APPLIED INFORMATION TECHNOLOGY AND COMPUTER SCIENCE
e-ISSN: 2773-5141

AITCS

Vol. 6 No. 2 (2025) 772-792
https://publisher.uthm.edu.my/periodicals/index.php/aitcs

PhishBlocker: Phishing Detection System based on Uniform Resource Locator (URL) using Hybrid Approach

Muhammad Ilyas Noor Izzri[1], Isredza Rahmi A Hamid[1]*,

[1] *Fakulti Sains Komputer dan Teknologi Maklumat,*
Universiti Tun Hussein Onn Malaysia, Parit Raja, Batu Pahat, 86400, MALAYSIA

*Corresponding Author: rahmi@uthm.edu.my
DOI: https://doi.org/10.30880/aitcs.2025.06.02.042

Article Info

Received: 21 July 2025
Accepted: 19 November 2025
Available online: 30 November 2025

Keywords

Phishing detection system, Support Vector Machine (SVM), Machine learning, URL detection, Phishing URLs, Blacklists and Heuristics

Abstract

Phishing attacks remain a significant threat, emphasizing the need for effective detection systems. This study presents PhishBlocker, a URL-based Phishing Detection System using an integrated approach combining Support Vector Machine (SVM) with rule-based scoring to distinguish phishing URLs from legitimate ones. The system uses a balanced dataset of 60,000 URLs that consists of 30,000 phishing, 30,000 legitimate URLs, cleaned and analyzed with 22 features like domain age, ssl validity, redirection depth and phishing-related keywords. Implementing Object-Oriented Analysis and Design (OOAD), PhishBlocker uses a consensus scoring approach, integrating SVM and rule-based scores. The dataset is split 80:20 for training and testing, with n-fold cross-validation for model evaluation. The system is developed in Python with Flask, Scikit-learn and Chart.js, the system achieved 93% accuracy from training and testing with the balanced dataset. Future work should integrate deep learning and real-time adaptive training to enhance detection, benefiting organizations, cybersecurity professionals, and end-users with improved phishing defenses and reduced false positives.

1. Introduction

Phishing is a type of cyber-attack where attackers deceive individuals into providing sensitive information by pretending to be trusted by organizations through fake Uniform Resource Locator (URLs) shared via emails, social media or messages [1][2]. The Anti-Phishing Working Group (APWG) Phishing Activity Trends Report[3] for quarter 3 of 2024 reported over 932,923 phishing attacks in three months, an increase from the 877,536 attacks recorded in the previous quarter. This highlights the urgent need for effective systems to detect and prevent phishing attempts [3].

As phishing techniques become more advanced, traditional methods such as manual checks and blacklisting struggle to keep up, leaving users vulnerable to cyber threats [4]. Blacklists and rule-based tools fail to detect new phishing links, as attackers frequently modify URL structures to bypass detection [5][6]. As a result, phishing detection systems may miss new phishing URLs, leaving users vulnerable to attacks. This leads to missed threats, data breaches, financial loss, and reduced trust especially when false positives flag legitimate URLs as phishing [7]. This not only frustrates users but also disrupts workflows, reduces trust in the detection system and limits the effectiveness of legitimate websites, creating a usability challenge. Moreover, current tools achieved high false-positive rates [8]. Also, existing systems overlook critical URL features, making them ineffective against evolving phishing tactics [5]. While machine learning can adapt to evolving phishing techniques, it relies heavily on large, well-labeled datasets and still faces issues like false positives, especially with unfamiliar threats. Its performance depends on data quality and requires regular retraining to stay effective. On the other hand, rule-based systems

are good at spotting known threats but struggle to keep up with new phishing patterns. As attackers change URL structures, these systems become less accurate and often need manual rule updates. This can lead to many false alarms, wrongly flagging safe URLs as malicious.

To address these challenges, the objectives of this project are to design PhishBlocker: Phishing Detection System based on URL which integrates Machine Learning algorithm and rule-based method and to evaluate the proposed system in terms of user acceptance and system functionality. We specify using Support Vector Machine (SVM) algorithm [1] that combines with heuristics rule-based scoring. The process involves gathering data, extracting features, training the model and deploying the system to analyze URL characteristics like length, subdomains and symbols for accurate classification [5]. We implemented the proposed system using Python with libraries like Scikit-learn and Flask for the web interface. The system utilize public datasets, including PhiUSIIL Phishing URL [9]. The PhishBlocker system ensures comprehensive feature analysis and improves phishing detection effectiveness. There are additional features included result visualization, archive history, admin control panel and user authentication.

The remainder of this paper is structured as follows: Section 2 covers related work, Section 3 outlines the methodology, Section 4 details the design and analysis, and Section 5 presents the results and implementation. The conclusion highlights key findings and future directions.

2. Related Work

This section covers types of phishing attacks, components of a URL, phishing detection techniques, machine learning algorithms, existing phishing detection system and the proposed SVM-based system.

2.1 Phishing Attack

Phishing attacks deceive people into sharing sensitive information by pretending to be trusted sources. These attacks have evolved from simple email scams to complex methods using psychological tactics and exploiting technology weaknesses, causing major cybersecurity risks, financial losses and data breaches [10]. According to the Anti-Phishing Working Group (APWG), phishing attacks reached a record high of over 1.5 million detected incidents in quarter 3 of 2024, highlighting the growing sophistication of these threats and their impact on individuals and organizations worldwide [3]. This alarming increases the urgent need to improve detection and prevention measures in this area.

Phishing works by exploiting human trust and technological vulnerabilities. Attackers, also known as phishers, pose as legitimate entities, such as banks, government agencies or well-known companies, to manipulate victims into taking specific actions [11]. They often send messages with urgent instructions, such as resetting a password or confirming account details. These messages may contain links leading to fake websites that look authentic or include malware-laden attachments. Once users provide private information, such as login credentials or financial details, phishers use it for unauthorized access, identity theft or financial fraud [10].

2.2 Types of Phishing Attacks

Phishers are cybercriminals who deceive individuals into sharing sensitive information, such as passwords, financial details or personal data. They impersonate trusted organizations or individuals, creating messages or websites that appear legitimate to trick their victims. These attackers rely on both technical skills and psychological manipulation to achieve their goals [12].

To steal personal information, phishers employ social engineering and technical techniques [13]. These techniques fall into two types, such as malware-based phishing, which installs malicious software for unauthorized access, and social engineering, which uses fear to coerce victims into disclosing personal information [14]. Phishing techniques have developed throughout time into a variety of forms, including spear phishing, whaling, clone phishing, and pharming, each specifically created to trick victims. Table 1 show types of phishing attacks.

Table 1 *Types of Phishing Attacks*

Types of Phishing Attacks	Description
Spear Phishing	Targets specific individuals or organizations using personal information to appear credible. Attackers collect details from public sources, such as social media, to create personalized messages, for example mentioning a project or mutual connection, increasing the likelihood that victims will unknowingly share sensitive information [11][12].

Table 1 *(cont)*

Types of Phishing Attacks	Description
Whaling	Focuses on high-level executives, like CEOs, to access valuable information. Attackers conduct detailed research and craft convincing messages using social engineering. These high-risk attacks often result in financial losses and reputational damage. Common channels include eFax and email [11].
Clone Phishing	Involves creating near-identical copies of legitimate messages or websites but includes harmful links or attachments. This tactic deceives users into thinking they are interacting with trusted sources, leading them to unknowingly share sensitive data or login details [10].
Pharming	Also known as DNS-based phishing, this method manipulates DNS settings to redirect users to fake websites without their knowledge. It impacts many users simultaneously, leading to large-scale data theft and financial losses as users unknowingly provide sensitive information [10].

2.3 Uniform Resource Locator (URL)

Uniform Resource Locator (URL) is a reference to a web resource that specifies its location on a computer network and the mechanism for retrieving it. URL consists of several components, each serving a specific purpose in accessing web resources as shown in Fig. 1. The scheme defines the protocol http or https for a secure connection. The subdomain organizes site sections such as www while the domain is the main name, and the top-level domain (TLD) indicates category or country code for example.com, .co.uk. The port number, such as 443 for HTTPS, supports server communication. The path identifies the page location, and the query string such as ?docid=720&hl=en provides additional parameters for dynamic content. Lastly, the fragment # points to specific page sections for navigation [5], [13][14].

For detecting malicious activity, key URL parts include the domain name and subdomain, as attackers often use lookalike or multiple subdomains to mimic legitimate sites. Suspicious paths with encoded characters or phishing keywords are also indicators. Query strings may be manipulated to hide intent, and the absence of HTTPS is a common red flag. Additionally, suspicious or uncommon top-level domains (TLDs) like .xyz or .info are often linked to phishing [15][16][17].

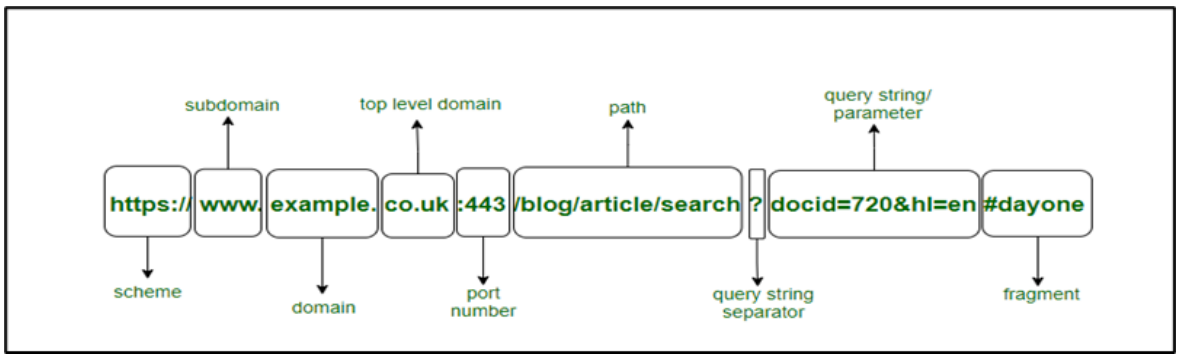


Fig. 1 *Components of a URL*[14], [18]

2.4 Phishing Detection Techniques

There are various phishing detection techniques that are used to detect phishing attacks, from basic rule-based systems to more advanced machine learning techniques [19]. These approaches help identify and prevent phishing attempts, with traditional methods relying on set rules, while machine learning algorithms adapt to recognize new phishing patterns more effectively [20].

2.4.1 URL-Based

URL-based detection focuses on analyzing various components and features of URL to identify phishing attempts. Features such as URL length, domain information and subdomains are commonly used. This approach offers a scalable solution since it does not rely on content analysis or heavy computational resources, making it efficient for batch processing and periodic detection [21].

2.4.2 Blacklist-Based

Blacklist-based detection blocks phishing sites by comparing URLs to a database of known malicious websites. However, it cannot detect new phishing URLs which are not yet listed. All the matching backlist URLs are denied, which prevents users from visiting them. Some methods also use Google's PageRank to evaluate website reliability [22]. Blacklist-based detection compares URLs against databases of known malicious sites. It uses features like URL similarity analysis and integration with tools such as Google Safe Browsing. While it is simple and effective for blocking known phishing URLs, it cannot detect new or zero-day attacks and requires frequent updates [4].

2.4.3 Heuristic-Based

Heuristic-based detection uses predefined rules to classify URLs based on phishing indicators like IP addresses, "@" symbols, disabled right-clicks and pop-ups requesting passwords [22]. Some rules also flag emails with specific keywords or domains. While effective for known threats, it may fail against evolving phishing tactics that do not match these patterns [23]. Heuristic-based detection relies on predefined rules to identify phishing URLs. Key features include detecting IP addresses in domains, analyzing URL length, special characters and anomalies in SSL certificates or favicons. It can spot new phishing patterns but has high false-positive rates and limited scalability for advanced attacks [7].

2.4.4 Machine Learning-Based

Machine learning-based detection uses algorithms like Support Vector Machine (SVM) to classify URLs by analyzing specific features. SVM is effective in finding boundaries between classes, distinguishing phishing from legitimate URLs. The accuracy depends on the algorithm used, making SVM suitable for improving phishing detection in this project [22]. Machine learning-based detection analyzes features like URL length, domain metrics, HTTPS usage and query string parameters. It adapts to new phishing patterns with higher accuracy and fewer false positives. However, it requires large datasets for training and is computationally intensive [24]. URL-based detection is lightweight and scalable, avoiding deep content analysis. Machine learning enhances adaptability to new phishing tactics, reduces false positives, and improves scalability by analyzing features like URL length, subdomains, and HTTPS usage.

2.5 URL Features

We use the PhiUSIIL Phishing URL dataset, which is categorized into five feature groups such as URL Structure, Domain Features, Character Analysis, Security Indicators and Content Analysis[9]. The PhiUSIIL dataset includes 235,795 URLs, with 134,850 legitimate and 100,945 phishing. A balanced subset of 60,000 URLs. Total of 30,000 URL for each class was selected from the UCI Repository for training and testing PhishBlocker [9][17][25]. This subset, drawn from the larger PhiUSIIL dataset, ensures consistency and reliability without mixing multiple datasets. The extracted features are listed in Table 2.

Table 2 *Features extraction in the dataset* [9]

No	Category	Features
1	URL Structure	url_length, tld_length, domain_length
2	Domain Features	no_of_subdomain, subdomain_length, has_ip
3	Character Analysis	digit_ratio, special_char_ratio, uppercase_ratio
4	Security Indicators	has_https, is_shortened, contains_at
5	Content Analysis	keywords_in_domain, suspicious_patterns

2.6 Machine Learning Algorithms

Phishing detection increasingly relies on machine learning algorithms due to their ability to analyze and classify URL features with high accuracy. This section explores three algorithms used in phishing detection such as Support Vector Machine (SVM), Decision Trees (DT) and Random Forest (RF) [26]**.** SVM is a supervised learning algorithm that classifies URLs by finding the optimal hyperplane to separate phishing and legitimate links. It uses features like URL length, subdomains, and special characters, and is effective for both linear and non-linear data, helping reduce false positives and improve accuracy [26]. DT classifies URLs using a tree structure with feature-based splits, offering simple and interpretable results but is prone to overfitting. RF improves this by combining multiple trees from random data subsets and uses majority voting for more reliable predictions [6]. While RF handles high-dimensional data well, its computational requirements make it less practical for this project

compared to the efficiency and adaptability of SVM [26]. SVM outperformed DT and RF by offering better accuracy, efficiency, and handling of complex phishing patterns, with studies supporting its effectiveness in this domain.

2.7 Existing Phishing Detection Systems

This section discusses current phishing detection systems such as VirusTotal [27], CheckPhish [28] and IsItHacked [27][28][29].

2.7.1 VirusTotal

VirusTotal is an online tool that scans files, URLs and IP addresses for threats using multiple antivirus engines and data sources [27]VirusTotal detects malware, phishing, and suspicious URLs by comparing them against known threat databases. Users can submit files, URLs, or IPs for analysis, with results aggregated from multiple antivirus engines. It relies on signature-based detection, making it effective for known threats but less so for zero-day attacks. VirusTotal does not use machine learning for phishing detection [27].

2.7.2 CheckPhish

CheckPhish is an AI-based tool designed to detect phishing URLs in real time [28]. CheckPhish analyzes URLs, detects phishing sites, and builds a phishing database using AI and machine learning. It assigns risk scores based on factors like domain age and SSL status, working without manual input. Its adaptive system effectively detects evolving threats through automated analysis and database updates [28].

2.7.3 IsItHacked

Is It Hacked checks websites for signs of hacking, such as spam links, unauthorized redirects and malware infections [29]. Is It Hacked assesses website security using rule-based detection to identify compromises like content changes or unauthorized access. It doesn't use machine learning but focuses on reporting breaches and warning users about unsafe sites by scanning for modifications and redirect patterns [29].

2.8 Comparison Between Existing Systems and Proposed System

Table 3 shows a comparison between the existing systems such as VirusTotal [27], CheckPhish [28] and IsItHacked [29] with the proposed system.

Table 3 *Comparison Table Between Existing Systems with the Proposed System*

Features \ Tools	VirusTotal	CheckPhish	IsItHacked	Proposed System
Dataset	Multiple antivirus and cybersecurity sources (Not defined)	Automated phishing URL detection, Bolster's internal data (Not defined)	Automated scans and user submissions (Not defined)	PhiUSIIL Phishing URL dataset from UCI Machine Learning Repository [9]
Scan Files	Yes	No	No	No
Scan URL	Yes	Yes	Yes	Yes
Scan IP address	Yes	No	No	Yes
Technique Used	Primarily blacklist-based	Machine learning-based detection, no manual input	Website scanning and analysis	Pattern analysis and feature extraction
Machine Learning Algorithm	Yes (Not Defined)	Yes (Not Defined)	Yes (Not Defined)	Support Vector Machine (SVM)
Heuristic rule-based method	No	No	Yes	Yes
Archive URLs	Yes	No	No	Yes
Downloadable CSV dataset	No	No	No	Yes

All systems share the capability to scan URLs, however, only the proposed system and VirusTotal provide an archive feature for historical data. Unlike the others, the proposed system exclusively uses the PhiUSIIL dataset

for training, ensuring higher adaptability to phishing patterns, while VirusTotal and CheckPhish rely on general databases or internal data. The proposed system uses Support Vector Machine (SVM) for pattern analysis and feature extraction, whereas VirusTotal primarily employs blacklist-based detection and CheckPhish uses machine learning without defining specific algorithms. The proposed system is designed for high adaptability to new phishing threats, unlike IsItHacked, which is limited in its ability to detect evolving threats. Additionally, only the proposed system provides a comprehensive feature set, including classification results, historical data storage, and report generation tailored to user needs.

3. Methodology

This section discusses the Object-Oriented analysis and Design (OOAD) model with five phases which are the requirement phase, design phase, implementation phase, verification phase and maintenance phase.

3.1 Object-Oriented Analysis and Design Model

The object-oriented model was chosen for its structured, modular, and scalable design. It organizes components like input handling, feature extraction, SVM classification, and output display into separate, integrated modules. This approach simplifies updates, supports iterative development, and ensures adaptability to evolving phishing threats. Fig. 2 and Table 4 illustrate the OOAD model and its phases [21].

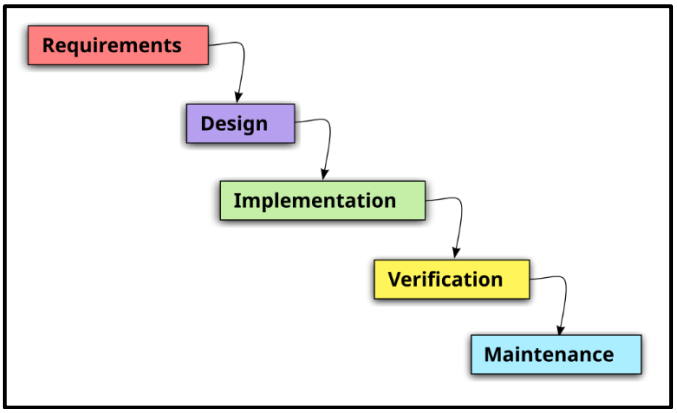


Fig. 2 *Object-Oriented Analysis and Design (OOAD) model [18]*

Table 4 *Methodology Phases with Description*

Phase	Description	Output
Requirement	Establishing project goals, defining system scope, reviewing phishing patterns, and selecting tools (VirusTotal [27], CheckPhish [28], IsItHacked [29]).	Defined functional and non-functional requirements, including URL validation, feature extraction, and accuracy goal (93%). Chose PhiUSIIL dataset[9] with 60,000 balanced URLs.
Design	Creating a modular system plan, UI mockups, and class diagrams for use cases, class structures, and workflows.	Modular system design, class diagrams, planned components for input handling, feature extraction, classification, and output visualization.
Implementation	Coding modules in Python using Flask, implementing feature extraction, classification, and a hybrid scoring system.	Developed system modules (data_preprocessing.py, scan_url.py, phishing_model_training.py), hybrid scoring system, session management, and database migrations.
Verification	Performing unit and integration testing, followed by performance analysis using the 60k cleaned URL dataset.	Accuracy of 93%, precision, recall, F1-score, and UI feedback incorporated from early testers.
Maintenance	Managing version-controlled updates, retraining support, and feature optimization. Admin features for model tracking, user supervision, and log management improved.	Ongoing updates, retraining capabilities, feature optimization, improved admin controls for model tracking and log management.

3.2 Hardware and Software Requirements

The software and hardware requirements outline the tools and setup needed to run the phishing detection system efficiently. It uses Python with Scikit-learn and Flask, and MySQL for data storage. The system runs smoothly on at least an Intel i5 processor, 8 GB RAM, and 512 GB storage. Table 5 details the full specifications to support development and operation.

Table 5 *Software and Hardware Requirements for the Phishing Detection System*

Software	
Categories	Requirement
Operating System	Windows 11
Programming Language	Python
Libraries	Scikit-learn, Flask, Pandas, NumPy
Database	MySQL
Development Environment	Visual Studio Code
Web Browser	Microsoft Edge
Web Development Framework	Flask
Hardware	
Categories	Requirement
Laptop Model	Dell Inspiron 15 5510
Processor	Intel Core i5-11300H or higher
RAM	8 GB DDR4
Storage	512 GB SSD
Network Connectivity	Wi-Fi 6
System Type	64-bit Operating System

4. System Analysis and Design

This section explains the analysis and design of the PhishBlocker Phishing Detection System, covering the system's architecture, requirements, structure, database design, user interface and testing plans. It uses Use Case, UML Class and Flowcharts to show how the system works, focusing on making a simple, reliable and easy to use system that classifies phishing URLS.

4.1.1 System Architecture

Fig. 3 illustrates PhishBlocker's architecture, where submitted URLs are validated and analyzed for features like length, subdomains, HTTPS, IP address, and special characters. The Classification module uses SVM and rule-based checks to label URLs as phishing, legitimate, or suspicious. Results with confidence scores appear in the Report Module, while the Performance Evaluation component tracks accuracy and stores data for future retraining.

4.1.2 System Requirement

The system requirements cover the features, qualities, and technical specifications needed to develop and run the phishing detection system effectively. These include functional and non-functional aspects, such as performance, scalability and reliability.

Applied Information Technology and Computer Science Vol. 6 No. 2 (2025) p. 772-792

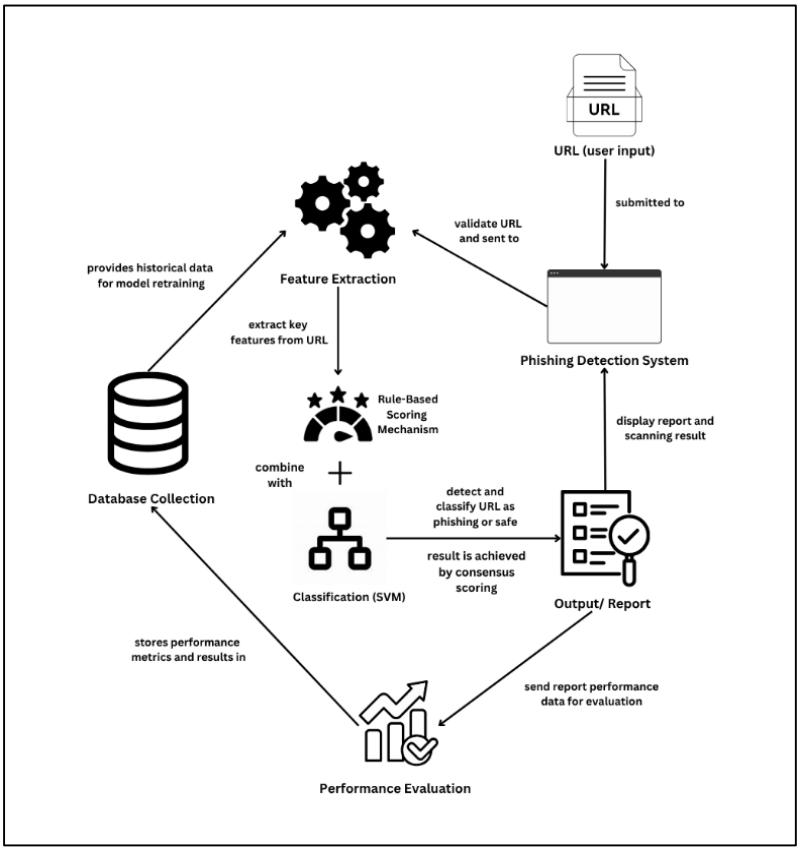


Fig. 3 *System Architecture of PhishBlocker*

4.1.2.1 Functional Requirement

The functional requirements cover key system tasks such as analyzing user-submitted URLs, validating input, extracting features for example length, subdomains, HTTPS, and classifying them using SVM and rule-based analysis. Results include confidence scores and are saved for future retraining. As outlined in Table 6, the system also supports report generation, admin dataset updates, and archive management to enhance functionality and user experience.

Table 6 *Functional Requirement for the Phishing Detection System*

No	Functional Requirement	Description
1	Input URL Validation	The system must allow users to input a URL and validate its format before processing.
2	Feature Extraction	The system extracts key features from the input URL, such as URL length, subdomains, special characters, and HTTPS usage.
3	URL Classification	The system must analyze the extracted features using the Support Vector Machine (SVM) model to classify URLs.
4	Display Results	The system must display the classification result (phishing, legitimate, suspicious) along with a confidence score.
5	Report Generation	Automatically generates detailed reports containing scanned URL results, confidence scores, and feature based analysis.
6	Admin Access for Dataset Updates	Admins must have access to update or upload new datasets to retrain the SVM model and improve accuracy.
7	Archive Management	Enables users to view and manage archived scan results through the system interface from previously scanned URLs.

4.1.2.2 Non-Functional Requirement

Non-functional requirements ensure the system is accurate, fast, secure, scalable, and user-friendly. As shown in Table 7, they cover performance, security, usability, reliability, and maintainability to support efficient operation and future enhancements.

Table 7 *Non-Functional Requirement for the Phishing Detection System*

No	Non-Functional Requirement	Description
1	Performance	The system must classify URLs and display results with minimal latency to provide quick feedback
2	Scalability	The system must be able to handle large datasets and increasing numbers of URL submissions without performance degradation.
3	Security	Utilizes encryption to securely store and transmit sensitive data, including user credentials.
4	Usability	Provides an intuitive interface, enabling both technical and non-technical users to easily navigate and use the system.
5	Reliability	Ensure consistent operation, handling unexpected errors gracefully and providing accurate results.
6	Maintainability	Uses modular components for easy debugging, updating and integration of additional features.

4.2 System Analysis

The system analysis section explains the phishing detection system for users and admin like account management and system monitoring. The system analysis is shown using case diagram, class diagram and flowchart diagram.

4.2.1 Use Case Diagram

The Use Case Diagram shows system functions and interactions for general users and admins. Fig. 4(a) shows admins can log in, manage users, delete user, manage profile, delete account, manage system logs and dashboard, manage archive URL and log out while the general users in Fig. 4(b) can register and log in, forgot password, manage profile, delete account, input URLs, view scan results, view scan history, view archive URL, view report and log out. It highlights key functionalities for both user types.

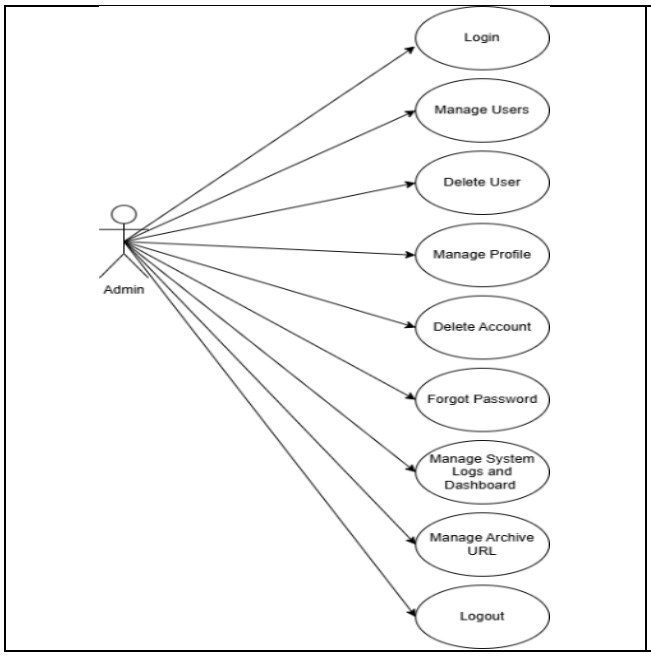


Fig. 4(a) *Use Case Diagram for Admin*

Fig. 4(b) *Use Case Diagram for User*

4.2.2 Class Diagram

Fig. 5 shows the PhishBlocker class diagram, highlighting its main components. The user class handles registration, login, and profile updates, while the scan class manages URL input, feature extraction, and classification. The archive class stores results with metadata, and the report class generates scan summaries. The admin class controls user management and dashboard access. The machineLearning class oversees model training and updates, and the dataAccessLayer handles database operations. This modular design supports scalability, organized development, and efficient data handling.

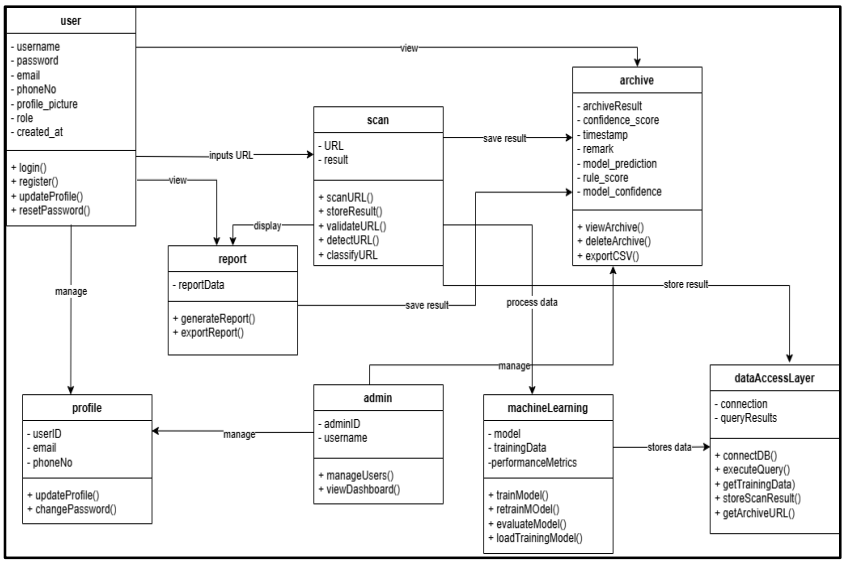


Fig. 5 *Class Diagram for PhishBlocker*

4.2.3 Flowchart Diagram

Fig. 6 shows the PhishBlocker system flowchart, detailing the process from user registration to URL classification. Users can register, log in, or reset passwords via a secure email link. Once logged in, the dashboard provides access to features like URL scanning, scan history, archive management, and profile updates. Submitted URLs are validated and analyzed using a hybrid model combining SVM and rule-based scoring. Results are categorized as phishing, legitimate, or suspicious and displayed with a confidence score and detailed report. All scan data is securely stored in the database. The system also supports continuous SVM retraining using stored datasets to improve accuracy and adapt to evolving phishing threats.

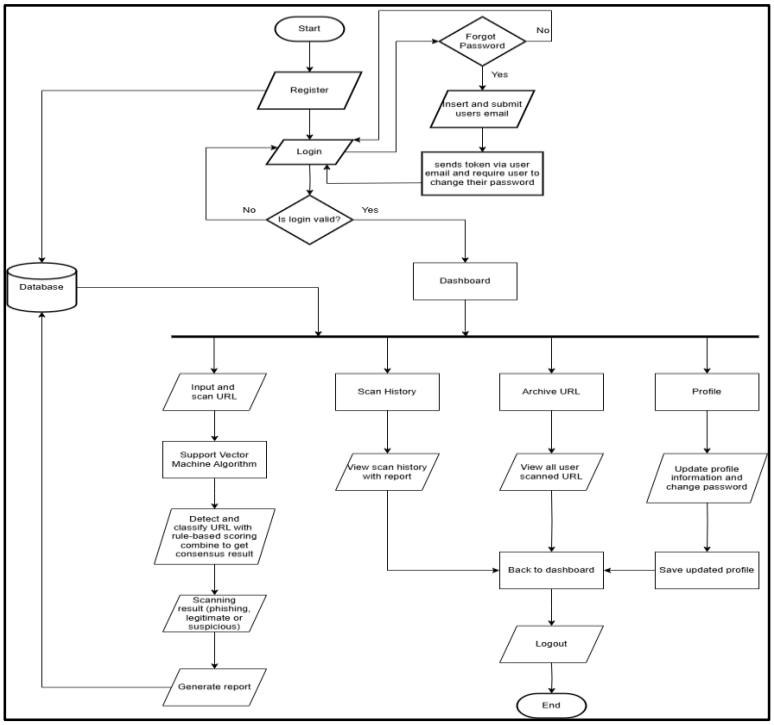


Fig. 6 *Flowchart Diagram for PhishBlocker*

4.3 Data Dictionary

The PhishBlocker database efficiently manages user authentication, URL scanning, and classification results using multiple linked tables. The users table stores login credentials, while the archive_urls table holds scan results, confidence scores, and extracted features for future analysis and model improvement. Key constraints like primary or foreign keys, unique fields, and default values ensure data consistency. Unique identifiers for example usernames, emails to prevent duplication, and timestamps log activities. This structured design supports smooth data retrieval and reliable tracking of user actions and scans. Tables 8 and 9 detail the fields, data types, and constraints for both tables, aiding development and testing.

Table 8 *User Table for the PhishBlocker System*

Field Name	Data Type	Description	Constraints
user_id	INTEGER	Unique identifier for each user user	Primary Key, Auto-Increment
username	VARCHAR(50)	User's login name	Unique, Not Null
email	VARCHAR(155)	User's email address	Unique, Not Null
password	VARCHAR(255)	Encrypted password for user authentication	Not Null
phone_number	VARCHAR(15)	User's contact number	Unique, Not Null
profile_picture	VARCHAR(255)	File path of user's profile image	Unique, Not Null
role	ENUM('user','admin')	Role of the user in the system	Default: 'user'
created_at	TIMESTAMP	Account creation time	Default: Default: CURRENT_TIMESTAMP

Table 9 *Archive URLs Table for the PhishBlocker System*

Field Name	Data Type	Description	Constraints
archive_id	INTEGER	Unique identifier for each user archived result	Primary Key, Auto-Increment
user_id	INTEGER	Foreign key linking to the Users Table	Not Null
url	TEXT	Submitted URL	Not Null
classification	VARCHAR(50)	Final Classification result of the URL	Not Null
confidence_score	FLOAT	Final confidence score from model and rule scoring	Not Null
timestamp	TIMESTAMP	Time when the scan was submitted	Default: Default: CURRENT_TIMESTAMP
remark	TEXT	Explanation or safety message	Not Null
url_length	INTEGER	Total length of the URL	Not Null
no_of_digits_in_url	INTEGER	Count the digits present in the URL	Not Null
location	VARCHAR(255)	Geolocation of the domain	Not Null
ipaddress	VARCHAR(255)	IP address of the URL	Not Null
model_prediction	VARCHAR(255)	Prediction label given the machine learning model	Not Null
rule_score	VARCHAR(255)	Score calculated using heuristic rules	Not Null
model_confidence	VARCHAR(255)	Confidence of the model's classification	Not Null
has_ip	TINYINT(1)	1 if URL contains IP address, else 0	Not Null
url_entropy	FLOAT	Shannon entropy of the URL	Not Null
domain_length	INTEGER	Length of the domain	Not Null
no_of_subdomain	INTEGER	Number of subdomains	Not Null

Table 9 (cont.)

Field Name	Data Type	Description	Constraints
contains_at	TINYINT(1)	1 if '@' symbol is found in the URL	Not Null
contains_percent	TINYINT(1)	1 if '%' symbol is found	Not Null
has_https	TINYINT(1)	1 if URL uses HTTPS	Default: 0
is_shortened	TINYINT(1)	1 if URL is shortened	Default: 0
domain_exists	TINYINT(1)	1 if domain exists	Default: 0
domain_age_score	TINYINT(1)	Estimated domain age in days	Not Null
has_valid_ssl	TINYINT(1)	1 if SSL certificate is valid	Not Null
redirect_count	INTEGER	Number of times the page redirects	Default: 0
suspicious_redirect	TINYINT(1)	1 if redirection behavior is suspicious	Not Null
domain_changed	TINYINT(1)	1 if domain in redirection differs	Not Null
phishing_keyword count	INTEGER	Count of phishing-related keywords in URL	Not Null
has_suspicious_tld	TINYINT(1)	1 if Top-Level Domain is suspicious	Not Null
resolved_url_length	INTEGER	Final resolved URL length	Not Null

4.4 User Interface Design

The user interface of the PhishBlocker system is designed to be clean, user-friendly, and easy to navigate. As shown in Fig. 7(a), the layout includes a sidebar menu for quick access to key sections such as the dashboard, scanner, history, archive and profile. After logging in, users are welcomed with a dashboard that displays the number of total scans and a summary of URL classifications. Fig. 7(b) shows the scanner page that allows users to enter a URL for checking and presents the result with a clear label and confidence score as in Fig. 7(c). The scan history section lists past scans with filtering options and export features as in Fig. 7(d). A public archive page as in Fig. 7(e) shows scan results from all users, offering transparency and the ability to search and download data. Users can also manage their profile by updating their email or password as in Fig. 7(f). Overall, the interface is structured to support smooth interaction, whether viewing past scans, submitting new URLs, or managing personal account details.

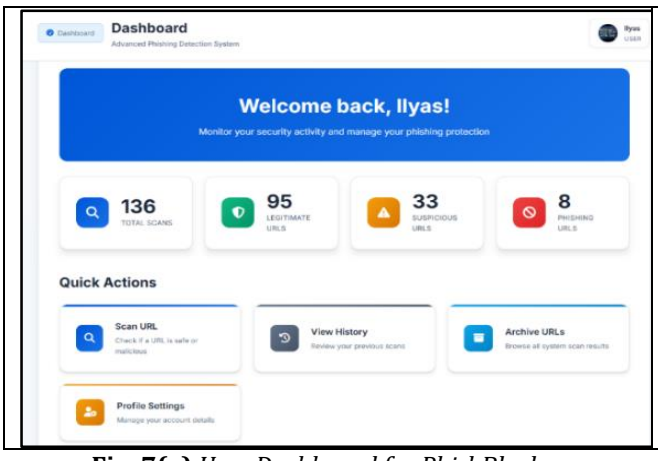


Fig. 7(a) *User Dashboard for PhishBlocker*

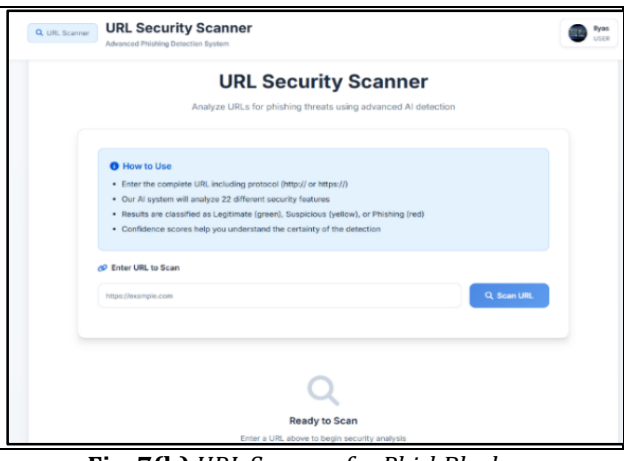


Fig. 7(b) *URL Scanner for PhishBlocker*

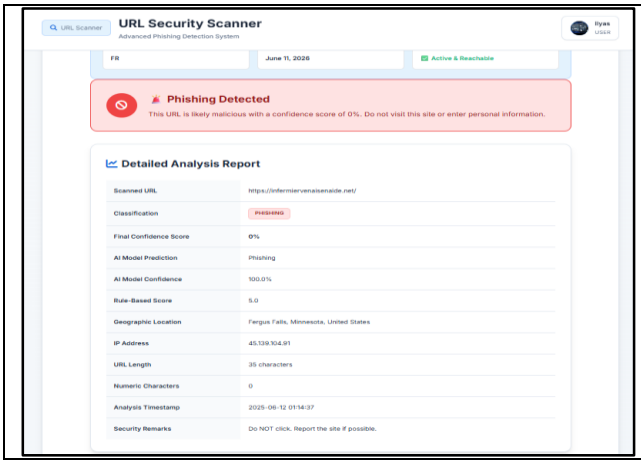


Fig. 7(c) *Detail Analysis for PhishBlocker*

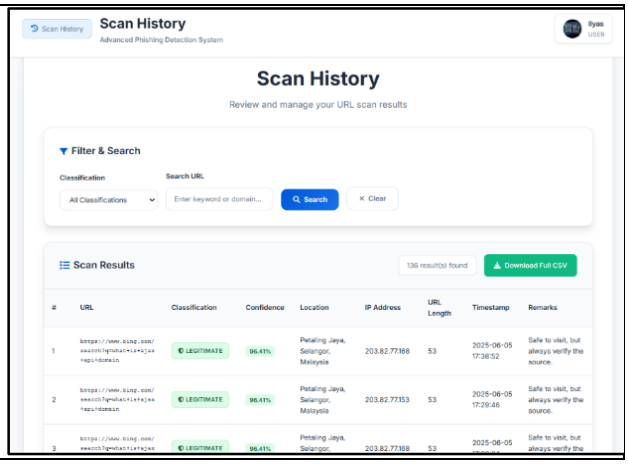


Fig. 7(d) *User Scan History for PhishBlocker*

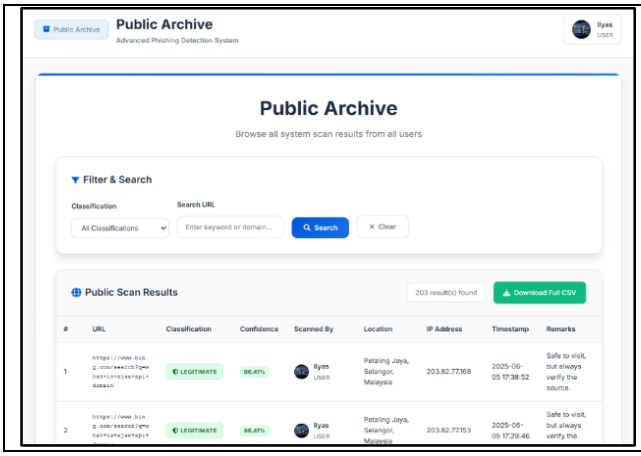


Fig. 7(e) *Public Archive URLs for PhishBlocker, where it contains all the scan in the database*

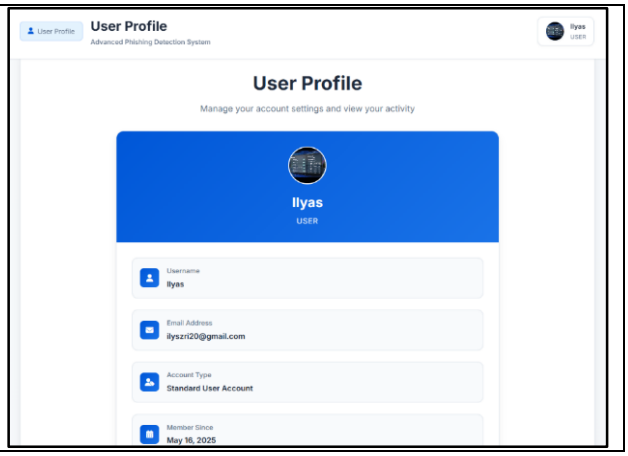


Fig. 7(f) *User Profile page for PhishBlocker*

4.5 Test Plan

This section outlines the test plan to evaluate the system's functionality, including homepage, login, URL scanning, archive, and report generation. Usability and performance tests ensure responsiveness, navigation and efficiency as shown in Table 10.

Table 10 *Test Plan for the Phishing Detection System*

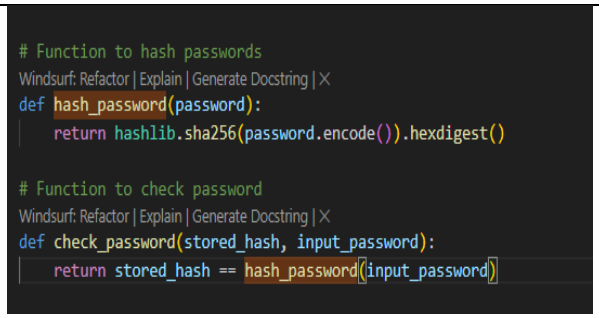
No	Test List	Description	Actual Result
1	Homepage Functionality	Verify that the home page features, such as navigation to other sections and instructional content, function correctly.	Pass/ Fail
2	Login & Registration	Assess the system's login and registration features for secure and seamless user authentication.	Pass/ Fail
3	Scanning Process	Evaluate the initiation, validation, and completion of the URL scanning process on the scanning page.	Pass/ Fail
4	Archive Functionality	Test the storage and retrieval of archived scan results for user reference and system maintenance.	Pass/ Fail
5	Report Generation	Validate the system's ability to generate detailed reports summarizing phishing statistics and user activity.	Pass/ Fail
6	Usability Test	Assess the overall user experience, including navigation, responsiveness, and interface clarity.	Pass/ Fail
7	Performance Test	Test the system's response time and efficiency under different scenarios and workloads.	Pass/ Fail

5. Result and Discussion

This section presents the results and discussion of the PhishBlocker system, covering the implementation of security, machine learning and scoring features, followed by the test outcomes from security verification, user acceptance and API testing. The system was developed to enhance phishing detection accuracy while maintaining a user-friendly interface and secure backend infrastructure.

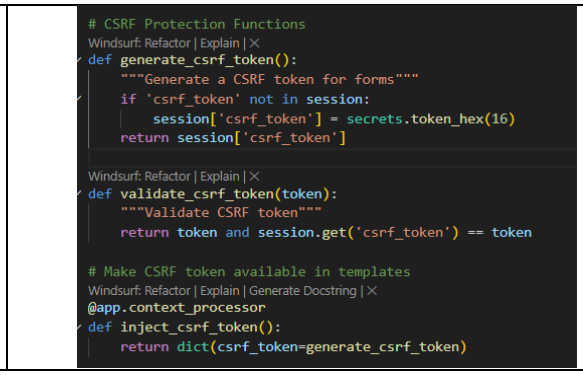
5.1 Implementation of Security Properties

The PhishBlocker system integrates essential security features to ensure safe access and secure data handling for all users. Passwords submitted during registration are not stored in plaintext; instead, they are hashed using a custom hash_password() function that applies the SHA-256 algorithm, ensuring a consistent and irreversible transformation. During login, the check_password() function compares the stored hash with the hash of the input password, effectively validating credentials without revealing actual passwords, as illustrated in Fig. 8(a). This method prevents sensitive data exposure, even in the event of database breach. In addition, the system enforces session management using Flask's built-in session functionality to securely maintain user login states. To restrict unauthorized access, critical views such as the dashboard, profile, and scanning features are protected using the @login_required decorator. For enhanced protection against cross-site request forgery (CSRF) attacks, the system includes custom CSRF token generation through generate_csrf_token(), validation via validate_csrf_token(), and automatic injection into templates using @app.context_processor as shown in Fig. 8(b). These mechanisms collectively uphold authentication integrity, user privacy, and overall system security throughout the application lifecycle.



```
# Function to hash passwords
Windsurf: Refactor | Explain | Generate Docstring | X
def hash_password(password):
    return hashlib.sha256(password.encode()).hexdigest()

# Function to check password
Windsurf: Refactor | Explain | Generate Docstring | X
def check_password(stored_hash, input_password):
    return stored_hash == hash_password(input_password)
```

Fig 8(a) *Code Snippet of hash passwords*

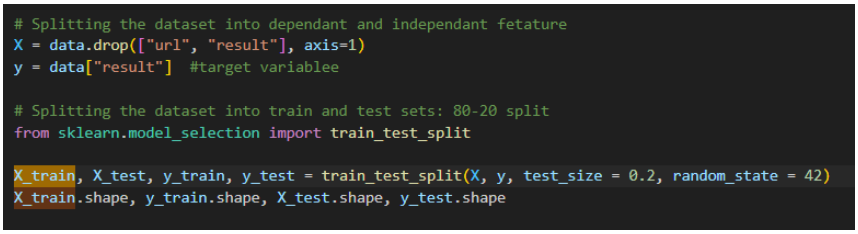


```
# CSRF Protection Functions
Windsurf: Refactor | Explain | X
def generate_csrf_token():
    """Generate a CSRF token for forms"""
    if 'csrf_token' not in session:
        session['csrf_token'] = secrets.token_hex(16)
    return session['csrf_token']

Windsurf: Refactor | Explain | X
def validate_csrf_token(token):
    """Validate CSRF token"""
    return token and session.get('csrf_token') == token

# Make CSRF token available in templates
Windsurf: Refactor | Explain | Generate Docstring | X
@app.context_processor
def inject_csrf_token():
    return dict(csrf_token=generate_csrf_token)
```

Fig 8(b) *Code Snippet of CSRF Tokens*

5.2 Implementation of Machine Learning Properties

The PhishBlocker system uses a machine learning model to classify URLs as either phishing or legitimate based on structured feature extraction. The Support Vector Machine (SVM) classifier implemented using the Scikit-learn library. It was chosen for its robustness and efficiency in handling binary classification problems. The dataset, composed of both phishing and legitimate URLs, is preprocessed by the getInputArray(url) function, which extracts over 22 features such as URL length, usage of HTTPS, subdomain presence, domain entropy, and inclusion of suspicious or temporary keywords, as illustrated in Fig. 9(b). The features extracted are derived from previous work which are then reused in PhishBlocker system [18]. The features are listed in Table 11 with each description. These features are then separated from the target labels using Pandas and split into training and testing sets with an 80:20 ratio to evaluate model generalization as shown in Fig. 9(a). The SVM is initialized with a linear kernel and probability=True to enable confidence scoring. It is then trained using the fit() method and tested on both datasets. Predictions are generated and converted to NumPy arrays to support further metric evaluation. The resulting model achieved an accuracy of 93%, F1-score of 93%, recall of 90%, and a precision of 97%, on the test set. This training and prediction process is shown in Fig. 9(c). The finalized model is saved for reuse, ensuring rapid and consistent classification during live URL scans.



```
# Splitting the dataset into dependant and independant fetature
X = data.drop(["url", "result"], axis=1)
y = data["result"]  #target variablee

# Splitting the dataset into train and test sets: 80-20 split
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

Fig 9(a) *Code Snippet of splitting the dataset to training and testing*

```
Windsurf: Refactor | Explain | Generate Docstring | ×
def getInputArray(url):
    result = []
    result.append(InvalidUrl(url))
    result.append(UsingIp(url))
    result.append(longUrl(url))
    result.append(shorteningService(url))
    result.append(Extension(url))
    result.append(symbol(url))
    result.append(getDepth(url))
    result.append(queryParameters(url))
    result.append(domainEntropy(url))
    result.append(redirecting(url))
    result.append(prefixSuffix(url))
    result.append(SubDomains(url))
    result.append(Https(url))
    result.append(tld(url))
    result.append(NonStdPort(url))
    result.append(pathTokens(url))
    result.append(HTTPSDomainURL(url))
    result.append(suspiciousDomain(url))
    result.append(temporaryDomain(url))
    result.append(englishLetters(url))
    result.append(dashUrl(url))
    result.append(digitUrl(url))
```

Fig. 9(b) *Code Snippet of feature extraction*

```
# Define the SVM classifier
svm = SVC(probability=True, kernel='linear', random_state=42)

# Train the SVM classifier
svm.fit(X_train, y_train)

# Predict on the training and test sets
y_train_pred = svm.predict(X_train)
y_test_pred = svm.predict(X_test)

# Ensure the predictions are numpy arrays
y_train_pred = np.array(y_train_pred)
y_test_pred = np.array(y_test_pred)
y_train = np.array(y_train)
y_test = np.array(y_test)
```

```
SVM: Accuracy on training Data: 0.935
SVM: Accuracy on test Data: 0.937

SVM: f1_score on training Data: 0.932
SVM: f1_score on test Data: 0.932

SVM: Recall on training Data: 0.897
SVM: Recall on test Data: 0.903

SVM: precision on training Data: 0.970
SVM: precision on test Data: 0.970
```

Fig. 9(c) *Code Snippet of SVM Prediction and Accuracy*

Table 11 *URL Features used in PhishBlocker*

No	Feature	Shortened Description
1	Invalid URL	Validates whether the URL format is correct.
2	Using IP	Checks if the URL uses an IP address instead of a domain.
3	Long URL	Flags URLs that are unusually long.
4	Shortening Services	Detect links from URL shorteners like bit.ly.
5	Extension	Identify suspicious file extensions in the URL.
6	Symbol	Checks for odd or uncommon symbols in the URL.
7	Depth	Counts how many directory levels the URL contains.
8	Query Parameters	Looks for strange or excessive query strings.
9	Domain Entropy	Measures domain complexity or randomness.
10	Redirection	Detects multiple slashes or redirect patterns.
11	Prefix Suffix	Flags domain using extra separators like dashes.
12	SubDomains	Checks for too many subdomains in the URL.
13	HTTPS	Confirms if the URL begins with “https.”
14	TLD	Detects use uncommon or suspicious domain endings.
15	Non-standard Port	Flags URLs that use ports other than 80 or 443.
16	HTTPS Domain URL	Finds fake “https” text placed inside domain names.
17	Suspicious Domain	Identifies domains linked to phishing or scams.
18	Temporary Domain	Detects use of disposable or short-term domains.
19	English Letters	Flags use of non-English characters in the URL.
20	Dash URL	Detect URLs with too many hyphens.
21	Digit URL	Flags URLs that contain excessive numeric characters.

5.3 Implementation of Scoring Mechanism Properties

PhishBlocker uses a hybrid scoring system that combines machine learning with rule-based logic to detect phishing URLs. When a user submits a URL, it is first analyzed by a trained Support Vector Machine (SVM) model, which generates a probability score using the predict_proba() method, indicating the likelihood of the URL being phishing or legitimate. This score is then complemented by a rule-based system that starts at 100 and deducts points based on over 30 risk-related features, such as IP address usage, suspicious characters, subdomains, recent domain age, and SSL validity, as shown in Fig. 10(a). Traits like missing domain info, unusual top-level domains,

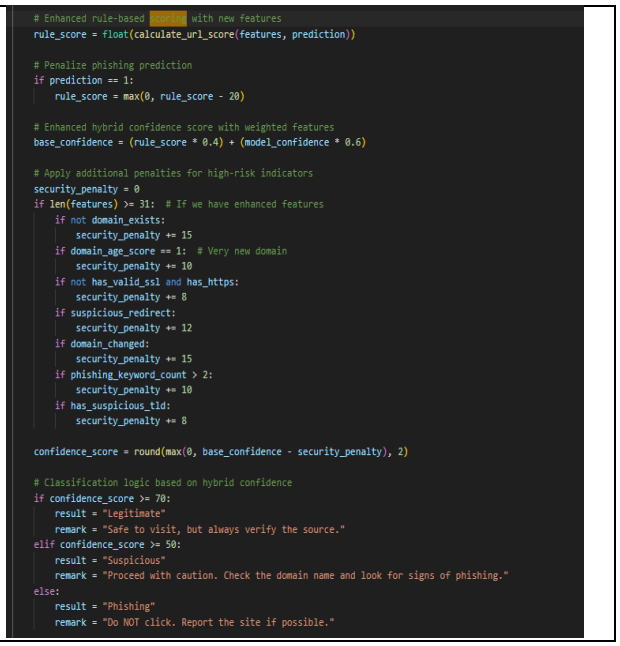
and phishing-related keywords are heavily penalized. The final score is a weighted combination of 60% from the SVM model and 40% from the rule-based logic with additional penalties for high-risk indicators. Based on the final score, URLs are classified as legitimate if score more than equal to 70, suspicious if score less than 70 and more than equal to 50, or phishing if score is less than 50, as outlined in Table 12. The classification process is illustrated in Fig. 10(b), and the system stores the result with a brief remark and displays it to the user to support informed decision-making.

Table 12 *Scoring Mechanism*

Statement	score > =70	score<70 and score >=50	score<50
Result	Legitimate	Suspicious	Phishing

```
def calculate_url_score(features, prediction, whois_info=None, ssl_info=None, access_info=None):
    score = 100

    # Enhanced deduction rules with new security features
    deduction_rules = {
        # Original features
        "HasIP": {"default": 0, "deduct": 15},
        "ContainsAt": {"default": 0, "deduct": 12},
        "ContainsPercent": {"default": 0, "deduct": 8},
        "KeywordsInDomain": {"default": 0, "deduct": 12},
        "NoOfSubDomain": {"condition": lambda x: x > 2, "deduct": 10},
        "URLEntropy": {"condition": lambda x: x > 4.5, "deduct": 12},
        "NoOfSlashes": {"condition": lambda x: x > 5, "deduct": 8},
        "DigitRatioInURL": {"condition": lambda x: x > 0.3, "deduct": 10},
        "HasHTTPS": {"default": 1, "deduct": 20},
        "IsShortened": {"default": 0, "deduct": 18},
        "UppercaseRatio": {"condition": lambda x: x > 0.2, "deduct": 8},
        "modelPrediction": {"malicious": 1, "deduct": 35},

        # New enhanced security features
        "DomainExists": {"default": 1, "deduct": 25},  # Major penalty if domain doesn't exist
        "DomainAge": {"default": 0, "deduct": 20},     # Penalty for very new domains
        "ValidSSL": {"default": 1, "deduct": 15},      # Penalty for invalid/missing SSL
        "SuspiciousRedirect": {"default": 0, "deduct": 15},
        "DomainChanged": {"default": 0, "deduct": 20}, # Major penalty for domain changes
        "PhishingKeywords": {"condition": lambda x: x > 2, "deduct": 15},
        "SuspiciousTLD": {"default": 0, "deduct": 12},
        "URLLength": {"condition": lambda x: x > 100, "deduct": 8},
    }
```

Fig. 10(a) *Snippet code for the rule-based scoring logic*

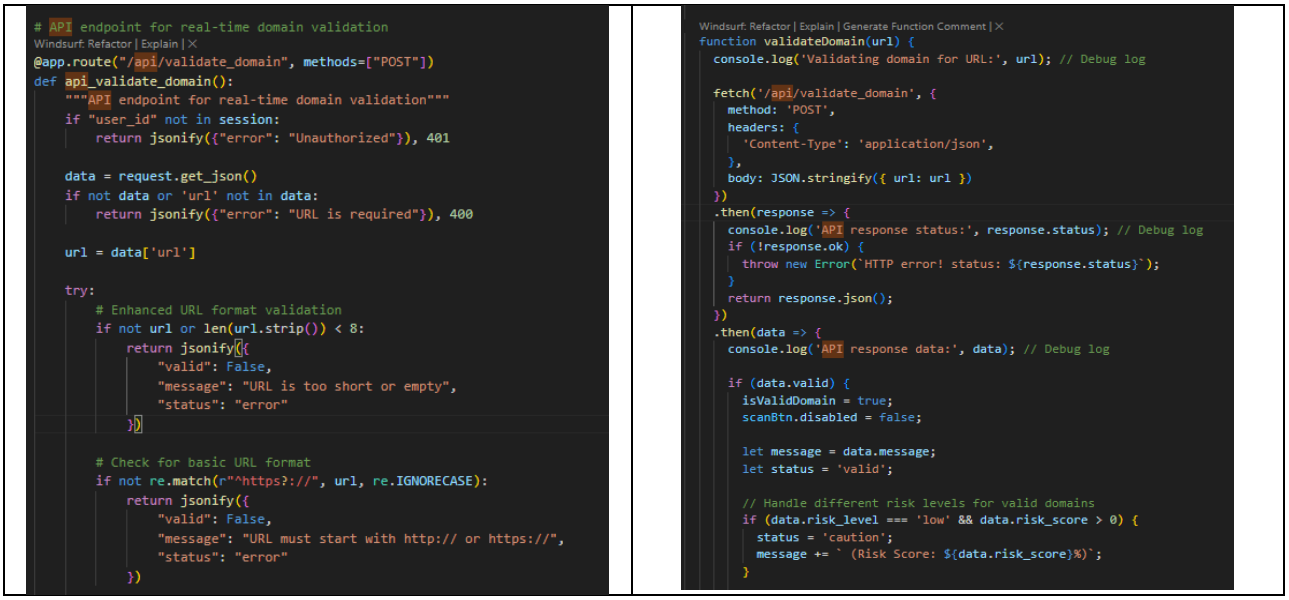


```
# Enhanced rule-based scoring with new features
rule_score = float(calculate_url_score(features, prediction))

# Penalize phishing prediction
if prediction == 1:
    rule_score = max(0, rule_score - 20)

# Enhanced hybrid confidence score with weighted features
base_confidence = (rule_score * 0.4) + (model_confidence * 0.6)

# Apply additional penalties for high-risk indicators
security_penalty = 0
if len(features) >= 31:  # If we have enhanced features
    if not domain_exists:
        security_penalty += 15
    if domain_age_score == 1:  # Very new domain
        security_penalty += 10
    if not has_valid_ssl and has_https:
        security_penalty += 8
    if suspicious_redirect:
        security_penalty += 12
    if domain_changed:
        security_penalty += 15
    if phishing_keyword_count > 2:
        security_penalty += 10
    if has_suspicious_tld:
        security_penalty += 8

confidence_score = round(max(0, base_confidence - security_penalty), 2)

# Classification logic based on hybrid confidence
if confidence_score >= 70:
    result = "Legitimate"
    remark = "Safe to visit, but always verify the source."
elif confidence_score >= 50:
    result = "Suspicious"
    remark = "Proceed with caution. Check the domain name and look for signs of phishing."
else:
    result = "Phishing"
    remark = "Do NOT click. Report the site if possible."
```

Fig. 10(b) *Snippet code for the hybrid consensus combining SVM and rule score*

5.4 Application Programming Interface (API) Testing for Domain Validation

The PhishBlocker system includes several RESTful API endpoints to support real-time phishing detection. A key endpoint is /api/validate_domain, which handles POST requests with a URL payload. This endpoint validates the URL's format and structure, including its length and use of HTTP or HTTPS protocols, as shown in Fig. 11(a). After validation, the domain is evaluated using a hybrid approach that combines machine learning predictions with a rule-based scoring system. Starting from a base score of 100, the system deducts points for over 30 phishing-related features such as newly registered domains, expired SSL certificates, blacklisted subdomains, suspicious keywords, and excessive redirects. The final classification legitimate, suspicious, or phishing is based on the adjusted score and confidence level. The frontend then uses JavaScript to fetch the risk score and validation results, updating the interface in real time with custom messages, as illustrated in Fig. 11(b). This visual feedback helps users make informed decisions by displaying outcomes like "Domain appears legitimate" or "High risk," depending on domain age and SSL status. Examples of these classification results are shown in Fig. 11(c) to 11(f), covering trusted domains like Google and Bing, as well as suspicious and high-risk cases.



```python
# API endpoint for real-time domain validation
@app.route("/api/validate_domain", methods=["POST"])
def api_validate_domain():
    """API endpoint for real-time domain validation"""
    if "user_id" not in session:
        return jsonify({"error": "Unauthorized"}), 401

    data = request.get_json()
    if not data or 'url' not in data:
        return jsonify({"error": "URL is required"}), 400

    url = data['url']

    try:
        # Enhanced URL format validation
        if not url or len(url.strip()) < 8:
            return jsonify({
                "valid": False,
                "message": "URL is too short or empty",
                "status": "error"
            })

        # Check for basic URL format
        if not re.match(r"^https?://", url, re.IGNORECASE):
            return jsonify({
                "valid": False,
                "message": "URL must start with http:// or https://",
                "status": "error"
            })
```

Fig. 11(a) *Code Snippet performing server-side URL and format validation*

```javascript
function validateDomain(url) {
  console.log('Validating domain for URL:', url); // Debug log

  fetch('/api/validate_domain', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify({ url: url })
  })
  .then(response => {
    console.log('API response status:', response.status); // Debug log
    if (!response.ok) {
      throw new Error(`HTTP error! status: ${response.status}`);
    }
    return response.json();
  })
  .then(data => {
    console.log('API response data:', data); // Debug log

    if (data.valid) {
      isValidDomain = true;
      scanBtn.disabled = false;

      let message = data.message;
      let status = 'valid';

      // Handle different risk levels for valid domains
      if (data.risk_level === 'low' && data.risk_score > 0) {
        status = 'caution';
        message += ` (Risk Score: ${data.risk_score}%)`;
      }
```

Fig. 11(b) *Code Snippet fetch function for calling the validation API*

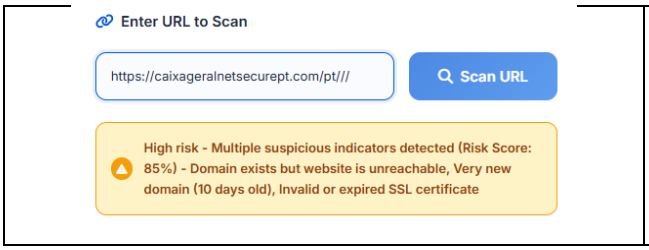


Fig. 11(c) *Output for high-risk domain with multiple indicators detected*

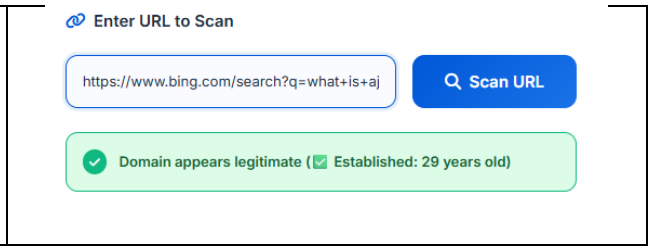


Fig. 11(d) *Output for legitimate domains*

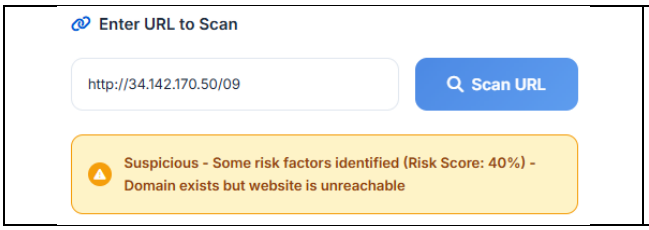


Fig. 11(e) *Output for suspicious domain that is unreachable*

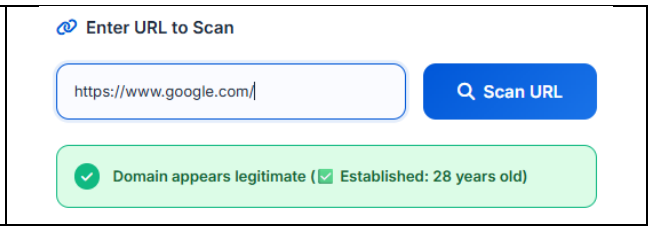


Fig. 11(f) *Output for legitimate domains*

5.5 Security Checklist Result

Table 13 presents the results of a security checklist used to assess the security properties implemented in the PhishBlocker system. The system meets all the listed criteria, demonstrating its commitment to ensuring secure access, protecting sensitive user data, and enforcing proper access control. The checklist reflects practical implementations such as hashed password storage, session control, form validation, and role-based restrictions.

Table 13 *Security Checklist result for PhishBlocker*

No	Checklist	Actual Result
1	Passwords are securely hashed using generate_password_hash() before storage	Pass
2	Access to protected pages is restricted by using proper access control for authentication	Pass
3	Limit role-based access ensures administrative features are only accessible to admins	Pass
4	Sensitive user information is handled securely, with no plain-text data stored at rest	Pass
5	All user inputs are validated and protected with CSRF tokens to prevent attacks	Pass
6	User sessions are properly terminated to prevent misuse	Pass

5.6 User Acceptance Testing (UAT) Result

A User Acceptance Testing (UAT) form was used to evaluate PhishBlocker's functionality from both the admin and user perspectives. As shown in Table 14, responses were collected from 15 participants through a structured Google Form consisting of yes-or-no questions. The respondents came from diverse backgrounds, including Computer Science students, FSKTM students, Information Technology students, Engineering Technology students, general students, as well as a professional and a domestic manager. All UAT criteria were successfully met by the system. Appendix A provides details of the data collection method.

Table 14 *User Acceptance Testing (UAT) result for PhishBlocker*

Description	Expected Result	Actual Result
Admin login with valid credentials	Admin access the dashboard without errors	Pass
Admin manages users on Manage Users page	Admin manages users accounts without errors	Pass
Admin can access the archive URLs in the Archive URL page	Admin manages archive URLs without errors	Pass
Admin can download full CSV from the Archive URL page	Admin able to download full CSV file without errors	Pass
Admin updates profile on Profile page	Profile updates successfully	Pass
Admin change password on Profile page	Admin changes password successfully	Pass
Admin logs out from the system	Admin terminates session without errors	Pass
User registers and logs in with valid information	User registers and logs in successfully	Pass
User password reset via Forgot Password page	User resets password successfully	Pass
Users submit URL in the Scan URL page	User scanned URL successfully	Pass
User views the report on the Scan URL page	The detailed report shown successfully	Pass
User updates profile information on Profile page	Profile updates successfully	Pass
User change their password on Profile page	Passwords change successfully	Pass
User views their scan history and archive URLs both on Scan History and Archive URL page	The archive URLs shown all the URL scanned successfully	Pass
User logs out from the system	User terminates session without errors	Pass

6. Conclusion

This project developed a phishing detection system called PhishBlocker, which uses a combination of Support Vector Machine (SVM) and rule-based scoring to identify whether a URL is legitimate or a phishing attempt. By looking at important URL features such as length, subdomains, special characters, and domain age, the system can catch patterns commonly used in phishing. Built using an object-oriented approach, the system was designed in clear phases, starting from planning and design to implementation and testing, making it easy to maintain and improve. A cleaned and balanced dataset from Kaggle and GitHub was used to train the SVM model, helping the system achieve 93% accuracy while reducing false alarms compared to traditional blacklist or basic rule-based tools. The platform includes useful features like URL validation, detailed result reports, archived history, and admin controls, all within a clean and user-friendly interface. Based on the testing and feedback, PhishBlocker proved to be reliable, effective, and ready to support safer online browsing.

Acknowledgement

The authors would like to thank the Faculty of Computer Science and Information Technology, Universiti Tun Hussein Onn Malaysia for its support.

Conflict of Interest

Authors declare that there is no conflict of interest regarding the publication of the paper.

Author Contribution

The authors confirm contribution to the paper as follows: ***study conception and design:*** *M. I. Noor Izzri, I. R. A. Hamid;* ***data collection:*** *M. I. Noor Izzri, I. R. A. Hamid;* ***analysis and interpretation of results:*** *M. I. Noor Izzri, I. R. A. Hamid;* ***draft manuscript preparation:*** *M. I. Noor Izzri, I. R. A. Hamid. All authors reviewed the results and approved the final version of the manuscript.*

References

[1] M. M. Elsheh and K. Swayeb, "Phishing Website Detection Using a Hybrid Approach Based on Support Vector Machine and Ant Colony Optimization," in *Proceeding - 2023 IEEE 3rd International Maghreb Meeting of the Conference on Sciences and Techniques of Automatic Control and Computer Engineering, MI-STA 2023*, Institute of Electrical and Electronics Engineers Inc., 2023, pp. 402–406. doi: 10.1109/MI-STA57575.2023.10169464.

[2] A. Dawabsheh, M. Jazzar, A. Eleyan, T. Bejaoui, and S. Popoola, "An Enhanced Phishing Detection Tool Using Deep Learning From URL," in *2022 International Conference on Smart Applications, Communications and Networking, SmartNets 2022*, Institute of Electrical and Electronics Engineers Inc., 2022. doi: 10.1109/SmartNets55823.2022.9993984.

[3] APWG, "Phishing E-mail Reports and Phishing Site Trends 4 Brand-Domain Pairs Measurement 5 Brands & Legitimate Entities Hijacked by E-mail Phishing Attacks 6 Use of Domain Names for Phishing 7-9 Phishing and Identity Theft in Brazil 10-11 Most Targeted Industry Sectors 12 APWG Phishing Trends Report Contributors 13." [Online]. Available: http://www.apwg.org,

[4] R. Yang, K. Zheng, B. Wu, C. Wu, and X. Wang, "Phishing Website Detection Based on Deep Convolutional Neural Network and Random Forest Ensemble Learning," *Sensors*, vol. 21, no. 24, p. 8281, Dec. 2021, doi: 10.3390/s21248281.

[5] D. Tao, D. Jinran, X. Aidong, X. Chuanmao, L. Boyu, and H. Chao, "Phishing Detection System Based on the SVM Active Learning Algorithm," in *2023 3rd International Conference on Smart Generation Computing, Communication and Networking, SMART GENCON 2023*, Institute of Electrical and Electronics Engineers Inc., 2023. doi: 10.1109/SMARTGENCON60755.2023.10442623.

[6] A. K. Putri, J. Wiratama, S. A. Sanjaya, S. F. Wijaya, M. E. Johan, and A. Faza, "Web URLs Phishing Detection Model with Random Forest Algorithm," in *2024 5th International Conference on Big Data Analytics and Practices, IBDAP 2024*, Institute of Electrical and Electronics Engineers Inc., 2024, pp. 1–5. doi: 10.1109/IBDAP62940.2024.10689685.

[7] S. Dalvi, G. Gressel, and Dr. K. Achuthan, "Tuning the False Positive Rate / False Negative Rate with Phishing Detection Models," *Int J Eng Adv Technol*, vol. 9, no. 1s5, pp. 7–13, Dec. 2019, doi: 10.35940/ijeat.A1002.1291S519.

[8] R. Ferdaws and N. E. Majd, "Phishing URL Detection Using Machine Learning and Deep Learning," in *2024 IEEE 5th World AI IoT Congress, AIIoT 2024*, Institute of Electrical and Electronics Engineers Inc., 2024, pp. 485–490. doi: 10.1109/AIIoT61789.2024.10579005.

[9] A. Prasad and S. Chandra, "PhiUSIIL: A diverse security profile empowered phishing URL detection framework based on similarity index and incremental learning," *Comput Secur*, vol. 136, p. 103545, Jan. 2024, doi: 10.1016/j.cose.2023.103545.

[10] Z. Alkhalil, C. Hewage, L. Nawaf, and I. Khan, "Phishing Attacks: A Recent Comprehensive Study and a New Anatomy," Mar. 09, 2021, *Frontiers Media S.A.* doi: 10.3389/fcomp.2021.563060.

[11] S. Das Guptta, K. T. Shahriar, H. Alqahtani, D. Alsalman, and I. H. Sarker, "Modeling Hybrid Feature-Based Phishing Websites Detection Using Machine Learning Techniques," *Annals of Data Science*, vol. 11, no. 1, pp. 217–242, Feb. 2024, doi: 10.1007/s40745-022-00379-8.

[12] R. O. Akinyede and J. A. Adelakun, "Detection and Prevention of Phishing Attack Using Linkguard Algorithm," *Journal of Information*, vol. 4, no. 1, pp. 10–23, 2018, doi: 10.18488/journal.104.2018.41.10.23.

[13] Q. A. Al-Haija and A. Al Badawi, "URL-based Phishing Websites Detection via Machine Learning," in *2021 International Conference on Data Analytics for Business and Industry, ICDABI 2021*, Institute of Electrical and Electronics Engineers Inc., 2021, pp. 644–649. doi: 10.1109/ICDABI53623.2021.9655851.

[14] GeeksForGeeks, "Components of a URL." Accessed: Dec. 02, 2024. [Online]. Available: https://www.geeksforgeeks.org/components-of-a-url/

[15] S. Das Guptta, K. T. Shahriar, H. Alqahtani, D. Alsalman, and I. H. Sarker, "Modeling Hybrid Feature-Based Phishing Websites Detection Using Machine Learning Techniques," *Annals of Data Science*, vol. 11, no. 1, pp. 217–242, Feb. 2024, doi: 10.1007/s40745-022-00379-8.

[16] R. Yang, K. Zheng, B. Wu, C. Wu, and X. Wang, "Phishing Website Detection Based on Deep Convolutional Neural Network and Random Forest Ensemble Learning," *Sensors*, vol. 21, no. 24, p. 8281, Dec. 2021, doi: 10.3390/s21248281.
[17] M. Nanda and S. Goel, "URL based phishing attack detection using BiLSTM-gated highway attention block convolutional neural network," *Multimed Tools Appl*, vol. 83, no. 27, pp. 69345–69375, Aug. 2024, doi: 10.1007/s11042-023-17993-0.
[18] D. Izzah, A. A. Patdeli, I. Rahmi, and A. Hamid, "PhishShield: URL Phishing Detection Tool using Machine Learning Algorithm," vol. 5, no. 2, pp. 212–231, 2024, doi: 10.30880/aitcs.2024.05.02.012.
[19] S. V. Simpson, B. P. Gatti, S. Setty, A. Papepalli, and B. Sompalle, "Intelligent URL Analysis for Phishing Threads," in *Proceedings of the 2024 10th International Conference on Communication and Signal Processing, ICCSP 2024*, Institute of Electrical and Electronics Engineers Inc., 2024, pp. 976–981. doi: 10.1109/ICCSP60870.2024.10543918.
[20] K. M. Sudar, M. Rohan, and K. Vignesh, "Detection of adversarial phishing attack using machine learning techniques", doi: 10.1007/s12046-024-02582-0S.
[21] G. Apruzzese and V. S. Subrahmanian, "Mitigating Adversarial Gray-Box Attacks Against Phishing Detectors," *IEEE Trans Dependable Secure Comput*, vol. 20, no. 5, pp. 3753–3769, Sep. 2023, doi: 10.1109/TDSC.2022.3210029.
[22] *ICACCS : 2019 5th International Conference on Advanced Computing & Communication Systems : 15-16 March 2019, Coimbatore, India*. Institute of Electrical and Electronics Engineers, 2019.
[23] J. Al-Sawwa, M. Almseidin, M. Alkasassbeh, K. Alemerien, and R. Younisse, "Spark-based multi-verse optimizer as wrapper features selection algorithm for phishing attack challenge," *Cluster Comput*, vol. 27, no. 5, pp. 5799–5814, Aug. 2024, doi: 10.1007/s10586-024-04272-2.
[24] M. Elsadig *et al.*, "Intelligent Deep Machine Learning Cyber Phishing URL Detection Based on BERT Features Extraction," *Electronics (Basel)*, vol. 11, no. 22, p. 3647, Nov. 2022, doi: 10.3390/electronics11223647.
[25] V. Zeng, S. Baki, A. El Aassal, R. Verma, L. F. T. De Moraes, and A. Das, "Diverse datasets and a customizable benchmarking framework for phishing," in *IWSPA 2020 - Proceedings of the 6th International Workshop on Security and Privacy Analytics*, Association for Computing Machinery, Inc, Mar. 2020, pp. 35–41. doi: 10.1145/3375708.3380313.
[26] S. Jain and C. Gupta, "A Support Vector Machine Learning Technique for Detection of Phishing Websites," in *2023 6th International Conference on Information Systems and Computer Networks, ISCON 2023*, Institute of Electrical and Electronics Engineers Inc., 2023. doi: 10.1109/ISCON57294.2023.10111968.
[27] VirusTotal, "VirusTotal." Accessed: Dec. 12, 2024. [Online]. Available: https://www.virustotal.com/gui/home/url
[28] Bolster, "CheckPhish."
[29] IsItHacked, "IsItHacked." Accessed: Dec. 22, 2024. [Online]. Available: https://www.isithacked.com/

Appendix A: User Acceptance Testing (UAT) Result for Admin and User Side Modules

Fig. A.1 shows the result of user acceptance testing for admin side modules.

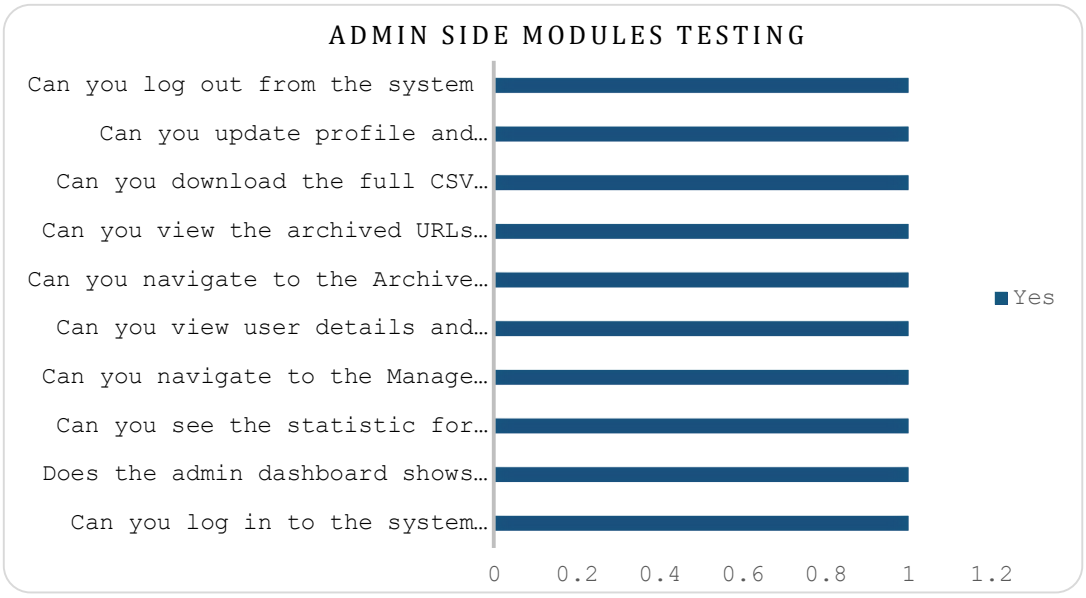


Fig A.1 *The UAT Result of Admin Side*

Fig. A.2 shows the result of user acceptance testing for user side modules.

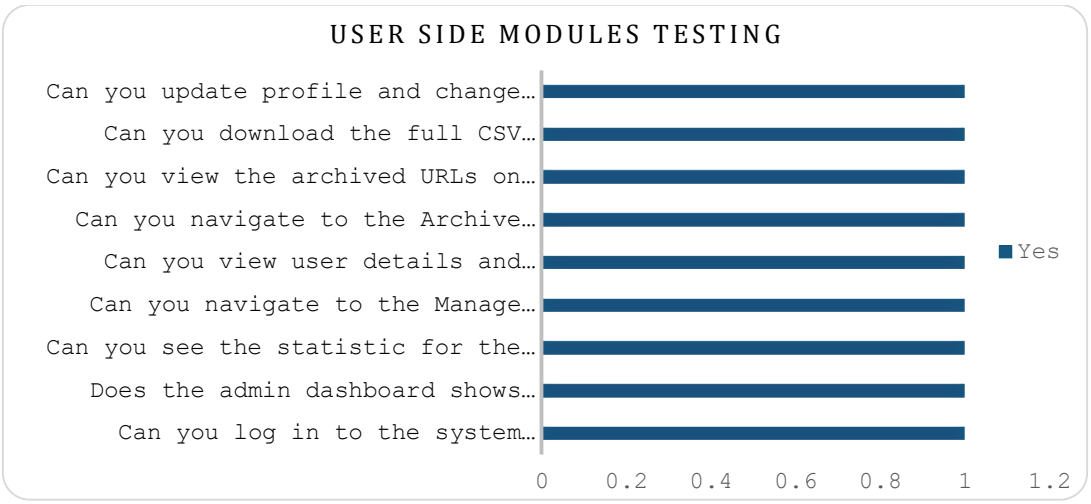


Fig A.2 *The UAT Result of User Side*