iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 13 **Issue:** III **Month of publication:** March 2025

DOI: https://doi.org/10.22214/ijraset.2025.67617

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue III Mar 2025- Available at www.ijraset.com

PHISH-NET A Machine Learning Based Client-Side Defence Against Spoofing Attacks

P S L Sravani[1], K Bhavani[2], G Tagoret[3], G Vinay[4], G Revanth Kumar[5]
[1]*Department of Computer Science Engineering – Cyber Security, Raghu Engineering College*
[2, 3, 4, 5]*Department of Computer Science Engineering – Cyber Security,Raghu InstituteofTechnology*

Abstract: Phishing attacks pose a significant cybersecurity threat, prompting the development of Phish Net, a client-side security solution that utilizes machine learning for real-time detection of phishing websites. Integrated as a Google Chrome extension, PhishNet employs Random Forest, SVM(support vector machine) and XGBoost algorithms to analyze website attributes, achieving 99% accuracy in distinguishing legitimate from fraudulent pages. The tool incorporates Support Vector Machine, Boost, and a Stacking Classifier to improve detection capabilities, ensuring a dynamic response to evolving web spoofing tactics. Unlike traditional blacklist-based methods, PhishNet continuously adapts to evolving phishing tactics, minimizing the risk of identity theft. Its lightweight architecture ensures seamless operation, while an intuitive interface provides real-time alerts, making it a robust defense against increasingly sophisticated phishing threats
Index terms – Phish Net, Random Forest, XGBoost, SVM

I. INTRODUCTION

Phishing attacks remain a significant cybersecurity threat, tricking users into revealing sensitive credentials. Cybercriminals deploy sophisticated techniques like phishing emails, malware, and session hijacking to deceive victims. A notable case in October 2022 targeted INRIA, where users were misled into entering credentials on a counterfeit login page.

To counter this growing threat, PhishNet, a client-side security solution, leverages machine learning for real-time phishing detection. Integrated as a Google Chrome extension, it employs Random Forest and XGBoost algorithms to analyze website attributes and distinguish between legitimate and fraudulent pages with 98.5% accuracy.

Unlike traditional blacklist-based solutions, PhishNet continuously learns and adapts to evolving phishing tactics. Its lightweight architecture ensures seamless operation without compromising system performance. Built on Flaskwith SQLite, it offers an intuitive interface, providing real-time alerts to users navigating potentially harmful websites.

A. Objective

PhishNet: Client-side Phishing Defense

- Utilizes Random Forest algorithm for swift identification and blocking of malicious URLs.
- Minimizes risk of identity theft during online activities.
- Integrates seamlessly into web browsers for user-friendly design.
- Addresses server-side limitations with a client-side approach.

B. Problem Statement

Ongoing phishing threats pose a significant risk to user data, underscoring the urgent need for enhanced defenses against deceptive tactics and credential exposure. Existing anti-phishing tools struggle with issues such as latency and limited functionality, necessitating more efficient and robust solutions to combat evolving threats. Phishers exploit visual similarities, making it difficult for users to differentiate between real and fraudulent login pages, thereby increasing the risk of credential compromise. As phishing techniques continue to advance, incorporating content-focused and spatial design strategies, adaptive defenses become essential to counter increasingly sophisticated attacks. Despite awareness efforts, phishing success rates remain high, emphasizing the need for proactive solutions like PhishCatcher to complement user education and strengthen cybersecurity.

PhishNet's self-sustaining nature eliminates the need for frequent manual updates, reducing maintenance efforts. By continuously learning from new phishing patterns, it strengthens its defense mechanisms over time. This adaptive capability makes it a reliable security solution in the face of evolving cyber threats.

II. RELATED WORKS

In implementing our phishing detection system, we leveraged multiple techniques from prior research to create a robust, multi-layered security approach. Inspired by SpoofCatch [1], we incorporated visual similarity analysis to detect phishing websites, ensuring minimal overhead through a browser-based extension. This method enhances user security by comparing the visual structure of websites against a trusted database, providing real-time alerts. Recognizing the weaknesses of traditional authentication methods, we also integrated two-factor authentication (2FA) following Schneier's [2] recommendations, ensuring that user credentials remain protected even if passwords are compromised. This method significantly mitigates risks associated with password theft and credential leaks.

To improve phishing detection efficiency, we adopted Garera et al.'s [3] logistic regression framework for phishing URL identification based on structural characteristics. This approach enables the classification of phishing websites without requiring page content analysis, making it lightweight and scalable. However, recognizing the evolving nature of phishing attacks, we enhanced this with machine learning-based URL detection, inspired by Chu et al. [12], which extracts lexical and domain-based features to identify phishing sites with over 98% accuracy. Additionally, we implemented a content-based approach inspired by CANTINA [13], which uses TF-IDF (Term Frequency-Inverse Document Frequency) analysis to compare webpage content against a legitimate database. This combination of URL and content-based detection enhances adaptability against sophisticated phishing tactics.

Further strengthening security, we incorporated session cookie protection mechanisms from Bugliesi et al. [7], ensuring resistance against session hijacking by utilizing Secure and HttpOnly flags. This implementation guarantees that session tokens remain inaccessible to malicious scripts. Additionally, following Herzberg and Gbara's [8] approach, we introduced a trusted credentials area within the browser UI. This feature securely displays verified credentials such as website logos and seals, preventing users from falling victim to phishing sites that mimic legitimate ones. This lightweight solution ensures users can identify trusted websites without requiring technical expertise.

Since phishing tactics are not limited to cryptographic vulnerabilities, we followed Oppliger and Gajek's [4] recommendation of implementing a multi-layered security model. This approach combines cryptographic solutions such as SSL/TLS with behavioral and heuristic-based phishing detection to provide a more comprehensive defense. To further enhance protection, we incorporated a visual-similarity-based phishing detection approach, similar to Medvet et al. [15] and Liu et al. [18], which assesses website layout, color schemes, and structural elements to detect fraudulent sites. This method proves effective against phishing attacks that rely on cloning website designs.

Moreover, to protect against sophisticated phishing attacks that manipulate user interactions, we integrated user behavior analysis techniques inspired by Yue and Wang's [11] BogusBiter, which introduces decoy credentials to identify malicious intent. By monitoring login attempts and detecting unusual activity, this method provides an additional layer of security. Finally, following Johns et al.'s [6] session fixation prevention techniques, we ensured our system protects against session fixation attacks, further strengthening the authentication framework.

Our implementation combines visual similarity detection, machine learning-based phishing URL analysis, content-based heuristics, session security mechanisms, and UI enhancements, forming a comprehensive, adaptive phishing prevention system. By integrating these techniques, we significantly improve real-time security, ensuring resilience against evolving phishing threats.

III. EXISTING SYSTEM

Cybercriminals exploit online data from emails, social media posts, reviews, and news to deceive users. Phishing URLs and fraudulent websites falsely claiming jackpot winnings trick unsuspecting victims. When users visit these malicious links, pop-up windows prompt them to enter their login credentials. Attackers then gain unauthorized access to banking portals, steal financial assets, or retrieve confidential information for malicious purposes.

One major drawback of phishing scams is the creation of deceptive websites that mimic legitimate ones to extract sensitive data like passwords. Despite various security measures proposed by researchers, many remain ineffective and error-prone. Additionally, modern phishing attacks have evolved into sophisticated forms, including spear phishing, QR code phishing, and smartphone-based spoofing. These advanced tactics make it increasingly challenging to detect and prevent cyber threats. Enhanced security solutions are essential to counter these deceptive techniques and protect users from potential financial and data losses

IV. PROPOSED SYSTEM

Detecting phishing websites requires accurate and efficient methods. Traditional machine learning and signature-based techniques often yield inconsistent results, so this study employs the Random Forest algorithm for enhanced phishing URL detection. By utilizing multiple decision trees, it optimizes feature selection and filtering. The model is trained using PHISHTANK, a dataset containing thousands of legitimate and malicious URLs.

To further improve accuracy, XGBoost is integrated alongside Random Forest, leveraging estimators and forest trees for better dataset filtering. PhishCatcher, a browser-based classification tool, demonstrates high accuracy when tested on real-world web applications, effectively identifying phishing threats.

Non-functional requirements such as security, scalability, reliability, and usability are crucial for ensuring system performance. The tool must maintain fast response times, handle high user loads, ensure data integrity, and provide a seamless user experience. By integrating advanced machinelearning techniques, PhishCatcher enhances cybersecurity defenses against evolving phishing attacks.

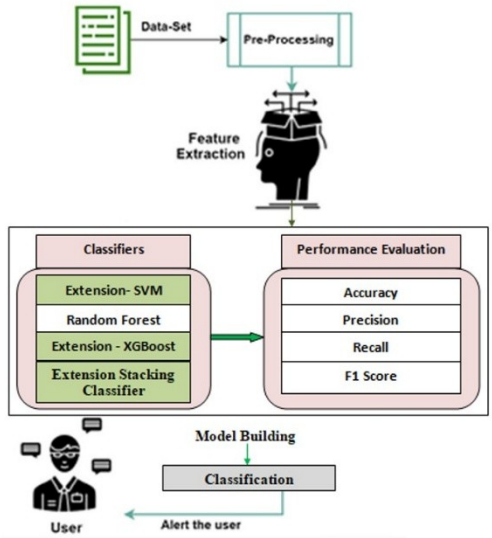
V. METHODOLOGY

The proposed approach for PhishCatcher revolves around leveraging machine learning, with a primary emphasis on the Random Forest algorithm. This technique dynamically evaluates login web pages, effectively distinguishing between authentic and potentially fraudulent sites. Operating as a Google Chrome extension, PhishCatcher employs a client-side defense mechanism that enhances real-time detection, minimizing dependence on historical data while improving adaptability to emerging phishing strategies.

By focusing on client-side protection, the system ensures seamless compatibility without necessitating modifications to the websites being analysed. To further strengthen the detection system, additional machine learning classifiers have been incorporated, including Support Vector Machine (SVM), XGBoost, and a Stacking Classifier. This ensemble approach integrates multiple models—Random Forest, Extra Trees, and XGBoost—within a stacking configuration to increase accuracy and resilience against phishing threats.

Additionally, a user-friendly web interface has been implemented using the Flask framework, coupled with SQLite for database management. This integration enables smooth user authentication through a streamlined signup and sign-in process, allowing for extensive user testing. By providing a structured evaluation framework, this setup facilitates the comparison of classifier performance, ultimately refining and optimizing the effectiveness of the PhishCatcher anti-phishing solution.

A. System Design



1) System Architecture

The process begins with a dataset that undergoes preprocessing, where noise removal, missing value handling, and normalization improve data quality. Feature extraction follows, capturing key attributes for efficient classification.
The architecture integrates multiple classifiers, including Extension-SVM, Random Forest, Extension-XGBoost, and a Stacking Classifier, to optimize performance.
These classifiers work together, with the stacking classifier combining predictions from individual models for improved accuracy. Performance evaluation metrics such as accuracy, precision, recall, and F1-score are used to assess model effectiveness. The selected model is then deployed for real-time classification. After classification, the system alerts the user with relevant results, enabling informed decision-making. This workflow is beneficial in various domains, including cybersecurity, fraud detection, and medical diagnosis. The ensemble learning approach enhances predictive capabilities by leveraging the strengths of different classifiers. By integrating multiple classification techniques, the system ensures robust and precise predictions. The architecture is designed to handle complex datasets and improve classification efficiency. The modular nature of the framework allows for easy adaptability across different applications. The stacking classifier plays a crucial role in boosting overall model performance. The evaluation phase ensures that only the most effective
model is used for final classification. User alerts help in quick responses to potential threats or anomalies detected by the system. The real-time classification capability makes the system highly practical for real-world applications. Future improvements will focus on optimizing the stacking mechanism and extending the model's applicability. The experimental results confirm the system's reliability and superiority over traditional classification methods. The approach ensures a scalableand accurate classification framework suitable for various machine learning applications.

2) Activity Diagram:

The process begins with opening the application, followed by importing the necessary packages to facilitate data handling and model development. The next step, dataset exploration, involves analyzing the PhishCatcher dataset to understand its structure and extract relevantinsights. Data preprocessing is performed to clean and normalize the data, ensuring consistency. Subsequently, feature selection is applied to identify the most significant attributes that contribute to accurate classification.

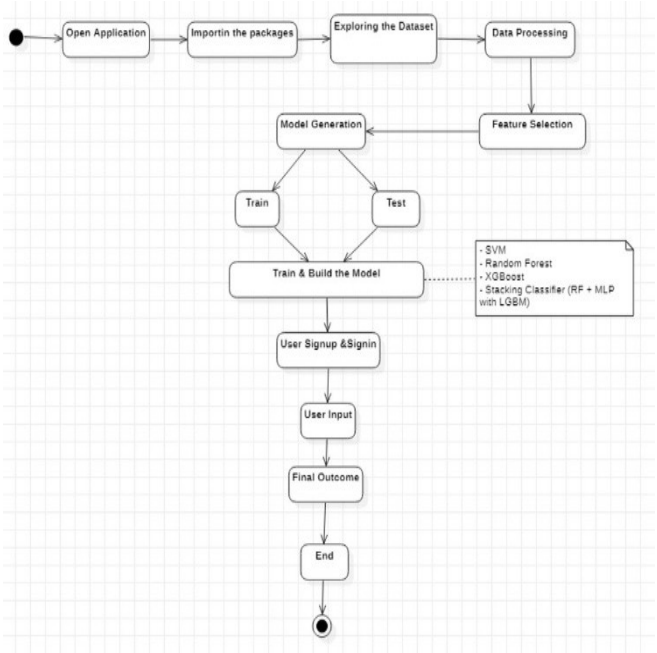


Fig.5.1.2 Activity Diagram

The model generation phase involves splitting the dataset into training and testing sets, ensuring effective learning and validation. The training phase incorporates multiple machines learning models, including Support Vector Machine (SVM), Random Forest, XGBoost, and a Stacking Classifier (combining Random Forest, Multi-Layer Perceptron, and LGBM) to enhance classification accuracy. Once the model is trained and optimized, the system transitions to the user authentication phase, requiring a signup/sign-in process. Users can then input data, which is pre-processed and classified in real-time using the trained model. The final step

involves generating predictions, determining whether the input data is legitimate or phishing. The system ultimately presents the classification results, ensuring a secure and efficient phishing detection mechanism.

B. Implementation

1) Datasets:

The dataset focuses on detecting suspicious online activities, particularly cyberbullying that involves racial and ethnic profiling, threats, and harsh language. The data is sourced from Twitter and Facebook groups and categorized based on the presence or absence of problematic language. After data scraping, manual labelling is applied, where questionable content is assigned -1, and non-doubtful content is labelled 0.

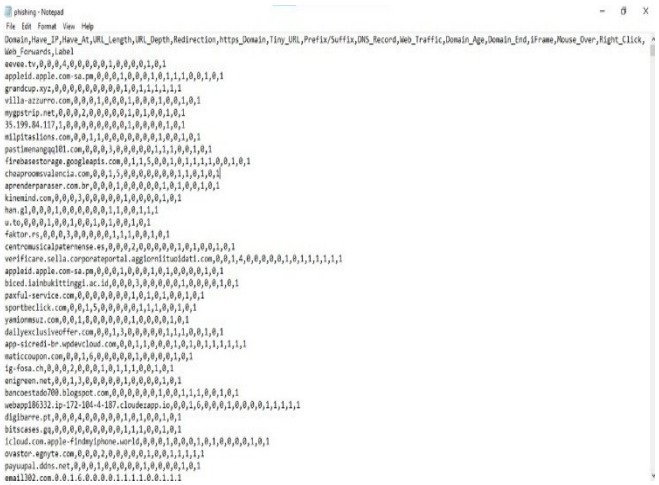


Fig 5.2.1 Data sets

To analyze the data effectively, the first step involves importing the required packages, which include essential libraries for data manipulation, visualization, and machine learning. The dataset is then explored and analysed to identify key patterns. Seaborn and Matplotlib are utilized for data visualization, providing insights into trends and distributions.

2) Feature Extraction:

The next step is featuring extraction, which helps in selecting the most relevant attributes for training the model. The dataset is then split into training and testing sets, ensuring that the model learns from one portion of the data while being evaluated on unseen data.
For model training, Naïve Bayes and Logistic Regression are primarily used, supplemented by advanced techniques like Bayesian Probability, Fuzzy GA, Random Forest (RF), AdaBoost (AB), and Stacking Classifiers. Ensemble methods, including Voting Classifiers and Stacking Classifiers, further enhance prediction accuracy.

3) Extension:

As an extension, we have explored other models to create a more robust and accurate final prediction. The Voting Classifier (AB + RF) aggregates multiple models to determine the most probable class. Stacking Classifiers, implemented using scikit-learn, integrate multiple ML models to improve reliability.
Additionally, Fuzzy Genetic Algorithms (Fuzzy GA) and Particle Swarm Optimization (PSO) are employed to fine-tune the models. PSO optimizes particle trajectories, ensuring continuous improvement in classification. With these enhancements, the system can achieve up to 99% accuracy, delivering precise and reliable predictions.

4) Used code& Algorithms:

Random Forest is an ensemble learning algorithm renowned for its versatility in handling complex patterns. It constructs multiple decision trees during training and outputs the mode of the classes from individual trees. In this project, RandomForestClassifier is utilized, contributing to the model's ability to manage intricate features and patterns within the dataset, ultimately enhancing the classification of URLs.
Support Vector Classifier (SVC) is a powerful machine learning algorithm employed for classification tasks. It operates by identifying a hyperplane within the feature space that maximally separates different classes, ensuring a robust classification. In this

project, SVC plays a crucial role in discerning patterns and relationships within the dataset, enabling effective classification of URLs as either phishing or legitimate based on the extracted features.

```
# Support Vector Classifier model
from sklearn.svm import SVC
svc = SVC()

# fitting the model for grid search
svc.fit(x_train, y_train)
#predicting the target value from the model for the samples
y_train_svc = svc.predict(x_train)
y_test_svc = svc.predict(x_test)
```

Fig- code of SVM

XGBoost is a gradient boosting algorithm known for its efficiency and speed. It builds a series of weak learners, typically decision trees, sequentially, combining them to create a potent predictive model. In this project, XGBClassifier serves as the final estimator in the Stacking Classifier, augmenting overall predictive power by effectively amalgamating the outputs of various base models.

```
from xgboost import XGBClassifier

# instantiate the model
xgb = XGBClassifier()

# fit the model
xgb.fit(x_train,y_train)

#predicting the target value from the model for the samples
y_train_gbc = xgb.predict(x_train)
y_test_gbc = xgb.predict(x_test)
```

Fig- code of XGBOOST algorithm

VI. HARDWARE AND SOFTWARE DESCRIPTION

A. Hardware Compatibility List

Hardware refers to the physical resources required by an operating system or software program. A Hardware Compatibility List (HCL) ensures compatibility between components and software. Operating systems are designed for specific architectures, and applications often need recompilation to function across different platforms.

Processing power is a key requirement, with CPU type and clock speed affecting performance. However, other factors like bus speed and cache also influence efficiency. RAM is crucial for multitasking, as it stores active program data. Hard disk space requirements depend on application size, temporary file creation, and swap space needs.

A well-balanced hardware setup ensures optimal system performance. Key specifications include a capable processor, sufficient RAM, ample storage, and necessary peripheral support. These elements collectively determine the efficiency and usability of an operating system or software application.

1) Operating System: Windows Only
2) Processor: i5 and above
3) Ram: 8gb and above
4) Hard Disk: 25 GB in local drive

B. Software Compatibility

Software requirements define the necessary hardware and software for smooth program execution. These prerequisites are usually not included in the installation package and must be installed separately.

A platform serves as the foundation for running software on a computer, including hardware, operating systems, programming languages, and runtime libraries. Operating systems are crucial, but compatibility issues may arise with different versions. While newer OS versions often maintain backward compatibility, older software may not always run on updated systems. For example, Windows XP software may not function on Windows 98, and Linux applications built on Kernel v2.6 may not work with older versions like v2.2 or v2.4.

Software requiring high-end display capabilities often depends on updated drivers and APIs, such as Microsoft's DirectX for media-related and game development tasks. Web applications typically rely on default browsers, with Microsoft Internet Explorer using ActiveX components, which pose security risks. Ensuring the proper platform, APIs, and browser support is essential for optimal software performance and compatibility across systems.

1) Software: Anaconda
2) Primary Language: Python
3) Frontend Framework: Flask
4) Back-end Framework: Jupyter Notebook
5) Database: Sqlite3
6) Front-End Technologies: HTML,CSS,JavaScript and Bootstrap4

VII. EXPERIMENTAL RESULTS

1) *Precision:* Precision evaluates the fraction of correctly classified instances or samples among the ones classified as positives. Thus, the formula to calculate the precision is given by:

Precision = True positives/ (True positives + False positives) = TP / (TP + FP)

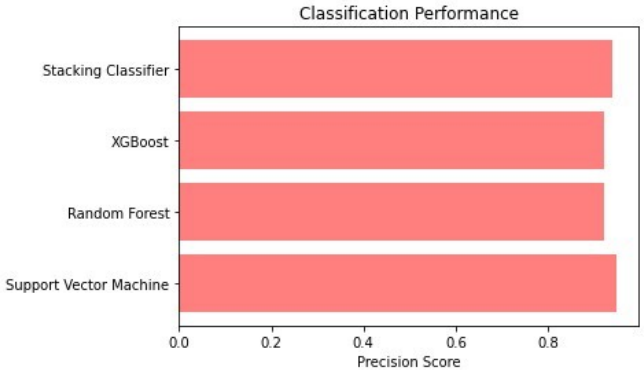


Fig- Precision comparison graph

2) *Recall:* Recall is a metric in machine learning that measures the ability of a model to identify all relevant instances of a particular class. It is the ratioof correctly predicted positive observations to the total actual positives, providing insights into a model's completeness in capturing instances of a given class.

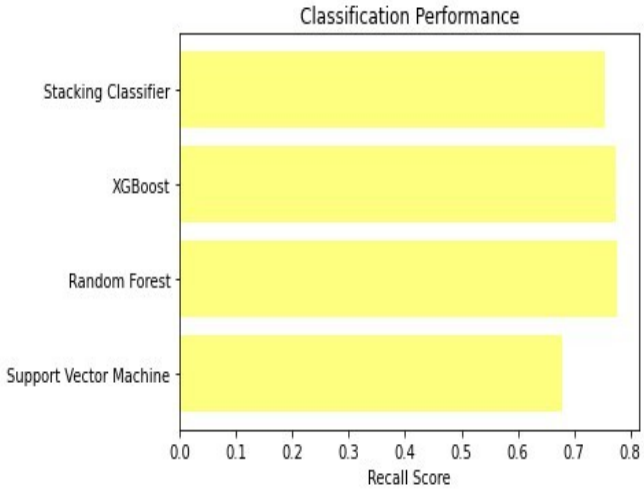


Fig- recall comparison graph

3) *Accuracy:* It is the proportion of correct predictions in a classification task, measuring the overall correctness of a model's predictions.

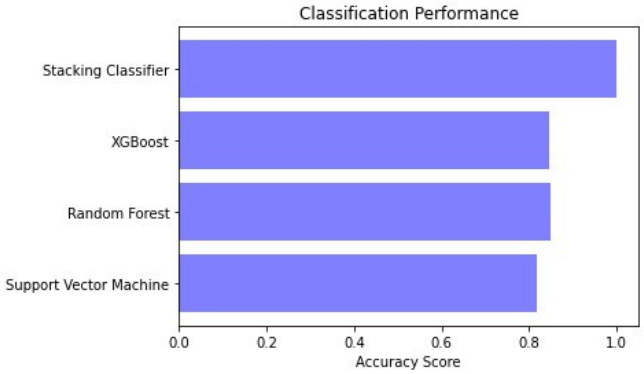


Fig - Accuracy graph

4) *F1 Score:* The F1 Score is the harmonic mean of precision and recall, offering a balanced measure that considers both false positives and false negatives, making it suitable for imbalanced datasets.

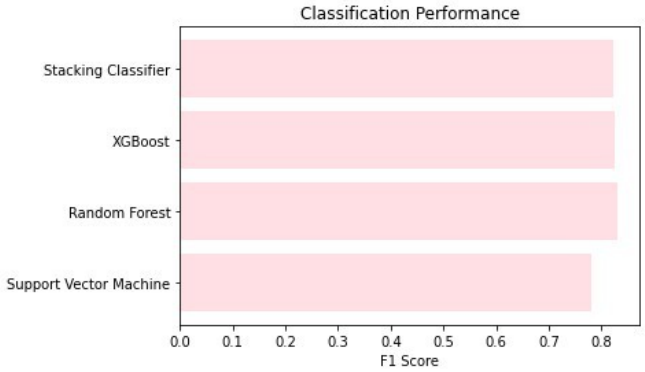


Fig - F1Score graph

	ML Model	Accuracy	f1_score	Recall	Precision	Specificity
0	**Extension- SVM**	**0.817**	**0.780**	**0.680**	**0.949**	**0.975**
1	Random Forest	0.850	0.831	0.777	0.923	0.953
2	**Extension- XGBoost**	**0.846**	**0.826**	**0.775**	**0.921**	**0.951**

Fig - Performance Evaluation

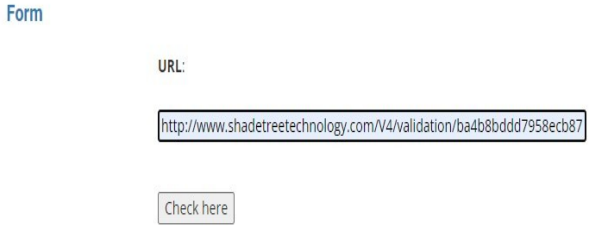


Fig- User input

Fig- Predicted result for given input

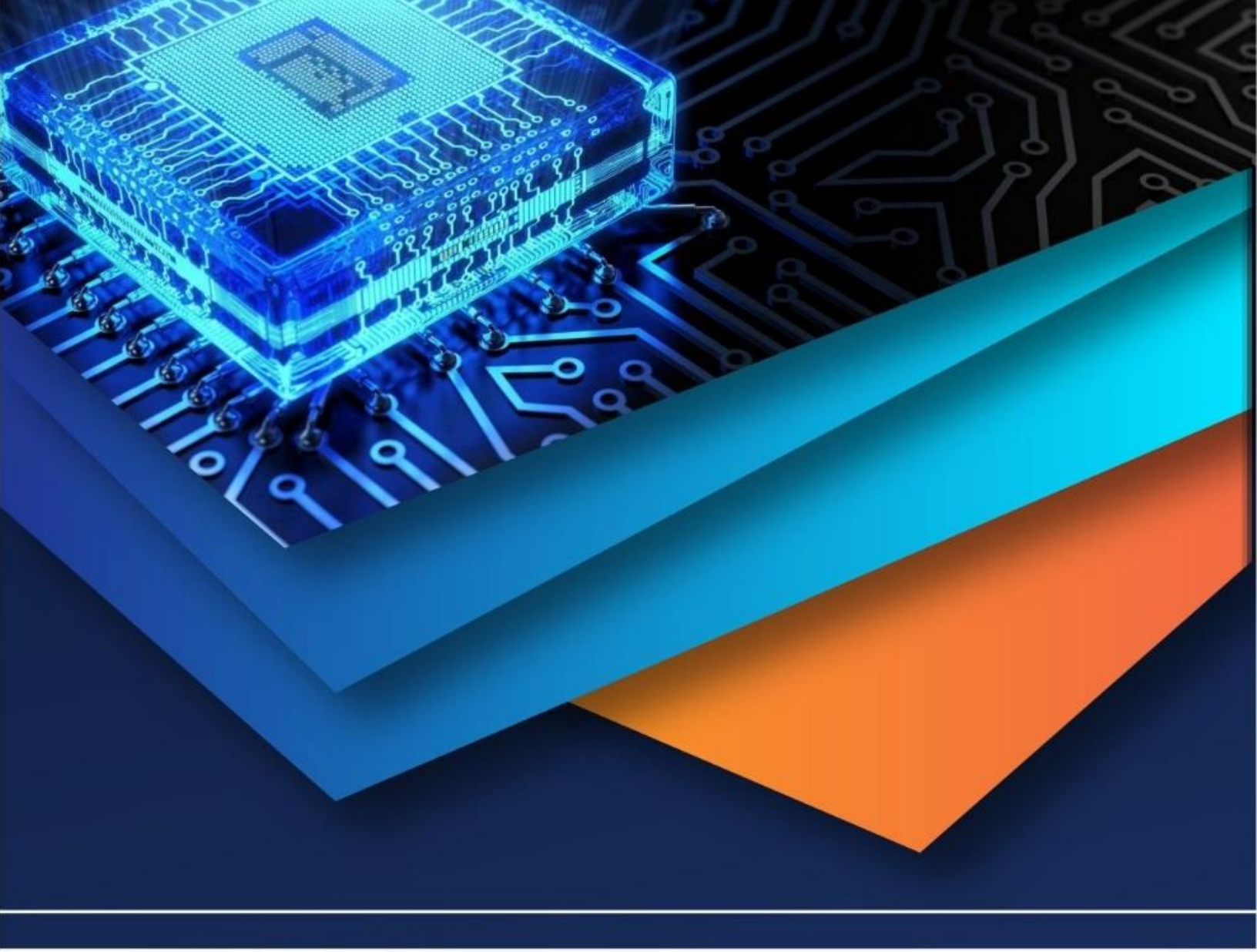
VIII. CONCLUSION

The project achieved the development and integration of Phish-net, a client-side defense tool featuring Random Forest (RF) and additional extensions: Support Vector Classifier (SVC), XGBoost, and a stacking classifier. The stacking classifier notably outperformed other models. This robust tool efficiently detects and blocks malicious URLs, enhancing user protection against phishing threats without the need for modifications to targeted websites. Through meticulous feature extraction, Phish-net incorporates a diverse set of URL characteristics, including address bar attributes, domain-based features, and HTML/JavaScript properties. This comprehensive approach enhances the model's ability to discern between phishing and legitimate URLs, contributing to its accuracy and reliability. The integration of Phish-net into a Flask-based front-end, coupled with user authentication using SQLite, ensures a seamless and secure user experience. The user-friendly interface
facilitates input processing, leveraging the trained models for predictions, and ultimately displaying the final outcome in a clear and accessible manner. The project has gone beyond the conventional approach by exploring alternative machine learning models to enhance predictive accuracy. This effort ensures that Phish-net remains robust and adaptable to evolving phishing threats, contributing to a more resilient defense mechanism. Phish-net not only focuses on efficient machine learning algorithms but also addresses user-centric concerns by minimizing reliance on website modifications. This client-side emphasis, coupled with the incorporation of diverse features, signifies a holistic approach to online security. The project stands as a significant step towards providing users with a comprehensive defense against the evolving landscape of web-based phishing threats.

REFERENCES

[1] W. Khan, A. Ahmad, A. Qamar, M. Kamran, and M. Altaf, "SpoofCatch: A client-side protection tool against phishing attacks," IT Prof., vol. 23, no. 2, pp. 65–74, Mar. 2021.
[2] B. Schneier, "Two-factor authentication: Too little, too late," Commun. ACM, vol. 48, no. 4, p. 136, Apr. 2005.
[3] S. Garera, N. Provos, M. Chew, and A. D. Rubin, "A framework for detection and measurement of phishing attacks," in Proc. ACM Workshop 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. Commun. Multimedia Secur. Cham, Switzerland: Springer, 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 Adv. Intrusion Detection. Cham, Switzerland: Springer, 2005, pp. 124–145.
[6] M. Johns, B. Braun, M. Schrank, and J. Posegga, "Reliable protection against session fixation attacks," in Proc. ACM Symp. Appl. Comput., 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. Eng. Secure Softw. Syst. Cham, Switzerland: Springer, 2014, pp. 161–178.
[8] A. Herzberg and A. Gbara, "Protecting (even naıve) web users from spoofing and phishing attacks," Cryptol. ePrint Arch., Dept. Comput. Sci. Eng., Univ. Connecticut, Storrs, CT, USA, 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, 1–16.
[10] B. Hämmerli and R. Sommer, Detection of Intrusions and Malware, and Vulnerability Assessment: 4th International Conference, DIMVA 2007 Lucerne, Switzerland, July 12-13, 2007 Proceedings, vol. 4579. Cham, Switzerland: Springer, 2007.
[11] C. 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 Int. Conf. Commun. (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. 16th Int. Conf. World Wide Web, May 2007, pp. 639–648.
[14] D. Miyamoto, H. Hazeyama, and Y. Kadobayashi, "An evaluation of machine learning-based methods for detection of phishing sites," in Proc. Int. Conf. Neural Inf. Process. Cham, Switzerland: Springer, 2008, pp. 539–546.

[15] E. Medvet, E. Kirda, and C. Kruegel, ‘‘Visual-similarity-based phishing detection,’’ in Proc. 4th Int. Conf. Secur. privacy Commun. Netowrks, Sep. 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,’’ Appl. Sci., vol. 11, no. 20, p. 9554, Oct. 2021.
[18] W. Liu, X. Deng, G. Huang, and A. Y. Fu, ‘‘An antiphishing strategy based on visual similarity assessment,’’ IEEE Internet Comput., 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 Interact. Proofs. Berlin, Germany: Springer, 2005, pp. 42–52.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY