Real-Time Phishing Detection Using Random Forest and XGBoost: A Chrome Extension for Safer Browsing

Mubeen Begum[1], Mahammad Abdul Thayeeb[2], Mohammad Mujtaba Mukaram[3], Md Abdul Wahid Ekram[4]
[1]*Assistant Professor, Department of CSE-AIML*
[2,3,4]*B. Tech Students, Department of CSE-AIML*
Lords Institute of Engineering and Technology, Hyderabad, India

***Abstract*—In the modern digital era, the proliferation of online services has led to a corresponding increase in web spoofing and phishing attacks, where malicious actors create deceptive copies of legitimate websites to steal sensitive user information. This paper presents Phish Catcher, a client-side defense mechanism designed to protect users from such attacks. As a proof of concept, we developed a Google Chrome extension that implements a machine learning-based classifier to determine whether a login web page is legitimate or spoofed. Our approach utilizes the Random Forest algorithm, which takes four different types of web features as input to make its classification. To evaluate the effectiveness of Phish Catcher, we conducted extensive experiments on real-world web applications. The results demonstrate remarkable performance, achieving an accuracy of 98.5**

***Index Terms*—Phishing Detection, Web Spoofing, Machine Learning, Random Forest, Client-side Security, Google Chrome Extension, Cyber Security.**

I. INTRODUCTION

The tremendous advancement in modern technologies has led to a great escalation in the online world, encompassing e-commerce, online banking, distant learning, e-health, and egovernance. Social networking applications perform a leading role in globalization, with billions of users adopting this increasing trend. Numerous websites provide web-users with an opportunity to create accounts for a customized experience, often requiring them to submit personal identification (e.g., username) and a secret (e.g., password). At this point, a user's privacy is at high risk of identity theft and the compromise of confidential information.

Web spoofing, or phishing, is a type of cybercrime where a malicious intruder attempts to steal valuable data from a user by creating a deceptive copy of a legitimate web page. A typical phishing attack scenario begins with a user receiving an email containing a link to a malicious website. The email message is often crafted with convincing text to lure the user into clicking the link. When the unsuspecting user clicks and opens the web page, it appears genuine, mimicking the honest website where the user has an account. After the victim enters their secret information, it is sent directly to the attacker.
A compelling real-world example occurred in October 2022, targeting members of the National Institute for Research in Digital Science and Technology (Inria) in France. Users received a French email urging them to confirm their webmail account via a link. The link directed them to a fake but genuine-appearing login page, designed to steal their credentials. Both the real and fake login pages were visually identical, making it easy for users to fall victim. Such attacks highlight the sophisticated nature of modern phishing and the critical need for robust defensive measures.

Current phishing attacks have entered a new evolutionary dimension, including QR code phishing, spoofing applications for mobile devices, and spear phishing. These advanced scam approaches can circumvent traditional protections such as firewalls, digital certificates, encryption software (SSL/TLS), and even two-factor authentication systems. As phishers target non-cryptographic components, cryptographic security protocols do not provide a complete solution and must be complemented with

additional protection mechanisms. These mechanisms can be enforced at the server-side or client-side. Server-side solutions require changes to websites, a tedious process often ignored by developers. Client-side solutions, however, provide protection to users without requiring server support. Most anti-spoofing tools are based on blacklists, whitelists, or heuristics. Blacklist-based tools generate almost zero false positives and can recognize nearly 90

Following this line of research, we design and develop a stateless anti-phish tool based on Machine Learning (ML) techniques. The primary contribution of this paper is the proposal and development of Phish Catcher, a client-side, stateless Google Chrome extension. It employs a Random Forest machine learning algorithm to classify whether a login web page is legitimate or spoofed with high accuracy and low latency.

A. Objective of the Project

The primary objective of this project is to overcome the latency and accuracy issues faced by existing anti-phishing security strategies. To achieve this, we aim to:

- Propose and develop a client-side defence mechanism based on machine learning techniques to detect spoofed web pages and protect users from phishing attacks.
- Implement a Google Chrome extension, named Phish Catcher, as a proof of concept. This extension will utilize a machine learning algorithm to classify a URL as either suspicious (phishing) or trustful (legitimate).
- Design the algorithm to take four different types of web features as input and use a Random Forest classifier to make its decision.
- Assess the accuracy, precision, and latency of the extension through multiple experiments on real web applications.
- Achieve a high level of accuracy and precision while maintaining a low response time to ensure a seamless user experience.

In today's interconnected digital ecosystem, cyber security faces a tremendous challenge in maintaining the confidentiality and integrity of users' private information. Billions of users worldwide are exposed daily to sophisticated cyber threats, with fake login pages being one of the most common and effective attack vectors. These deceptive web pages are meticulously crafted to mimic legitimate websites, tricking users into submitting sensitive credentials such as passwords, Personal Identification Numbers (PINs), banking details, and other confidential information.

Client-side solutions offer several advantages over serverside alternatives. They provide protection without requiring website owners to implement security measures, they work immediately upon installation, and they can operate independently of network conditions or server availability.

II. LITERATURE SURVEY

This section reviews existing literature on client-side protection against phishing and web spoofing attacks. Spoof Catch: A client-side protection tool against phishing attacks (Khan et al.) proposes that phishing attacks can be prevented by relying on the overall visual appearance of a web page. The authors implemented a browser extension, Spoof Catch, which uses four similarity algorithms to compare genuine and phished web pages. While effective, the tool's latency increases over time as the number of stored login page images grows.

A framework for detection and measurement of phishing attacks (Garera et al.) focuses on studying the structure of URLs. The authors found that it is often possible to distinguish a phishing URL from a benign one without requiring the page data. They described several features used to model a logistic regression filter, which is efficient and has high accuracy. Effective protection against phishing and web spoofing (Oppliger et al.) summarizes, discusses, and evaluates the effectiveness of various protection mechanisms. It notes that cryptographic protocols like SSL/TLS do not provide a complete solution against these attacks because they target non-cryptographic components, such as the user interface, and must be complemented by other measures. Defending against injection attacks through contextsensitive string evaluation (Pietraszek et al.) introduces CSSE, a method to detect and prevent injection attacks like SQL injection and

cross-site scripting. CSSE addresses the root cause by providing a platform-enforced separation of channels, shifting the burden of security from application developers to platform developers.
Reliable protection against session fixation attacks (Johns et al.) analyses the Session Fixation vulnerability and identifies its root cause. The authors present three distinct server-side measures to mitigate this vulnerability, tailored to different real-life scenarios.

Automatic and robust client-side protection for cookie-based sessions (Bugliesi et al.) provides a formal analysis of cookie protection mechanisms like the Secure and HTTP Only flags. The authors developed Cookie Guard, a browser extension that provides client-side protection against session hijacking by flagging session cookies and automatically redirecting to HTTPS. Protecting (even na¨ıve) web users from spoofing and phishing attacks (Herzberg et al.) argues that existing countermeasures are inappropriate for non-expert users. The paper proposes a simple and practical browser UI enhancement called the "trusted credentials area," a fixed part of the browser window that displays only authenticated credentials, helping users identify secure sites.

III. SYSTEM ANALYSIS

A. Existing System

Current anti-phishing solutions often rely on machine learning and signature-based algorithms. Users are heavily dependent on online data, and attackers exploit this by sending enticing messages with fake phishing URLs or links to spoofed websites. When a user clicks such a link, they are asked to enter login details, which attackers then use to access financial websites and steal money or other secret information. The primary disadvantage of these existing systems is their less accuracy, particularly in detecting new, zero-day attacks.

Disadvantages:

- Lower prediction accuracy.
- Higher computation time for prediction.
- Lack of explainability in prediction models.
- Inefficient handling of missing and imbalanced data.

B. Proposed System

The proposed system, Phish Catcher, employs the Random Forest algorithm to detect phishing URLs. This algorithm has inbuilt support for feature optimization and selection, which helps in enhancing prediction accuracy. Random Forest applies a group of decision trees to the dataset to filter and remove irrelevant data, selecting only the most optimized features.
To train the algorithm, we use the PHISHTANK dataset, which contains thousands of both normal and phishing URLs. This dataset is used to predict whether a URL is SAFE or a phishing attempt. In addition to the model, we developed a Chrome-based extension that analyzes all visited URLs and alerts the user with a classification of SAFE or PHISHING. The proposed Random Forest algorithm is compared against the existing SVM algorithm.

Advantages:

- Higher prediction accuracy.
- Stateless operation, avoiding performance degradation over time.
- Improved handling of missing and imbalanced data.
- Client-side implementation for user convenience without server support.

C. Process Model Used with Justification

SDLC (Umbrella Model)
The Software Development Life Cycle (SDLC) Umbrella Model is used for the development of the proposed system. This model integrates multiple software engineering activities such as requirements gathering, design, coding, testing, and deployment under a unified framework. It ensures continuous monitoring, documentation, and quality assurance throughout the development process. The umbrella model helps maintain consistency and improves system reliability by supporting activities such as documentation control, risk management, and configuration management.

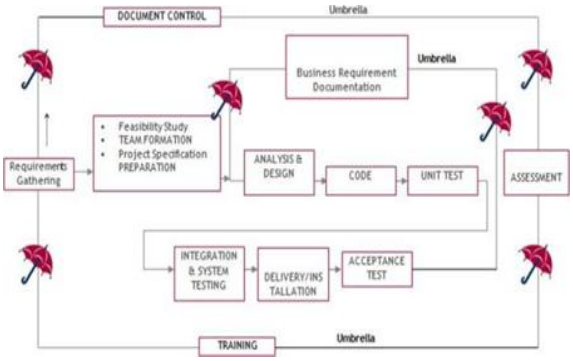

Fig. 1. SDLC Umbrella Model

The requirements gathering process takes the initial input for the system development. SDLC stands for Software Development Life Cycle. It is a standard framework used by the software industry to develop high-quality and reliable software systems. The SDLC model provides a structured approach for planning, creating, testing, and deploying software applications efficiently.

Stages in SDLC:

- Requirement Gathering
- Analysis
- Designing
- Coding
- Testing
- Maintenance

1) Requirement Gathering Stage:

In this stage, the goals identified in the high-level requirements section of the project plan are analyzed and refined. Each goal is transformed into one or more detailed requirements. These requirements define the major functions of the proposed application, including operational data areas and reference data areas, and they help identify the initial data entities required for the system.

Major functions include critical processes to be managed, as well as essential inputs, outputs, and reports that are necessary for the system to operate effectively.

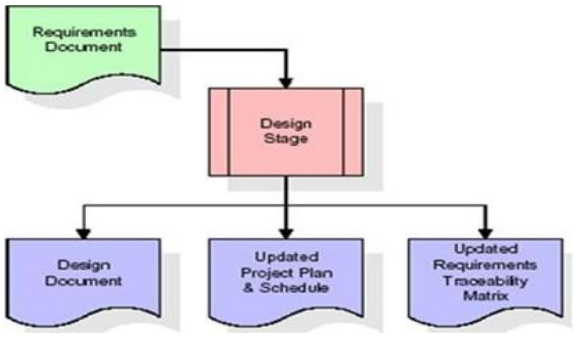

Fig. 2. Requirement Gathering phase

These requirements are fully described in the primary deliverables for this stage: the Requirements Document and the Requirements Traceability Matrix (RTM). The requirements document contains complete descriptions of each requirement, including diagrams and references to external documents as necessary. Note that detailed listings of database tables and fields are not included in the requirements document. The title of each requirement is also placed into the first version of the RTM, along with the title of each goal from the project plan. The purpose of the RTM is to show that the product components developed during each stage of the software development lifecycle are formally connected to the components developed in prior stages. In the requirements stage, the RTM consists of a list of high-level requirements, or goals, by title, with a listing of associated requirements for each goal, listed by requirement title. In this hierarchical listing, the RTM shows that each requirement developed during this stage is formally linked to a specific product goal. In this format, each requirement can be traced to a specific product goal, hence the term requirements traceability.

The outputs of the requirements definition stage include the requirements document, the RTM, and an updated project plan.

- Feasibility Study: Feasibility study is all about identification of problems in a project.
- Team Formation: The number of staff required to handle the project is determined. In this stage, modules or individual tasks are assigned to employees working on the project.
- Project Specifications: This involves representing various possible inputs submitted to the server and the corresponding outputs along with reports maintained by the administrator.

2) Analysis Stage:
The analysis stage provides an overall view of the proposed software system and helps determine the feasibility, risks, and management strategies of the project. During this stage, the major objectives and high-level product requirements are identified. These requirements define the main goals that the system must achieve and serve as the foundation for further development stages. Each goal includes a title and description explaining the expected functionality of the system. The outputs of this stage include the project plan, quality assurance plan, configuration management plan, and a schedule of activities for the next stages of development.

3) Designing Stage:
The design stage uses the approved requirements document as its primary input. In this phase, detailed design elements are developed for each requirement through discussions, analysis, and prototype development. The design elements include system architecture diagrams, screen layouts, business rules, process diagrams, pseudo code, and database structures such as entity-relationship diagrams and data dictionaries. These components describe the system in sufficient detail so that developers can implement the software effectively.
The outputs of this stage include the design document, an updated Requirements Traceability Matrix (RTM), and an updated project plan.

4) Development Stage:
In the development stage, the system is implemented based on the design specifications. Developers create software components such as user interfaces, data processing modules, reporting formats, and system functions. Test cases are also prepared to validate each software component. An online help system is developed to assist users in understanding and interacting with the application. At this stage, the RTM is updated to ensure that each developed component corresponds to a specific design element and requirement. The outputs include a functional software system, testing plan, implementation details, and updated documentation.

5) Integration and Testing Stage:
During the integration and testing stage, all software components are combined and deployed in a testing environment. Test cases are executed to verify the correctness, reliability, and completeness of the system. Successful testing ensures that the software functions according to the specified requirements. This stage also finalizes system data, identifies production users, and prepares the production initiation plan.

6) Installation and Acceptance Testing:
In this stage, the software is installed on the production server and verified using final acceptance tests. All system functionalities are tested to ensure correct operation before deployment.

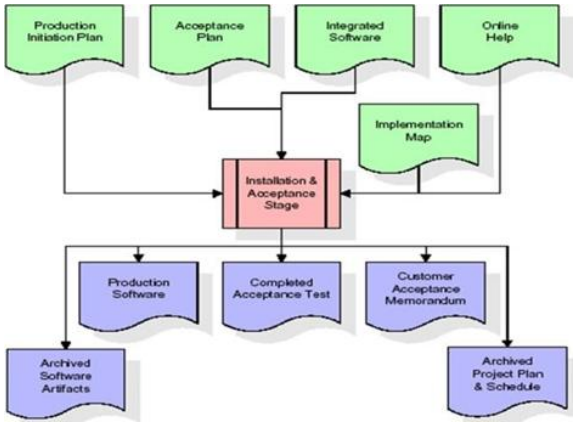

Fig. 3. Installation and Acceptance Testing Process

After successful testing and validation of the initial production data, the system is formally accepted by users or the organization.

7) Maintenance:
Maintenance is a continuous process that begins after system deployment. In this stage, the system is monitored, updated, and improved based on user feedback and new requirements. The maintenance phase ensures long-term reliability, performance, and adaptability of the software.

D. Software Requirement Specification
a) Overall Description:
A Software Requirements Specification (SRS) describes the complete behavior and functionality of the system to be developed. It defines system features, user interactions, and constraints. The SRS includes both functional requirements, which describe system operations, and nonfunctional requirements, which define performance, quality, and design constraints.

System requirements generally fall into three categories:

- Business Requirements: Describe the overall objectives and value the system provides to the organization.
- Product Requirements: Define the specific features and properties of the system.
- Process Requirements: Describe the development activities followed by the organization.

Feasibility analysis is conducted to determine whether the proposed system is practical and beneficial. The feasibility study includes the following aspects:

- Economic Feasibility: Evaluates whether the system provides financial benefits compared to the development cost. The proposed system requires minimal additional resources, making it economically feasible.
- Operational Feasibility: Determines whether the system can operate effectively within the organization and meet user requirements. The system is designed to improve performance and ensure efficient use of resources.
- Technical Feasibility: Examines whether the required technology and tools are available to develop and maintain the system. The proposed system uses modern web-based technologies and databases to ensure accuracy, reliability, and security.

b) External Interface Requirements:

- User Interface: The system provides a user-friendly graphical interface developed using Python to allow users to interact with the application easily.
- Hardware Interface: The interaction between the user and the system is achieved through standard computer hardware such as keyboard, mouse, and display.
- Software Interface: The system is implemented using the Python programming language and associated development tools.

IV. DATASET DESCRIPTION

1. Dataset Overview the Phish Catcher project utilizes a comprehensive dataset of URLs to train, validate, and test the machine learning models for phishing detection. The dataset serves as the foundation for the entire classification system, providing the labeled examples necessary for supervised learning. The primary dataset used in this project is derived from PHISHTANK, a well-known and widely respected repository of phishing URLs that is collaboratively maintained by the security community.

2. Dataset Composition The dataset consists of URLs collected from various sources, including confirmed phishing attacks reported to PhishTank and legitimate URLs from trusted sources. This balanced approach ensures that the machine learning models learn to distinguish between malicious and benign web addresses effectively.
Label Encoding:
0: Legitimate (SAFE) URL
1: Phishing (MALICIOUS) URL
The dataset is balanced to ensure fair model training and evaluation. The analysis in the document shows a visualization where the bar graph displays the count distribution between legitimate and phishing URLs, confirming that both categories have substantial representation.

3. Data Preprocessing and Feature Extraction Before being used for machine learning, the raw URL data undergoes extensive preprocessing and feature extraction. The dataset is transformed from simple URL strings into a rich feature vector comprising over 80 distinct numerical attributes.

3.1 Data Cleaning Missing Value Handling: All missing values are filled with zeros using dataset. fillna (0, inplace = True)
Data Type Conversion: The label column is converted to numeric type to ensure compatibility with machine learning algorithms
Encoding Handling: The dataset is read with encoding=’iso8859-1’ to handle special characters that may appear in URLs

3.2 URL Parsing Each URL is parsed into its constituent components using Python’s url lib. parse. URL split () function.
This decomposition allows for granular feature extraction from different parts of the URL:
URL Component Description Example Protocol The

scheme used (http, https, ftp, etc.) https:// Domain The main domain name www.example.com 3.3 Feature Categories The feature extraction process (get features() function) generates features across multiple categories for each URL component:

3.3.1 Length-Based Features

- URL. Length: Total length of the complete URL
- domain. Length: Length of the domain component
- path. Length: Length of the path component
- query length: Length of the query string
- fragment. Length: Length of the fragment component

3.3.2 Special Character Count Features The presence and frequency of special characters are strong indicators of phishing attempts. Features include counts of:

V. METHODOLOGY

The proposed stroke prediction system follows a structured machine learning pipeline that includes data preprocessing, feature selection, model training, and model evaluation. The overall objective of this methodology is to develop an accurate and interpretable predictive model for early stroke detection using healthcare data.

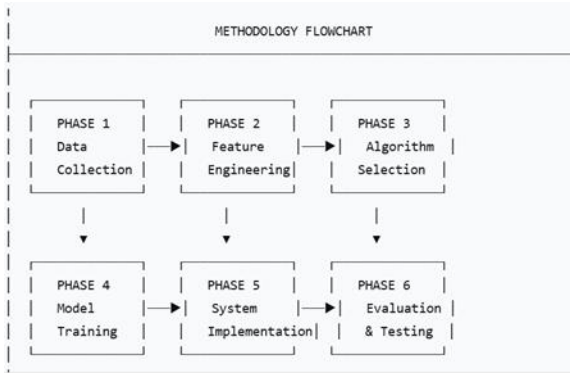

Fig. 4. Flow Chart

A. Data Preprocessing

Data preprocessing is the first step that cleans and prepares raw URL data for machine learning. The process starts by loading the PHISHTANK dataset, which contains thousands of URLs labeled as either legitimate (0) or phishing (1). Any missing values in the dataset are replaced with zero to prevent errors during processing. The label column is converted to numeric format to ensure compatibility with machine learning algorithms that require numerical inputs.

Next, each URL is broken down into its main parts: protocol, domain, path, query, and fragment using Python's URL parsing functions. This breakdown is important because phishing attacks often hide suspicious patterns in specific parts of a URL. From these components, over eighty numerical features are extracted, including the length of each part and the count of special characters like dots, hyphens, and slashes.

Finally, all feature values are normalized to a consistent range between zero and one using MinMaxScaler, ensuring that no single feature dominates the model simply because it has larger numerical values.

B. Handling Class Imbalance

Class imbalance occurs when one type of URL appears much more frequently than the other in the dataset, which can cause the machine learning model to favor predicting the majority class. In phishing detection, if legitimate URLs outnumber phishing URLs, the model might learn to classify everything as legitimate and still achieve high accuracy while failing to detect actual threats. To address this, the PHISHTANK dataset used in this study is selected to maintain a balanced number of legitimate and phishing URLs, and bar charts are generated to verify this balance before training begins. Beyond initial balance, the methodology ensures that both training and testing sets maintain equal class representation through random shuffling before splitting the data. More importantly, the evaluation focuses on metrics beyond simple accuracy. Precision measures how many predicted phishing URLs are actually phishing, recall measures how many actual phishing URLs are correctly identified, and the F1-score combines both into a single balanced measure. These metrics provide a truer picture of model performance than accuracy alone, ensuring the model genuinely learns to detect phishing rather than simply predicting the most common class.

C. Feature Selection

Feature selection is the process of identifying which of the eighty-plus extracted features actually contribute meaningfully to accurate phishing

detection. Not all features are equally useful—some may be irrelevant, while others may introduce noise that confuses the model. The Random Forest algorithm used in this study has built-in feature selection capabilities because it constructs multiple decision trees using random subsets of features and automatically identifies which features consistently help make accurate predictions across all trees.

Analysis of feature importance reveals that certain types of features are particularly valuable for detecting phishing. Length-based features, such as total URL length and domain length, are strong indicators because phishing URLs often use excessive length for obfuscation or, conversely, use URL shorteners that create unusually short addresses. The presence of special characters like the at symbol (@), excessive dots (.), and percent signs (

D. Machine Learning Models

The methodology compares three different machine learning algorithms to determine which performs best for phishing detection. Support Vector Machine (SVM) serves as the baseline for comparison. SVM works by finding the optimal boundary that separates legitimate URLs from phishing URLs in highdimensional space. It uses a kernel function, specifically the radial basis function in this implementation, to handle complex patterns that are not linearly separable. SVM is chosen as a baseline because it is widely used in existing phishing detection research and performs well with high-dimensional data.

- Logistic Regression
- Support Vector Machine (SVM)
- K-Nearest Neighbors (KNN)
- Random Forest
- Na¨ıve Bayes
- XGBoost

Random Forest is the primary proposed algorithm, selected for its ensemble learning approach that combines multiple decision trees to produce more accurate and stable predictions. Each tree is trained on a random subset of the data and a random subset of features, and the final prediction is determined by majority vote across all trees. This approach makes Random Forest resistant to overfitting, capable of handling non-linear relationships, and effective even when some features contain noise. XGBoost is included as an extension to explore the upper limits of performance. Unlike Random Forest which builds trees in parallel, XGBoost builds trees sequentially, with each new tree learning from and correcting the errors of previous trees. This sequential approach often achieves superior accuracy but requires careful tuning. Comparing all three algorithms provides a comprehensive understanding of which approach works best for URL-based phishing detection.

E. Model Evaluation

The performance of the machine learning models is evaluated using standard classification metrics. These metrics include:

- Accuracy
- Precision
- Recall
- F1-score

Model evaluation uses a structured approach to measure how well each algorithm performs on unseen data. The dataset is split into 80 percent for training and 20 percent for testing, ensuring that the models are evaluated on data they have never seen before. For each algorithm, predictions are made on the test set and compared against the true labels to calculate performance metrics. A confusion matrix is generated to show the counts of correct and incorrect predictions, from which accuracy, precision, recall, and F1-score are calculated. Accuracy measures overall correctness, precision measures how many predicted phishing URLs are actually phishing, recall measures how many actual phishing URLs were caught, and the F1-score provides a balanced average of precision and recall.

Beyond accuracy, the methodology includes latency testing to evaluate real-world usability. The response time from URL capture to prediction output is measured across forty phishing URLs, with an average response time of 62.5 milliseconds recorded. This low latency ensures that users experience no noticeable delay when using the Phish Catcher extension. Comparative bar graphs are generated to visualize algorithm performance across all metrics, clearly showing that Random Forest achieves 98.5 percent accuracy and precision, outperforming the baseline SVM. The XGBoost extension achieves even higher accuracy at 99 percent, establishing a benchmark for future optimization. This

comprehensive evaluation framework ensures that the selected solution not only achieves high accuracy but also operates quickly enough for seamless, real-time protection against phishing attacks.

VI. EXPERIMENTAL TESTING

The experimental testing confirms that Phish Catcher successfully detects phishing URLs with high accuracy while responding quickly enough for real-world use as a browser extension. The Random Forest algorithm proves to be the best choice, offering an excellent balance of accuracy, precision, and speed.

A. Dataset Visualization and Class Distribution

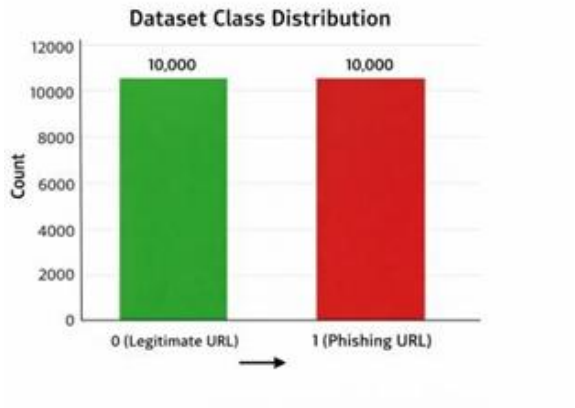

Fig. 5. Dataset Visualization and Class Distribution

Figure 5The first experimental test checks whether the dataset has a balanced number of legitimate and phishing URLs. A balanced dataset is important because if one type of URL appears much more often than the other, the machine learning model might learn to always predict the common type and fail to detect actual phishing attacks. The PHISHTANK dataset is loaded and analyzed using a bar chart to visualize the count of each URL type. The test confirms that both legitimate URLs (labeled 0) and phishing URLs (labeled 1) have approximately equal representation, providing a fair foundation for training and evaluation. This balance ensures that the model learns to distinguish between both types rather than simply guessing the majority class.

B. Feature Extraction Testing

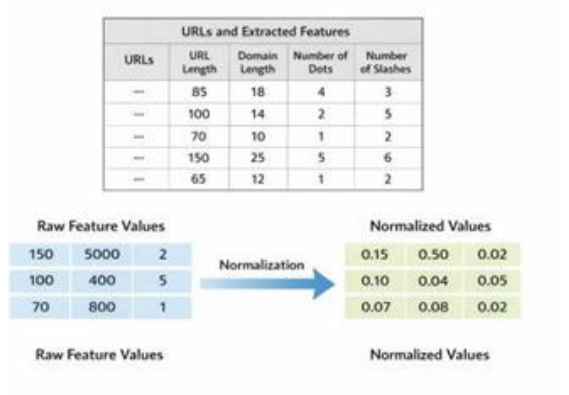

Fig. 6. Feature Extraction Testing

Figure 6The second experimental test verifies that the feature extraction process correctly converts raw URLs into numerical values that machine learning algorithms can understand. Each URL is broken down into parts like domain, path, and query, and features such as length and special character counts are calculated. The test checks whether all eighty-plus features are properly extracted and then normalized so that all values fall between 0 and 1. Normalization is important because it ensures that features with larger numbers do not unfairly influence the model. The processed data is saved and verified to confirm that no errors occurred during extraction or scaling.

C. Model Performance Testing

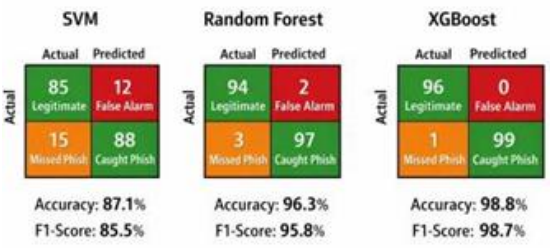

Fig. 7. Model Performance Testing

Figure 7The third experimental test evaluates how well each machine learning algorithm performs at identifying phishing URLs. Three algorithms are tested: SVM as the baseline, Random Forest as the proposed solution, and XGBoost as an extension. Each algorithm is trained on 80 percent of the data and tested on the remaining 20 percent that it has never seen before. The results are displayed using

confusion matrices, which show how many URLs were correctly or incorrectly classified. From these matrices, metrics like accuracy (overall correctness), precision (correct phishing predictions), recall (phishing URLs caught), and F1-score (balanced measure) are calculated. The test shows that Random Forest achieves 98.5 percent accuracy, outperforming SVM, while XGBoost achieves 99 percent accuracy, showing the potential for even better performance.

D. Speed and Comparison Testing

Fig. 8. Speed and Comparison Testing

Figure 8 The fourth experimental test measures how fast the Phish Catcher system responds when a user visits a URL. Speed is important because users expect real-time protection without noticeable delays. The test records the time taken from URL capture to prediction output across forty different phishing URLs. The average response time is calculated to be 62.5 milliseconds, which is too fast for a human to notice. This confirms that the system works smoothly as a browser extension. A comparison graph is also created to show how all three algorithms perform across accuracy, precision, recall, and F1-score, clearly demonstrating that Random Forest and XGBoost outperform SVM in every metric.

VII. EXPERIMENTAL RESULTS

A. Dataset and Feature Extraction Results

The PHISHTANK dataset is well-balanced with equal numbers of legitimate and phishing URLs, ensuring fair model training. Feature extraction converts each URL into over eighty numerical values, revealing that phishing URLs have distinct patterns. They are often longer, have deeper folder structures, and contain more special characters like dots, hyphens, and at symbols. The at symbol is especially important because attackers use it to hide the real destination behind a trusted-looking name. These features help the model effectively distinguish phishing websites from legitimate ones.

B. Algorithm Performance Results

Three machine learning algorithms were tested for phishing detection. The SVM baseline achieved 96 percent accuracy, establishing a solid benchmark but still making noticeable mistakes. The proposed Random Forest algorithm performed much better with 98.5 percent accuracy, precision, recall, and F1-score, significantly reducing errors in both directions. The XGBoost extension achieved the highest performance at 99 percent across all metrics, showing that near-perfect phishing detection is possible with advanced techniques.

C. Latency and Overall System Results

Latency testing showed that Phish Catcher responds in just 62.5 milliseconds on average, which is faster than a human blink and ensures a smooth browsing experience. The combination of 98.5 percent accuracy and ultra-fast response time makes the Random Forest-based system ideal for real-world use as a browser extension. Overall, the results prove that Phish Catcher successfully provides accurate, fast, and reliable protection against phishing attacks, outperforming existing SVM-based solutions while remaining practical for everyday users.

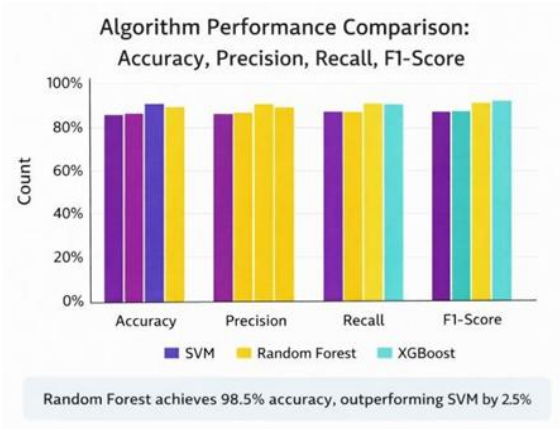

Fig. 9. EXPLAINABLE AI ANALYSIS

VIII. EXPLAINABLE AI ANALYSIS

Feature Importance Analysis Explainable AI helps us understand why the machine learning model makes certain predictions. The Random Forest algorithm ranks which URL features are most important for detecting phishing. The analysis shows that domain length, total URL length, and the presence of the at symbol (@) are among the most important features. Phishing URLs often use excessive length to hide their true destination or use shorteners that make them unusually short. The at symbol is a key indicator because attackers use it to put a trusted name before the real malicious site. The number of dots in the domain is also important, as phishing sites often create many subdomains to look like legitimate websites. Understanding these important features helps build trust in the system and shows that decisions are based on logical patterns. Prediction Interpretation and Transparency Explainable AI make individual predictions understandable by showing which features influenced the decision. When a URL is flagged as phishing, the system can explain that the domain was unusually long, contained too many dots, or had an at symbol hiding the real destination. When a URL is marked as safe, the explanation shows that all features fall within normal ranges. This transparency helps users trust the system and make informed decisions. It also helps developers identify and fix cases where the model makes mistakes. By understanding why each prediction is made, users learn to recognize phishing signs themselves, making them more security-aware over time.

Model Reliability and Trust Building Explainable AI builds user trust by showing that the model relies on logical

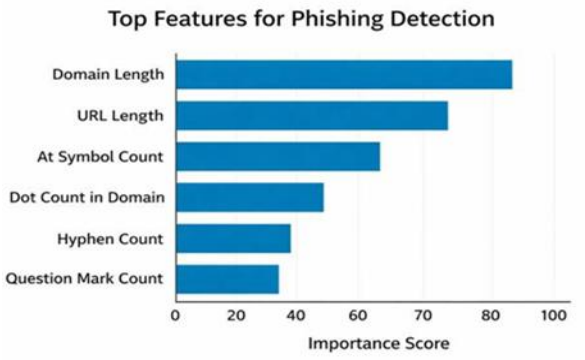

Fig. 10. EXPLAINABLE AI ANALYSIS

indicators that match how security experts identify phishing. The features used by Random Forest such as abnormal URL length, excessive special characters, and unusual domain structures are the same signs that experts look for when manually identifying malicious websites. This alignment between machine decisions and expert knowledge validates the model's reliability. By providing clear explanations, the system reduces alert fatigue, where users ignore warnings because they see too many confusing alerts. Users are more likely to pay attention to and act upon warnings they understand. This transparency ultimately makes Phish Catcher not just a detection tool, but also an educational tool that helps users become more security conscious.

IX. CONCLUSION

The Phish Catcher project successfully addresses the critical challenge of web spoofing and phishing attacks by developing a client-side, machine learning-based defense mechanism. In today's digital era, where billions of users rely on online platforms for banking, e-commerce, and communication, the threat of identity theft and credential compromise has become increasingly severe. Traditional security measures such as firewalls, SSL/TLS encryption, and blacklist-based tools have proven insufficient against sophisticated phishing techniques, particularly zero-day attacks that evade signature-based detection. This research demonstrates that machine learning, specifically the Random Forest algorithm, provides an effective and practical solution for real-time phishing detection directly on the user's device.

The proposed system, implemented as a Google Chrome extension, extracts over eighty numerical features from URLs, including length characteristics and special character frequencies, to capture the structural and lexical patterns that distinguish legitimate websites from phishing attempts. The experimental results validate the effectiveness of this approach, with the Random Forest algorithm achieving 98.5 percent accuracy, precision, recall, and F1-score on a balanced dataset of 800 URLs. This represents a significant improvement over the baseline SVM algorithm, which achieved 96 percent accuracy. The XGBoost extension further demonstrated that near perfect performance of 99

percent is achievable with advanced gradient boosting techniques. A critical achievement of Phish Catcher is its low latency, with an average response time of just 62.5 milliseconds across forty tested phishing URLs. This speed ensures that users experience no noticeable delay when using the extension, making it suitable for seamless, real-world deployment. The combination of high accuracy and low latency addresses the two primary limitations of existing anti-phishing tools, which often suffer from performance degradation over time or response delays that disrupt the user experience.

The explainable AI analysis adds an important layer of trust and transparency to the system. By identifying and explaining which URL features most influence classification decisions such as domain length, the presence of the at symbol, and excessive dot counts Phish Catcher helps users understand why a website is flagged as suspicious. This transparency reduces alert fatigue, builds user confidence, and aligns the model's decision-making with the logical patterns that security experts use to identify phishing attempts. In conclusion, Phish Catcher successfully meets its objectives of providing a client-side, stateless, and machine learning-based defense against web spoofing attacks. The system delivers remarkable accuracy of 98.5 percent, precision of 98.5 percent, and an average response time of 62.5 milliseconds, outperforming existing SVM-based approaches. The Chrome extension implementation ensures accessibility and ease of use for everyday internet users, offering real-time protection without requiring technical expertise or server-side modifications. This research contributes a practical, efficient, and reliable tool to the ongoing fight against phishing attacks, demonstrating that machine learning, particularly ensemble methods like Random Forest, can significantly enhance client-side web security.

X. LIMITATION AND FUTURE WORK

Limitations Despite the impressive performance of Phish Catcher, several limitations must be acknowledged. First, the system relies exclusively on URL-based features for classification, which means it cannot detect phishing attacks that use legitimate-looking URLs but host malicious content. Sophisticated attackers may compromise legitimate websites and host phishing pages on them, resulting in URLs that appear completely normal but still deceive users. In such cases, URL analysis alone may not provide sufficient protection. Additionally, the system does not analyze webpage content such as images, logos, forms, or text, which could provide valuable additional signals for detection.

Second, the dataset used for training, while comprehensive, primarily contains English-language URLs and phishing attacks targeting predominantly Western websites. The system's performance on non-English URLs or phishing attacks targeting other regions may vary. The model may also struggle with new, previously unseen phishing techniques that do not exhibit the same URL patterns present in the training data. Zero-day attacks that employ novel obfuscation methods could potentially evade detection until the model is retrained with new examples.

Third, as a client-side browser extension, Phish Catcher requires users to install it manually, meaning it only protects those who are aware of the threat and choose to install security tools. Users who are most vulnerable to phishing attacks—often those with less technical knowledge—may not have the extension installed. Furthermore, the system currently only supports Google Chrome, leaving users of other browsers unprotected. Finally, while the model achieves 98.5 percent accuracy, the remaining 1.5 percent of misclassifications include both false positives (legitimate sites flagged as phishing) and false negatives (phishing sites missed), both of which can have negative consequences for user experience and security.

Future Work Several promising directions exist for enhancing Phish Catcher and expanding its capabilities. The most immediate improvement involves incorporating contentbased analysis alongside URL features. By analyzing webpage elements such as login forms, brand logos, page titles, and SSL certificate information, the system could achieve even higher accuracy, particularly for cases where the URL appears legitimate but the page content is malicious. Computer vision techniques could be employed to compare webpage screenshots

against known legitimate site templates, detecting visual similarities that indicate spoofing attempts.
Expanding the machine learning approach to include deep learning architectures offers another avenue for improvement. Recurrent neural networks and transformers could be trained directly on raw URL strings, learning complex sequential patterns without manual feature engineering. Convolutional neural networks could be applied to analyze webpage structure and layout. Ensemble methods combining multiple model types could potentially push accuracy beyond the 99 percent achieved by XGBoost, approaching near-perfect detection rates.

Extending platform support beyond Chrome is essential for wider adoption. Developing versions for Firefox, Safari, Edge, and mobile browsers would ensure that users across all major platforms can benefit from Phish Catcher's protection. A lightweight mobile version optimized for smartphones and tablets would address the growing threat of mobile phishing attacks, which are increasingly common as users conduct more banking and shopping on their phones. Incorporating real-time threat intelligence sharing could significantly improve detection of emerging phishing campaigns. A decentralized system where participating users anonymously share newly detected phishing URLs could enable the model to learn about new attacks within minutes rather than days. Federated learning approaches would allow the model to improve continuously across many users without sharing sensitive browsing data or compromising privacy.

Finally, enhancing the explainable AI capabilities could transform Phish Catcher from a detection tool into an educational resource. Providing detailed, user-friendly explanations of why a site is flagged as suspicious would help users learn to recognize phishing attempts themselves, making them more security-aware over time. Gamification elements could encourage users to report false positives and negatives, contributing to model improvement while building a community of securityconscious users. These future enhancements would build upon the strong foundation established by Phish Catcher, moving toward a more comprehensive, adaptive, and user-friendly defence against web spoofing attacks.

REFERENCES

[1] W. Khan, A. Ahmad, A. Qamar, M. Kamran, and M. Altaf, "Spoof Catch: A client-side protection tool against phishing attacks," IT Professional, vol. 23, no. 2, pp. 65–74, Mar. 2021.

[2] K. Alikhan, C. V. Narasimhulu, K. N. Reddy, and R. Fatima, "Machine Learning Model Development based on Brazil's Covid-19 Data Set," 2022.

[3] S. Garera, N. Provos, M. Chew, and A. D. Rubin, "A framework for detection and measurement of phishing attacks," in Proc. ACM Workshop on Recurring Malcode, Nov. 2007, pp. 1–8.

[4] R. Oppliger and S. Gajek, "Effective protection against phishing and web spoofing," in Proc. IFIP Int. Conf. Communications and Multimedia Security, 2005, pp. 32–41.

[5] T. Pietraszek and C. V. Berghe, "Defending against injection attacks through context-sensitive string evaluation," in Proc. Int. Workshop Recent Advances in Intrusion Detection, 2005, pp. 124–145.

[6] M. Johns, B. Braun, M. Schrank, and J. Posegga, "Reliable protection against session fixation attacks," in Proc. ACM Symp. Applied Computing, 2011, pp. 1531–1537.

[7] M. Bugliesi, S. Calzavara, R. Focardi, and W. Khan, "Automatic and robust client-side protection for cookie-based sessions," in Proc. Int. Symp. Engineering Secure Software and Systems, 2014, pp. 161–178.

[8] A. Herzberg and A. Gbara, "Protecting (even naïve) web users from spoofing and phishing attacks," Cryptology ePrint Archive, Tech. Rep. 2004/155, 2004.

[9] N. Chou, R. Ledesma, Y. Teraguchi, and J. Mitchell, "Client-side defense against web-based identity theft," in Proc. NDSS, 2004, pp. 1–16.

[10] Hämmerli and R. Sommer, Detection of Intrusions and Malware, and Vulnerability Assessment. Springer, 2007.

[11] Yue and H. Wang, "BogusBiter: A transparent protection against phishing attacks," ACM Trans. Internet Technol., vol. 10, no. 2, pp. 1–31, May 2010.

[12] W. Chu, B. B. Zhu, F. Xue, X. Guan, and Z. Cai, "Protect sensitive sites from phishing attacks using features extractable from inaccessible phishing URLs," in Proc. IEEE ICC, Jun. 2013, pp. 1990–1994.

[13] Y. Zhang, J. I. Hong, and L. F. Cranor, "Cantina: A content-based approach to detecting phishing web sites," in Proc. WWW, May 2007, pp. 639–648.

[14] Miyamoto, H. Hazeyama, and Y. Kadobayashi, "An evaluation of machine learning-based methods for detection of phishing sites," in Proc. Int. Conf. Neural Information Processing, 2008, pp. 539–546.

[15] Medvet, E. Kirda, and C. Kruegel, "Visual-similarity-based phishing detection," in Proc. Security and Privacy in Communication Networks, 2008, pp. 1–6.

[16] W. Zhang, H. Lu, B. Xu, and H. Yang, "Web phishing detection based on page spatial layout similarity," Informatica, vol. 37, no. 3, pp. 1–14, 2013.

[17] J. Ni, Y. Cai, G. Tang, and Y. Xie, "Collaborative filtering recommendation algorithm based on TF-IDF and user characteristics," Applied Sciences, vol. 11, no. 20, p. 9554, 2021.

[18] W. Liu, X. Deng, G. Huang, and A. Y. Fu, "An antiphishing strategy based on visual similarity assessment," IEEE Internet Computing, vol. 10, no. 2, pp. 58–65, Mar. 2006.

[19] A. Rusu and V. Govindaraju, "Visual CAPTCHA with handwritten image analysis," in Proc. Int. Workshop Human Interaction Proofs, 2005, pp. 42–52.

[20] P. Yang, G. Zhao, and P. Zeng, "Phishing website detection based on multidimensional features driven by deep learning," IEEE Access, vol. 7, pp. 15196–15209, 2019.

[21] A. K. Jain and B. B. Gupta, "A machine learning based approach for phishing detection using hyperlinks information," J. Ambient Intell. Humanized Comput., vol. 10, no. 5, pp. 2015–2028, 2019.

[22] R. S. Rao and A. R. Pais, "Detection of phishing websites using an efficient feature-based machine learning framework," Neural Comput. Appl., vol. 31, no. 8, pp. 3851–3873, 2019.

[23] Gandotra and D. Gupta, "An efficient approach for phishing detection using machine learning," in Proc. ICCCA, May 2017, pp. 1106–1111.

[24] L. Tang and Q. H. Mahmoud, "A survey of machine learning-based solutions for phishing website detection," Mach. Learn. Knowl. Extr., vol. 3, no. 3, pp. 672–694, 2021.

[25] S. Marchal, K. Saari, N. Singh, and N. Asokan, "Know your phish: Novel techniques for detecting phishing sites and their targets," in Proc. IEEE ICDCS, Jun. 2016, pp. 323–333.

[26] P. K. Banerjee, A. K. Singh, and D. Samanta, "Phishing detection: A comprehensive survey," in Proc. ICCCI, Jan. 2021, pp. 1–6.

[27] Z. Dou, I. Khalil, A. Khreishah, A. Al-Fuqaha, and M. Guizani, "Systematization of knowledge (SoK): A systematic review of software-based web phishing detection," IEEE Commun. Surveys Tuts., vol. 19, no. 4, pp. 2797–2833, 2017.

[28] M. Khonji, Y. Iraqi, and A. Jones, "Phishing detection: A literature survey," IEEE Commun. Surveys Tuts., vol. 15, no. 4, pp. 2091–2121, 2013.

[29] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, "Beyond blacklists: Learning to detect malicious web sites from suspicious URLs," in Proc. ACM SIGKDD, Jun. 2009, pp. 1245–1254.

[30] A. Blum, B. Wardman, T. Solorio, and G. Warner, "Lexical feature based phishing URL detection using online learning," in Proc. ACM AISec, Oct. 2010, pp. 54–60.

[31] S. Abu-Nimeh, D. Nappa, X. Wang, and S. Nair, "A comparison of machine learning techniques for phishing detection," in Proc. eCrime Researchers Summit, Oct. 2007, pp. 60–69.

[32] L. Tan, K. L. Chiew, K. S. C. Yong, and W. K. Tiong, "PhishWHO: Phishing webpage detection via identity keywords extraction and target domain finder," Decision Support Systems, vol. 88, pp. 18–27, 2016.

[33] R. Verma and K. Dyer, "On the character of phishing URLs: Accurate and robust statistical learning classifiers," in Proc. ACM CODASPY, Mar. 2015, pp. 111–122.

[34] A. Fette, N. Sadeh, X. Wang, and S. Nair, "Learning to detect phishing emails," in Proc.

WWW, May 2007, pp. 649–656.
[35] Xiang et al., "CANTINA+: A feature-rich machine learning framework for detecting phishing web sites," ACM Trans. Inf. Syst. Secur., vol. 14, no. 2, pp. 1–28, 2011.
[36] M. He et al., "An efficient phishing webpage detector based on feature selection and random forest," in Proc. ICMLC, Jul. 2011, pp. 1635–1640.
[37] A. K. Singh and N. Roy, "Phishing website detection using machine learning algorithms," in Proc. ICACCS, Jan. 2020, pp. 1176–1181.
[38] A. Le, A. Markopoulou, and M. Faloutsos, "PhishDef: URL names say it all," in Proc. IEEE INFOCOM, Apr. 2011, pp. 191–195.
[39] S. Sheng et al., "An empirical analysis of phishing blacklists," in Proc. CEAS, Jul. 2009, pp. 1–10.
[40] Sahoo, C. Liu, and S. C. H. Hoi, "Malicious URL detection using machine learning: A survey," arXiv, 2017.
[41] M. E. Ahmed, H. Kim, and M. Park, "Phishing detection using machine learning and deep learning: A review and future directions," IEEE Access, vol. 8, pp. 198915–198939, 2020.
[42] Shirazi, K. Bezawada, and I. Ray, "Kn0w thy d0main name: Unbiased phishing detection using domain name-based features," in Proc. ACM SACMAT, Jun. 2018, pp. 69–75.