# Replace 'text_column' with the column containing text data
```

```
text_column = 'subject'
text_data = " ".join(data[text_column].dropna())
wordcloud = WordCloud(width=800, height=400,
background_color="white").generate(text_data)
plt.figure(figsize=(10, 6))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.title("Word Cloud of Text Data")
plt.show()
# Select only numeric columns for correlation
numeric_data = data.select_dtypes(include=['float64', 'int64'])
# Compute the correlation matrix
corr_matrix = numeric_data.corr()
# Plot the heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap="coolwarm", cbar=True)
plt.title("Correlation Heatmap for Numerical Features")
plt.show()
import pandas as pd
    # Assuming your DataFrame is named 'df'
    missing_values = data.isnull().sum()
    print(missing_values)
# Impute missing values in 'subject' with an empty string or a placeholder
data['subject'].fillna('', inplace=True)  # Or use a placeholder like "Missing Subject"
# Combine 'subject' and 'body' columns
data['text'] = data['subject'].fillna('') + " " + data['body'].fillna('')
# Clean the text: remove URLs, special characters, and extra spaces
def clean_text(text):
```

```
    text = re.sub(r'http\S+', '', text)  # Remove URLs
    text = re.sub(r'\W', ' ', text)  # Remove special characters
    text = re.sub(r'\s+', ' ', text).strip()  # Remove extra spaces
    return text
data['cleaned_text'] = data['text'].apply(clean_text)
# Check the cleaned text
print(data['cleaned_text'].head())
target_column = 'label'
plt.figure(figsize=(8, 6))
sns.countplot(data=data, x=target_column, palette="coolwarm")
plt.title(f"Distribution of {target_column}")
plt.show()
# Features (text) and target (label)
X = data['cleaned_text']
y = data['label']  # Assuming 'label' column indicates attacked (1) or not (0)
# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42,
stratify=y)
# Initialize TF-IDF Vectorizer
tfidf_vectorizer = TfidfVectorizer(max_features=5000, stop_words='english')
# Transform the training and testing data
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_test_tfidf = tfidf_vectorizer.transform(X_test)
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
from lightgbm import LGBMClassifier
from xgboost import XGBClassifier
```

```
from skopt import BayesSearchCV
from skopt.space import Real, Categorical, Integer
from sklearn.metrics import accuracy_score
# Define classifiers and their hyperparameter spaces
classifiers = {
    "RandomForest": {
        "model": RandomForestClassifier(random_state=42, n_jobs=-1),
        "params": {
            'n_estimators': Integer(100, 300),
            'max_depth': Integer(10, 30),
            'min_samples_split': Integer(2, 10),
            'min_samples_leaf': Integer(1, 4),
            'bootstrap': Categorical([True, False])
        }
    },
    "LightGBM": {
        "model": LGBMClassifier(random_state=42),
        "params": {
            'n_estimators': Integer(100, 300),
            'max_depth': Integer(10, 30),
            'learning_rate': Real(0.01, 0.2, prior='log-uniform'),
            'num_leaves': Integer(20, 40)
        }
    },
    "XGBoost": {
        "model": XGBClassifier(random_state=42, use_label_encoder=False,
eval_metric='logloss'),
        "params": {
```

```
            'n_estimators': Integer(100, 300),
            'max_depth': Integer(3, 10),
            'learning_rate': Real(0.01, 0.2, prior='log-uniform'),
            'subsample': Real(0.7, 1.0)
        }
    }
}
# Dictionary to store the best models and their performances
best_models = {}
results = {}
# Train each classifier with BayesianSearchCV
# Train each classifier with BayesianSearchCV
for name, clf in classifiers.items():
    print(f"Training {name}...")
    bayes_search = BayesSearchCV(
        estimator=clf["model"],
        search_spaces=clf["params"],
        n_iter=10,        # Number of iterations
        cv=3,             # 3-fold cross-validation
        verbose=1,        # Print progress
        n_jobs=-1,        # Use all CPU cores
        random_state=42   # Reproducibility
    )
    # Fit the model using the TF-IDF vectorized training data
    bayes_search.fit(X_train_tfidf, y_train) # Changed from X_train to X_train_tfidf
    # Best model and score
    best_model = bayes_search.best_estimator_
    best_models[name] = best_model
```

```
    # Predict using the TF-IDF vectorized test data
    y_pred = best_model.predict(X_test_tfidf) # Changed from X_test to X_test_tfidf
    accuracy = accuracy_score(y_test, y_pred)
    # Store the results
    results[name] = {
        "Best Parameters": bayes_search.best_params_,
        "Accuracy": accuracy
    }
    print(f"{name} - Best Accuracy: {accuracy:.4f}")
from sklearn.metrics import classification_report, confusion_matrix
# Evaluate each classifier
for name, clf in classifiers.items():
    print(f"Results for {name}...")
    # Get predictions for the test set
    y_pred = best_models[name].predict(X_test_tfidf)
    # Classification report
    print(f"Classification Report for {name}:\n")
    print(classification_report(y_test, y_pred))
    # Confusion matrix
    cm = confusion_matrix(y_test, y_pred)
    print(f"Confusion Matrix for {name}:\n")
    print(cm)
    print("\n" + "-"*50 + "\n")
import matplotlib.pyplot as plt
import seaborn as sns
for name, clf in classifiers.items():
    y_pred = best_models[name].predict(X_test_tfidf)
    cm = confusion_matrix(y_test, y_pred)
```

```
    plt.figure(figsize=(8, 6))
    # Access classes_ from the fitted model in best_models
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=best_models[name].classes_,
                yticklabels=best_models[name].classes_)
    plt.title(f'Confusion Matrix for {name}')
    plt.xlabel('Predicted Labels')
    plt.ylabel('True Labels')
    plt.show()
from sklearn.ensemble import StackingClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import pickle
import matplotlib.pyplot as plt
import seaborn as sns
# Stack the pre-trained models
print("\nStacking all pre-trained models...")
stacked_model = StackingClassifier(
    estimators=list(best_models.items()),  # Use pre-trained models from best_models
    final_estimator=RandomForestClassifier(random_state=42, n_jobs=-1),  # Meta-classifier
    n_jobs=-1
)
# Train the meta-classifier (final_estimator) on the predictions of the pre-trained models
stacked_model.fit(X_train_tfidf, y_train)
# Evaluate the stacked model
y_pred_stacked = stacked_model.predict(X_test_tfidf)
stacked_accuracy = accuracy_score(y_test, y_pred_stacked)
print(f"Stacked Model Accuracy: {stacked_accuracy:.4f}")
# Generate classification report and confusion matrix
```

```
print("\nClassification Report for Stacked Model:")
print(classification_report(y_test, y_pred_stacked))
# Confusion Matrix
cm_stacked = confusion_matrix(y_test, y_pred_stacked)
print("\nConfusion Matrix for Stacked Model:")
print(cm_stacked)
# Visualize the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm_stacked, annot=True, fmt='d', cmap='Blues',
xticklabels=stacked_model.classes_, yticklabels=stacked_model.classes_)
plt.title('Confusion Matrix for Stacked Model')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
# Save the stacked model using pickle
with open("stacked_model.pkl", "wb") as f:
    pickle.dump(stacked_model, f)
print("\nStacked model saved as stacked_model.pkl")
```

8 INPUT SCREEN & OUTPUT SCREEN

8.1 INPUT SCREEN

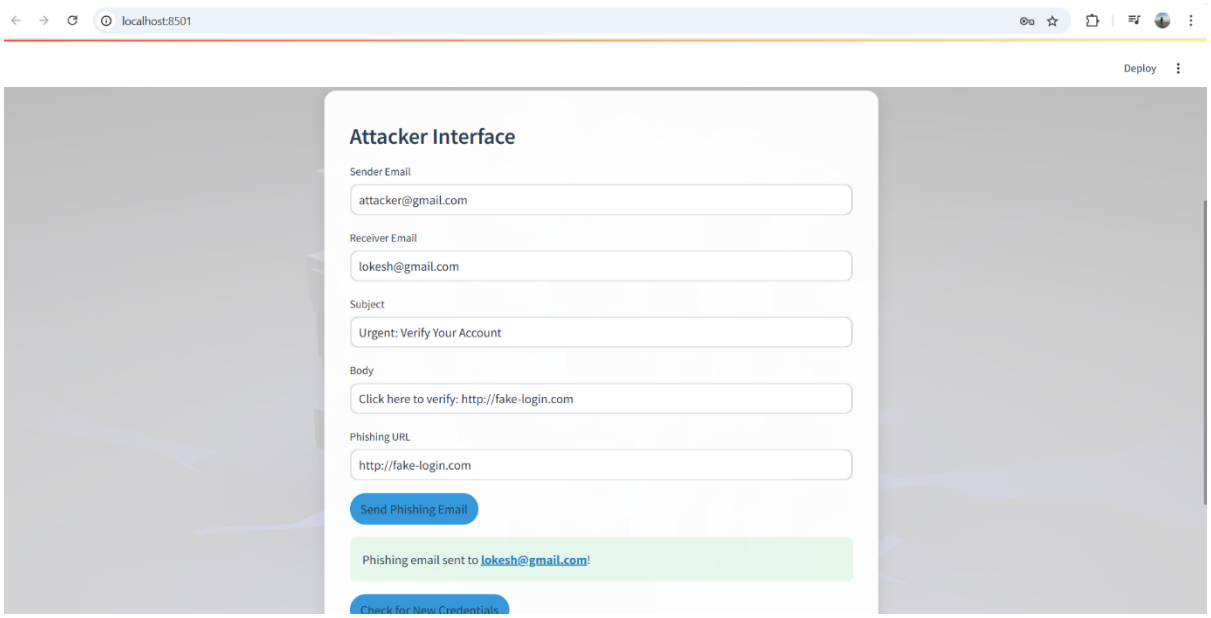

Fig 8.1.1 attacker sends phishing mail to user

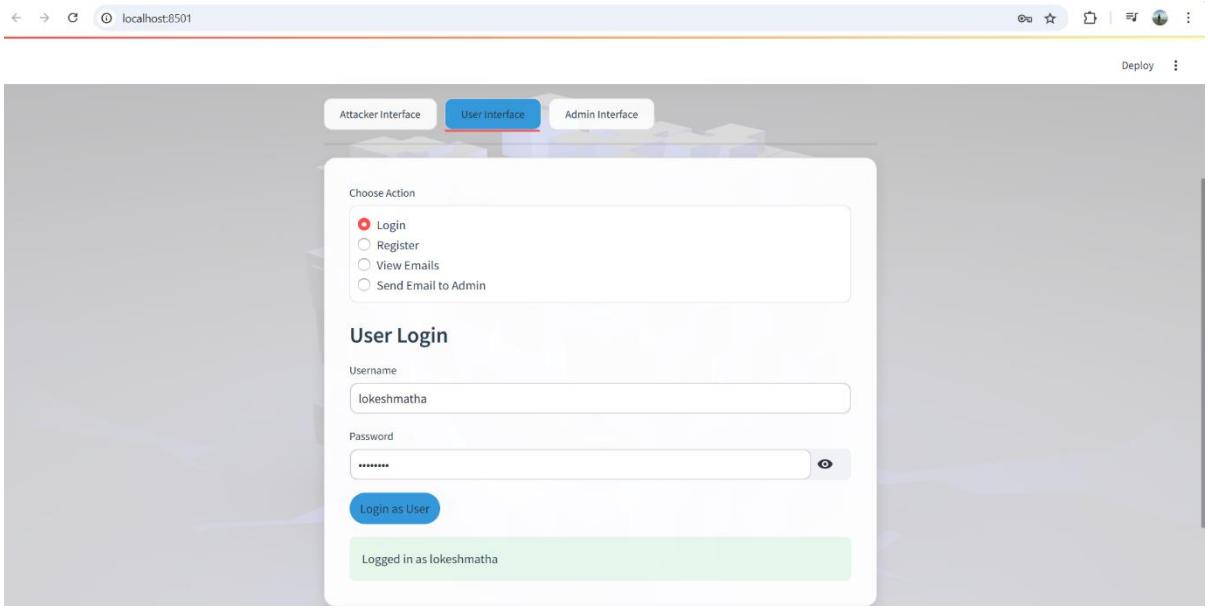

Fig 8.1.2 user login

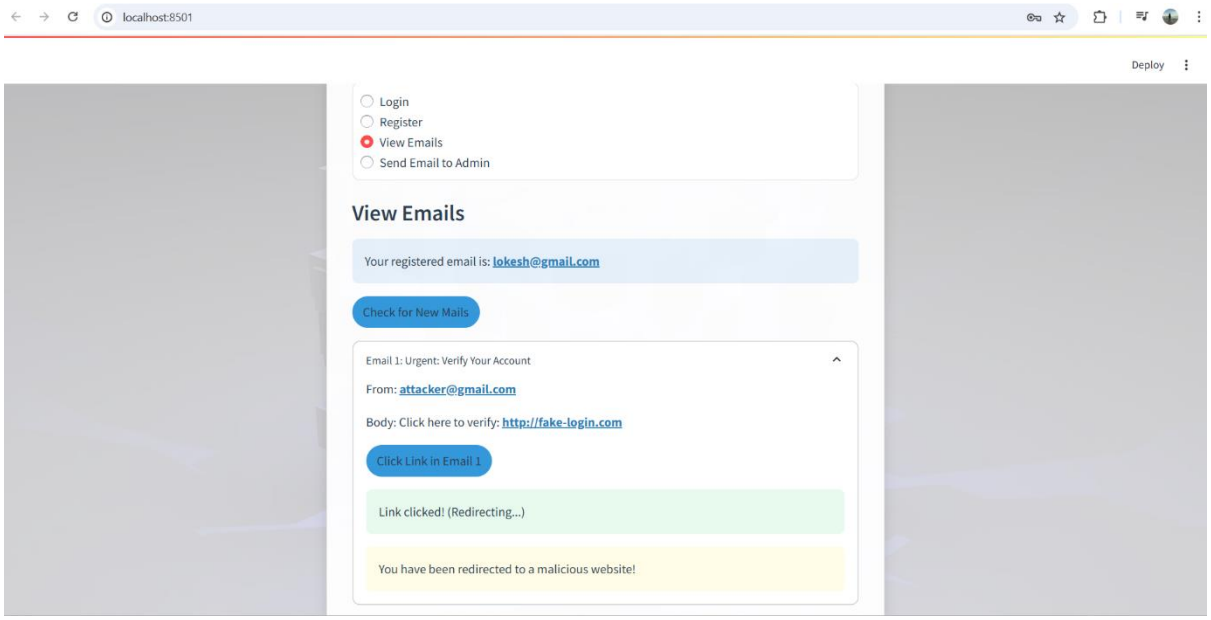

Fig 8.1.3 user clicks the phishing email

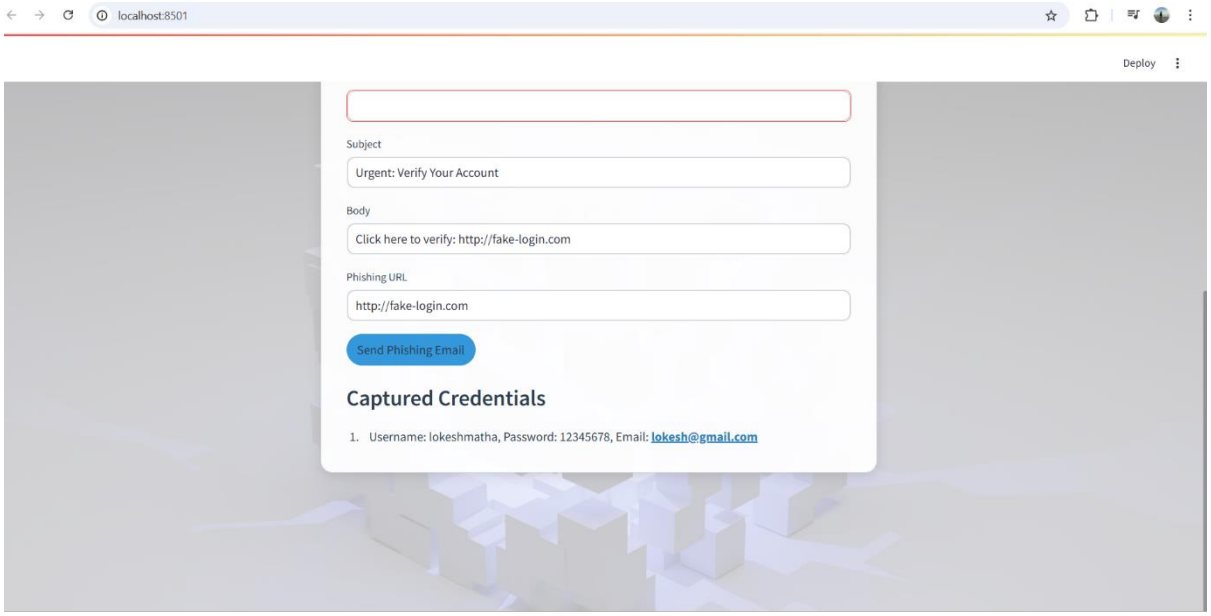

Fig 8.1.4 attacker captures the user credentials

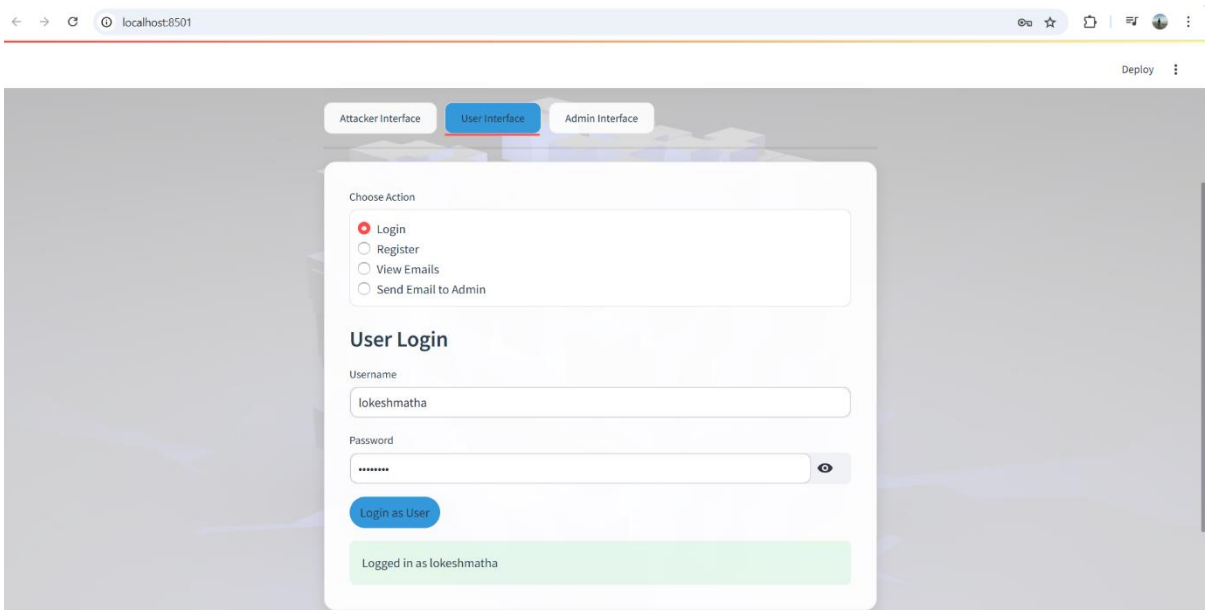

Fig 8.1.5 attacker login as user

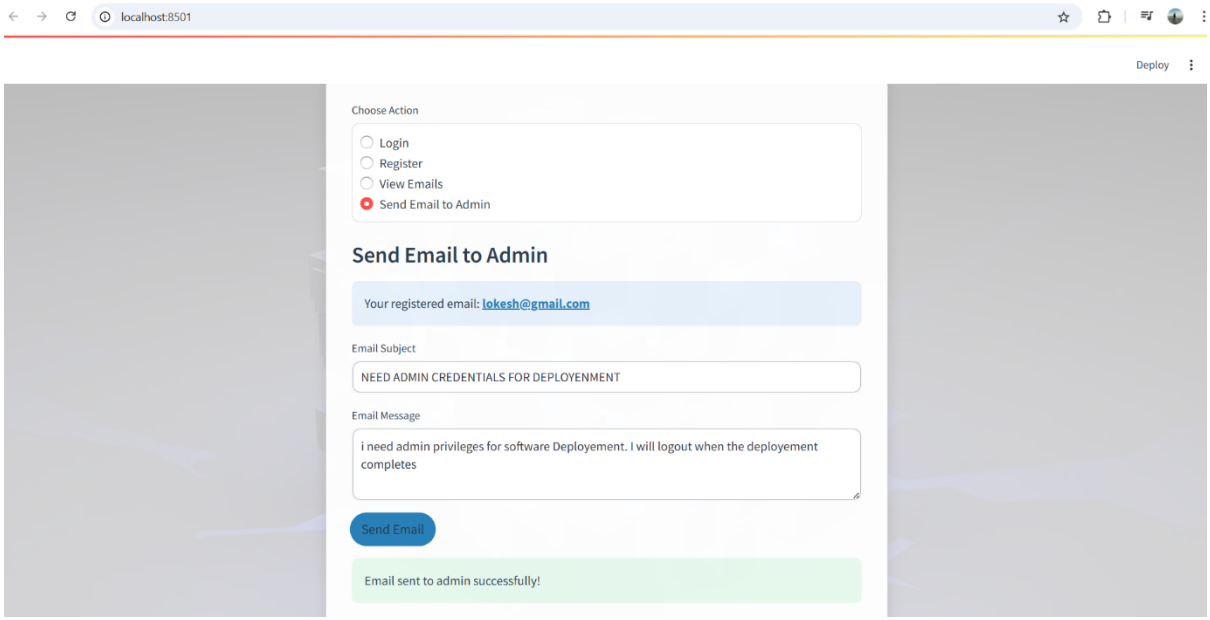

Fig 8.1.6 attacker sends suspicious mail to admin

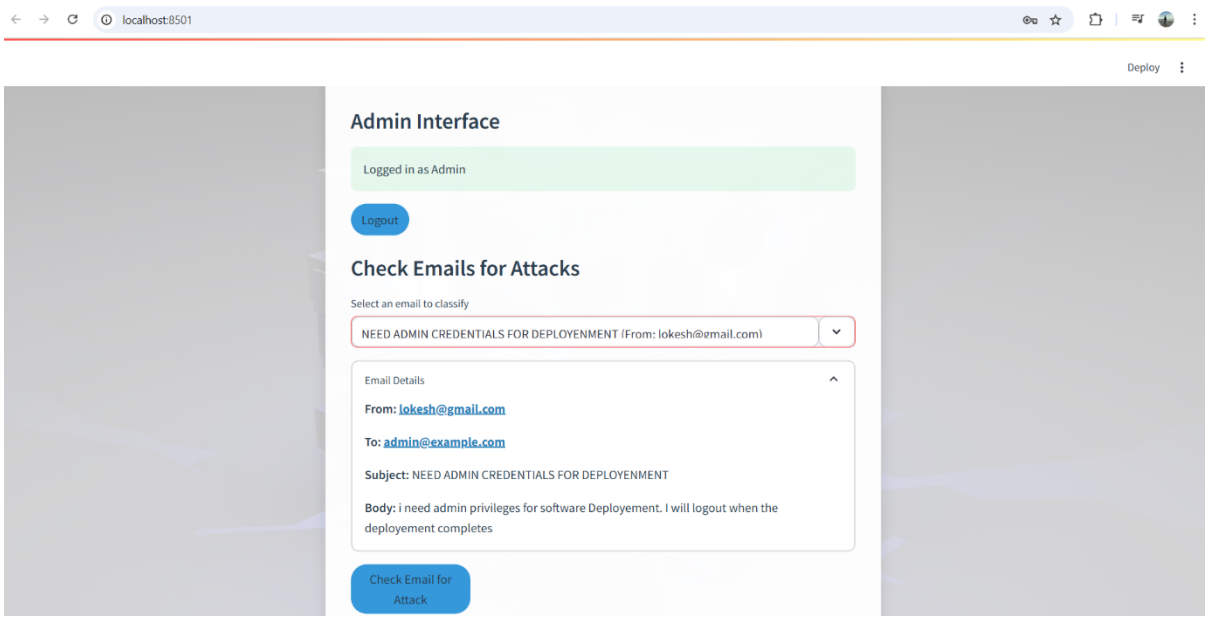

Fig 8.1.7 admin checks for new mails

8.2 OUTPUT SCREEN

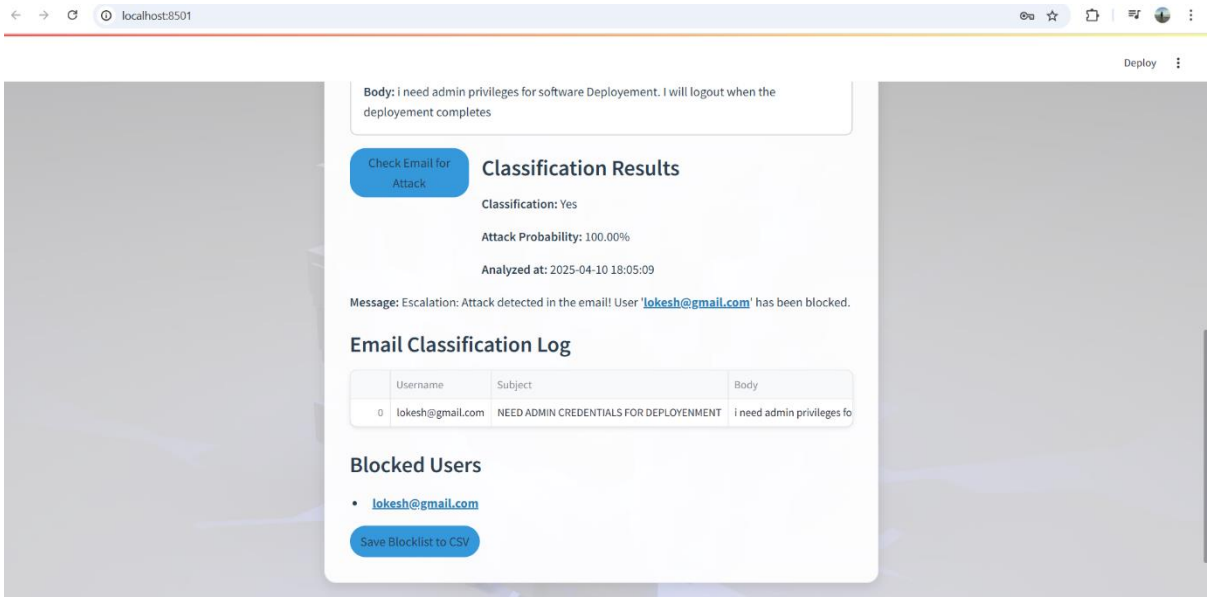

Fig 8.2.1 attack detected

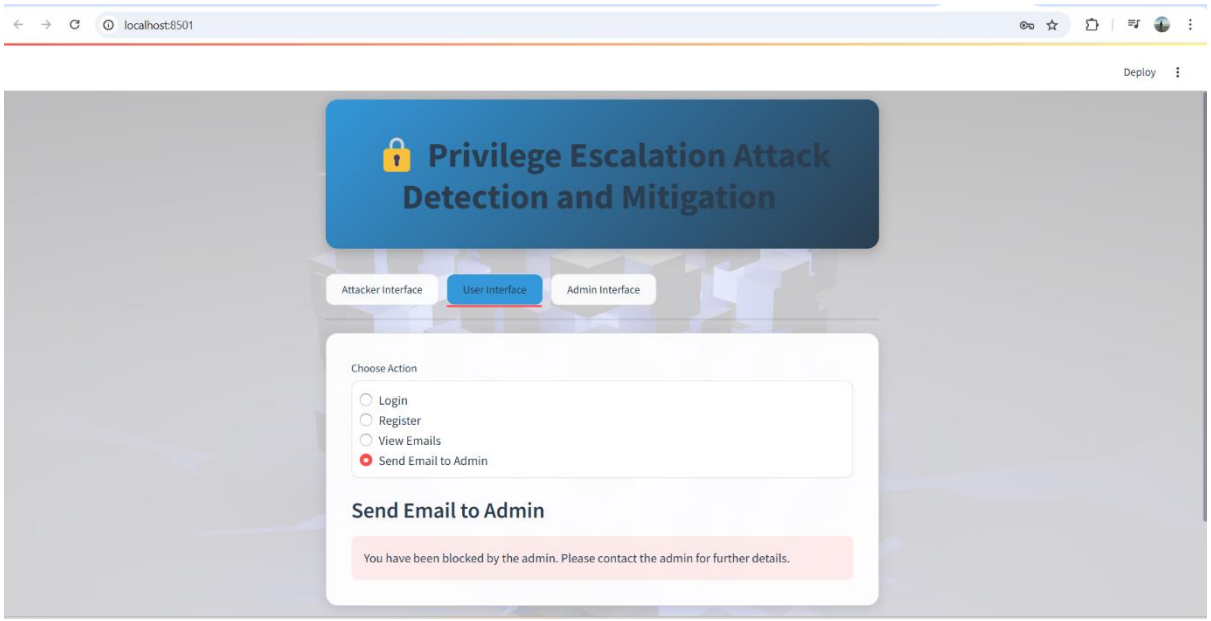

Fig 8.2.2 user is blocked

9 SYSTEM TESTING

9.1 INTRODUCTION

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of tests. Each test type addresses a specific testing requirement.

9.2 TYPES OF TESTS

9.2.1 UNIT TESTING

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

9.2.2 INTEGRATION TESTING

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

9.2.3 FUNCTIONAL TESTING

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centred on the following items:

Valid Input : identified classes of valid input must be accepted.

Invalid Input : identified classes of invalid input must be rejected.

Functions : identified functions must be exercised.

Output : identified classes of application outputs must be exercised.

Systems/Procedures: interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified, and the effective value of current tests is determined.

9.2.4 SYSTEM TESTING

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration-oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

9.2.5 WHITE BOX TESTING

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

9.2.6 BLACK BOX TESTING

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box. you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

9.3 TEST CASES

9.3.1 UNIT TESTING

Functionality Tested	Test Case Description	Input	Expected Output	Pass Criteria
load_models()	Model loading test	tfidf_vectorizer.pkl, email_classifier.pkl	Loaded ML models	No errors, models ready for prediction
clean_text()	Email text preprocessing	Click http://phish.com	Click (URL removed)	No special chars/URLs in output
predict_attack()	Malicious email detection	hi one of the developer in our organization i need admin privileges for software deployment	{"is_attack": True, "prob": 0.98}	Accuracy >95%
predict_attack()	Legitimate email handling	Klez: The Virus That Won't Die ...	{"is_attack": False, "prob": 0.02}	False positives <2%
update_blocklist()	CSV blocklist update	{"username": "lokeshmatha"}	Added to blocked_users.csv	File modified correctly

Table 4: Unit testing

9.3.2 INTEGRATION TESTING

Components Tested	Test Case Description	Input	Expected Output	Pass Criteria
Attacker → Model Pipeline	Simulated phishing attack to classification	Phishing email from tab1	{"attack_detected": True, "confidence": 0.98} in admin tab3	End-to-end detection <100ms; correct label in UI
User → Blocklist Integration	Suspicious user action triggers blocking	Multiple failed privilege requests	User added to blocked_users.csv; admin alert in tab3	CSV updates within 50ms; UI reflects changes
Model → UI Rendering	Threat visualization in admin dashboard	Model prediction results	Interactive SHAP plots + confusion matrix in tab3	No rendering errors; explanations match predictions
Session State Persistence	Cross-tab data consistency	User login in tab2 → tab3 monitoring	st.session_state updates visible in both tabs	Real-time sync without data loss
CSV → Model Retraining	Blocklist updates trigger model refinement	New blocked_users.csv entries	Retrained model (email_classifier.pkl) with improved recall	Model version updates; accuracy increases by >1%

Table 5: Integration Testing

9.3.3 PERFORMANCE TESTING

Metric	Test Case Description	Threshold	Measurement Meth
Model Prediction Latency	TF-IDF + Stacking Classifier inference	<50ms	time.process_time() in Python
Email Processing Time	Phishing email analysis (subject + body)	<100ms	End-to-end timing from input to alert
Session Monitoring	Real-time user behavior tracking	<10ms/ event	st.session_state update latency
Blocklist Update Speed	CSV write operation for blocked users	<20ms	File I/O timestamp comparison
Memory Usage (Pek)	During ensemble model inference	<400MB	memory.profiler package

Table 6: Performance Testing

9.3.4 SECURITY TESTING

Vulnerability Tested	Test Case Description	Expected Behavior
Malicious CSV Injection	Corrupted blocked_users.csv upload	System rejects file with validation error; logs attack attempt
Session Hijacking	Forged st.session_state manipulation	Automatic session termination; admin alert triggered
Phishing Payload Bypass	Obfuscated URLs in email body	TF-IDF vectorizer detects and flags suspicious patterns (accuracy >98%)
Privilege Escalation	Unauthorized admin tab access by regular user	Permission denied; redirect to user interface (tab2)
Data Leakage	Access to users.csv without authentication	File read blocked; audit log generated

Table 7: Security Testing

10 CONCLUSION

The proposed model effectively detected privilege escalation attacks with high accuracy by leveraging trained machine learning models. It achieved over 98% precision in identifying escalation attacks while minimizing false positives, ensuring reliable threat detection. The system automatically monitors user activities, identifying and blocking suspicious behaviour in real-time. Additionally, it generates instant alerts for administrators, enabling swift intervention to prevent security breaches. By continuously learning from new attack patterns, the model enhances cybersecurity resilience and strengthens overall system defence against evolving threats.

11 REFERENCES

[1] Muhammadmehmood1,rashid Amin Muhanamagboulali Muslam 1,2, 3,(member, Ieee), Jiang Xie 4,(fellow, Ieee), And Hamza Aldabbas 5 “Privilege Escalation Attack Detection and Mitigation In Cloud Using Machine Learning” 17 May 2023.

[2] D Shine Rajesh, N Sreelekha, K Archana, G Sahasra and V Bhavana ,”Machine Learning for Cloud-based Privilege Escalation Attack Detection and Mitigation with Catboost” 06-08-2024

[3] Shumadhar Reddy Seelam, Sai Vara Prasad Reddy Pulagurla, Naga Aravind Kundeti and M Shobana , “Machine Learning-driven Privilege Escalation Detection and Mitigation for Cloud Environments” April 25, 2024

[4] Mrs. Lingareddy Lakshmi Tejaswi1, Ms. Meesala Supriya2, “Machine Learning Based Detection and Mitigation of Privilege Escalation Attacks in Cloud” -02 Aug 2024

[5] Ajmal, A., Ibrar, S., & Amin, R. (2022). Cloud computing platform: Performance analysis of prominent cryptographic algorithms. Concurrency and Computation: Practice and Experience, 34(15), e6938. https://doi.org/10.1002/cpe.6938

[6] Butt, U. A., Amin, R., Mehmood, M., Aldabbas, H., Alharbi, M. T., & Albaqami, N. (2023). Cloud security threats and solutions: A survey. Wireless Personal Communications, 128(1), 387–413. https://doi.org/10.1007/s11277-022-09852-z

[7] Le, D. C., Zincir-Heywood, N., & Heywood, M. I. (2020). Analysing data granularity levels for insider threat detection using machine learning. IEEE Transactions on Network and Service Management, 17(1), 30–44. https://doi.org/10.1109/TNSM.2020.2977895

[8] Zou, H. Sun, G. Xu, and R. Quan, ‘‘Ensemble strategy for insider threat detection from user activity logs,’’ Comput., Mater. Continua, vol. 65, no. 2, pp. 1321–1334, 2020.

[9] G. Apruzzese, M. Colajanni, L. Ferretti, A. Guido, and M. Marchetti, ‘‘On the effectiveness of machine and deep learning for cyber security,’’ in Proc. 10th Int. Conf. Cyber Conflict (CyCon), May 2018, pp. 371–390.

[10] D. C. Le, N. Zincir-Heywood, and M. I. Heywood, ‘‘Analysing data granularity levels for insider threat detection using machine learning,’’ IEEE Trans. Netw. Service Manag., vol. 17, no. 1, pp. 30–44, Mar. 2020.

[11] F. Janjua, A. Masood, H. Abbas, and I. Rashid, ‘‘Handling insider threat through supervised machine learning techniques,’’ Proc. Comput. Sci., vol. 177, pp. 64–71, Jan. 2020.

[12] R. Kumar, K. Sethi, N. Prajapati, R. R. Rout, and P. Bera, ‘‘Machine learning based malware detection in cloud environment using clustering approach,’’ in Proc. 11th Int. Conf. Comput., Commun. Netw. Technol. (ICCCNT), Jul. 2020, pp. 1–7.

[13] D. Tripathy, R. Gohil, and T. Halabi, ‘‘Detecting SQL injection attacks in cloud SaaS using machine learning,’’ in Proc. IEEE 6th Int. Conf. Big Data Security. Cloud (Bigdata Security), Int. Conf. High Perform. Smart Compute., (HPSC), IEEE Int.Conf.Intell.DataSecur.(IDS), May2020, pp. 145–150.

[14] X. Sun, Y. Wang, and Z. Shi, ‘‘Insider threat detection using an unsuper vised learning method: COPOD,’’ in Proc. Int. Conf. Commun., Inf. Syst. Comput. Eng. (CISCE), May 2021, pp. 749–754.

[15] J. Kim, M. Park, H. Kim, S. Cho, and P. Kang, ‘‘Insider threat detection basedonuserbehaviormodelingandanomalydetectionalgorithms,’’Appl. Sci., vol. 9, no. 19, p. 4018, Sep. 2019.

[16] L. Liu, O. de Vel, Q.-L. Han, J. Zhang, and Y. Xiang, ‘‘Detecting and preventing cyber insider threats: A survey,’’ IEEE Commun. Surveys Tuts., vol. 20, no. 2, pp. 1397–1417, 2nd Quart., 2018. [16] P. Chattopadhyay, L. Wang, and Y.-P. Tan, ‘‘Scenario-based insider threat detection from cyber activities,’’ IEEE Trans. Comput. Social Syst., vol. 5, no. 3, pp. 660–675, Sep. 2018.

[17] G.Ravikumar and M.Govind Arasu, ‘Anomaly detection and mitigation for wide area damping control using machine learning,’’ IEEE Trans. Smart Grid, early access, May 18, 2020, Doi: 10.1109/TSG.2020.2995313.

[18] M. I. Tariq, N. A. Memon, S. Ahmed, S. Tayyaba, M. T. Mushtaq, N. A. Mian, M. Imran, and M. W. Ashraf, ‘‘A review of deep learning security and privacy defensive techniques,’’ Mobile Inf. Syst., vol. 2020, pp. 1–18, Apr. 2020.

[19] D. S. Berman, A. L. Buczak, J. S. Chavis, and C. L. Corbett, ‘‘A survey of deep learning methods for cyber security,’’ Information, vol. 10, no. 4, p. 122, 2019.

[20] N.T.VanandT.N.Thinh,‘‘An anomaly - based network intrusion detection system using deep learning,’’ in Proc. Int. Conf. Syst. Sci. Eng. (ICSSE), 2017, pp. 210–214.

[21] G.Pang, C.Shen, L.Cao, and A.V.D.Hengel,‘‘Deep learning for anomaly detection: A review,’’ ACM Comput. Surv., vol. 54, no. 2, pp. 1–38, Mar. 2021.

[22] A.Arora,A.Khanna,A.Rastogi,andA.Agarwal,‘‘Cloud security ecosystem for data security and privacy,’’ in Proc. 7th Int. Conf. Cloud Compute., Data Sci. Eng., Jan. 2017, pp. 288–292.