PHISH DETECT: BUILDING AND DEPLOYING A PHISHING URL DETECTION SYSTEM WITH MACHINE LEARNING

A PROJECT REPORT

***Submitted* by**

ARUNKUMAR K. V. (211420104027)

EMMANUEL JOSHUA C. (211420104074)

DILIP G. (211420104067)

in partial fulfilment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

PANIMALAR ENGINEERING COLLEGE

(An Autonomous Institution, Affiliated to Anna University, Chennai)

APRIL 2024

PANIMALAR ENGINEERING COLLEGE

(An Autonomous Institution, Affiliated to Anna University, Chennai)

BONAFIDE CERTIFICATE

Certified that this project report **"Phish Detect: Building and Deploying a Phishing URL Detection System with Machine Learning"** is the bonafide work of **ARUN KUMAR K.V. (211420104027), EMMANUEL JOSHUA C. (211420104074), DILIP G. (211420104067)** who carried out the project work under my supervision.

SIGNATURE	**SIGNATURE**
Dr. L. JABASHEELA M.E., Ph.D., **PROFESSOR AND HEAD,**	**Mrs. N. SIVAKAMY M.E., Ph.D,** **ASSISTANT PROFESSOR,**
DEPARTMENT OF CSE, PANIMALAR ENGINEERING COLLEGE, NASARATHPETTAI, POONAMALLEE, CHENNAI-600 123.	DEPARTMENT OF CSE, PANIMALAR ENGINEERING COLLEGE, NASARATHPETTAI, POONAMALLEE, CHENNAI-600 123.

Certified that the above candidate(s) were examined in the Mini Project Viva-Voce Examination held on...........................

INTERNAL EXAMINER EXTERNAL EXAMINER

DECLARATION BY THE STUDENT

We **ARUN KUMAR K.V. (211420104027), EMMANUEL JOSHUA C. (211420104074), DILIP G. (211420104067)** hereby declare that this project report titled "**PHISH DETECT: BUILDING AND DEPLOYING A PHISHING URL DETECTION SYSTEM WITH MACHINE LEARNING**", under the guidance of **Mrs. SIVAKAMY N.** is the original work done by us and we have not plagiarized or submitted to any other degree in any university by us.

NAME OF THE STUDENTS

ARUNKUMAR K.V. (211420104027)

EMMANUEL JOSHUA C. (211420104074)

DILIP G. (211420104067)

ACKNOWLEDGEMENT

Our profound gratitude is directed towards our esteemed Secretary and Correspondent, **Dr. P. CHINNADURAI, M.A., Ph.D**., for his benevolent words and fervent encouragement. His inspirational support proved instrumental in galvanizing our efforts, ultimately contributing significantly to the successful completion of this project

We want to express our deep gratitude to our Directors, **Tmt. C. VIJAYARAJESWARI, Dr. C. SAKTHI KUMAR, M.E., Ph.D., and Dr. SARANYASREE SAKTHI KUMAR, B.E., M.B.A., Ph.D.,** for graciously affording us the essential resources and facilities for undertaking of this project.

Our gratitude is also extended to our Principal, **Dr. K. MANI, M.E., Ph.D.,** whose facilitation proved pivotal in the successful completion of this project.

We express my heartfelt thanks to **Dr. L. JABASHEELA, M.E., Ph.D.,** Head of the Department of Computer Science and Engineering, for granting the necessary facilities that contributed to the timely and successful completion of project.

We would like to express our sincere thanks to **Project Coordinator Dr. N. PUGHAZENDI M.E., Ph.D.** and **Project Guide Mrs. N. SIVAKAMY M.E., Ph.D.** and all the faculty members of the Department of CSE for their unwavering support for the successful completion of the project.

NAME OF THE STUDENTS

ARUNKUMAR K.V. (211420104027)

EMMANUEL JOSHUA C. (211420104074)

DILIP G. (211420104067)

ABSTRACT

Phishing attacks have emerged as one of the most prevalent and insidious cyber threats in recent years, exploiting the trust of unsuspecting users to steal sensitive information, such as login credentials, financial data, and personal details. These attacks typically involve deceptive tactics, where attackers masquerade as legitimate entities, such as banks, social media platforms, or government agencies, to trick individuals into divulging confidential information or performing actions that compromise their security. The methods used in phishing attacks continue to evolve and become increasingly sophisticated, making them harder to detect and mitigate. Attackers employ various techniques, including email spoofing, fake websites, social engineering, and malware distribution, to deceive users and gain unauthorized access to their accounts or sensitive data. Phishing attacks have a profound and far-reaching impact on online security, affecting individuals, businesses, and society as a whole in various ways. Firstly, they can result in significant financial losses for both individuals and organizations, with attackers stealing funds directly from bank accounts, conducting fraudulent transactions, or engaging in identity theft. Secondly, successful phishing attacks often lead to data breaches, compromising sensitive information such as usernames, passwords, and credit card numbers, which not only jeopardizes individuals' privacy but also damages the reputation and credibility of businesses. Thirdly, phishing enables identity theft, allowing attackers to impersonate victims and misuse their personal information for fraudulent purposes, such as opening accounts or filing tax returns. Additionally, these attacks can disrupt online services and operations by compromising user accounts, spreading malware, or launching distributed denial-of-service (DDoS) attacks against websites and networks, further undermining the reliability and availability of digital platforms. Moreover, phishing erodes trust in online communication and transactions, diminishing users' confidence in the security of digital services, which can have long-term consequences for businesses and the broader digital ecosystem. Given the escalating sophistication and prevalence of phishing attacks, there's an urgent need for advanced solutions like Phish Detect to effectively counter this threat. Traditional security measures often fall short in detecting and preventing these evolving attacks. Phish Detect offers real-time analysis, seamless browser integration, adaptive detection mechanisms, thorough scrutiny of sub links, and a user-friendly interface. Advanced solutions like Phish Detect are essential in safeguarding online security, requiring continuous innovation and proactive measures to protect individuals, businesses, and society from the damaging impacts of phishing attacks.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

CHAPTER 1
INTRODUCTION

1.1 OVERVIEW

The threat landscape of phishing attacks is persistent and continually evolving, posing significant risks to individuals and businesses alike. Exploiting both human psychology and technological vulnerabilities, phishing attacks remain a top concern in cybersecurity. Traditional security measures often struggle to keep pace with the dynamic tactics employed by cybercriminals, necessitating innovative and proactive approaches to effectively counter these evolving threats. In response to this challenge, the "Phish Detect" browser extension emerges as a pioneering solution, designed to empower users with real-time protection against phishing attempts directly within their web browsing environment. Phish Detect represents a significant advancement in the realm of cybersecurity, offering a comprehensive suite of features aimed at mitigating the risks posed by phishing attacks. One of its primary strengths lies in its seamless integration into popular web browsers, ensuring widespread accessibility and ease of use for a diverse user base. By embedding real-time analysis and adaptive detection mechanisms directly into the browsing experience, Phish Detect provides users with timely alerts and actionable insights to help them discern the legitimacy of URLs encountered online. The extension's proactive approach to phishing detection enables users to make informed decisions about potentially malicious links, significantly reducing the likelihood of falling victim to phishing scams. Through continuous monitoring of web traffic and page content, Phish Detect swiftly identifies suspicious activity indicative of phishing attempts, delivering timely warnings to users and empowering them to take appropriate action to protect themselves. Moreover, Phish Detect's adaptive detection mechanisms represent a significant advancement in phishing defence strategies. Leveraging machine learning algorithms and data analytics, the extension continuously evolves and learns from emerging phishing patterns over time. By analysing large volumes of data and identifying common characteristics of phishing attempts, Phish Detect can adapt its detection capabilities to effectively counter new and sophisticated phishing techniques. The extension's commitment to empirical analysis and user feedback further enhances its efficacy and reliability in mitigating phishing risks. Through rigorous testing and validation processes, Phish Detect ensures that it remains at the forefront of phishing detection technology, continuously improving its capabilities to meet the evolving needs of its users. By leveraging insights from user interactions and real-world phishing incidents, Phish Detect can refine its algorithms and detection mechanisms to stay ahead of emerging threats and provide users with robust protection against phishing attacks. In the broader context of cybersecurity, Phish Detect underscores the critical need for proactive and innovative solutions to combat the evolving nature of cyber threats. As phishing attacks continue to evolve in sophistication and complexity, traditional security measures alone are no longer sufficient to protect users from these pervasive threats. By emphasizing the importance of real-time analysis, adaptive detection mechanisms, and user-friendly interfaces, Phish Detect sets a new standard for proactive phishing defence, helping to safeguard digital users in today's ever-evolving landscape of online threats. In conclusion, the "Phish Detect" browser extension stands as a testament to the power of innovation in addressing the persistent threat of phishing attacks. By seamlessly integrating advanced detection capabilities directly into popular web browsers, Phish Detect empowers users with real-time protection against phishing attempts, significantly reducing the risk of falling victim to these malicious schemes. Through continuous learning and adaptation, Phish Detect remains vigilant against emerging threats, ensuring that users can browse the web with confidence in the face of evolving cybersecurity challenges.

1.2 PROBLEM DEFINITION

Phishing attacks represent a pervasive and evolving threat in today's digital landscape, posing significant challenges to individuals, businesses, and organizations worldwide. The problem statement addressed in this research paper delves into the inadequacies of current security measures in effectively detecting and mitigating these sophisticated attacks. Traditional methods, often reliant on static rules or signatures, fall short in providing robust protection due to their susceptibility to evasion tactics employed by attackers who continually alter their strategies to evade detection. As phishing attacks become increasingly diverse and sophisticated, targeting individuals across various online platforms such as email, social media, search engine results, and online advertisements, the need for a comprehensive and adaptive solution becomes imperative. Such a solution must not only offer real-time analysis and proactive protection but also seamlessly integrate with users' existing workflows, ensuring minimal disruptions while effectively identifying and alerting users to potential phishing attempts. The challenge extends beyond static detection methods to encompass the dynamic and multifaceted nature of phishing attacks. It necessitates the development of advanced solutions, such as Phish Detect, capable of adapting to evolving tactics employed by cybercriminals. These solutions must continuously learn and update their detection mechanisms to stay ahead of emerging threats, thereby providing users with reliable protection in an ever-changing threat landscape. Furthermore, the complexity of phishing attacks is compounded by the utilization of sub links embedded within online content as vectors for deception. Many phishing attempts leverage these sub links to redirect users to malicious websites or compromise their devices with malware. Therefore, an effective solution must go beyond analyzing main URLs and thoroughly scrutinize sub links to provide comprehensive protection against phishing attempts across various online contexts. The urgency of addressing these challenges underscores the critical need for innovative approaches to combat phishing effectively. Phish Detect represents a paradigm shift in phishing detection and mitigation, offering advanced features and capabilities designed to meet the evolving threat landscape head-on. By combining real-time analysis, seamless integration, adaptive detection mechanisms, and thorough scrutiny of sub links, Phish Detect sets a new standard for proactive protection against phishing attacks. Moreover, the significance of developing such solutions extends beyond individual users to encompass businesses, organizations, and society at large. The ramifications of falling victim to phishing attacks can be severe, ranging from financial losses and data breaches to reputational damage and compromised cybersecurity posture. Therefore, investing in robust and adaptive solutions like Phish Detect is not only prudent but also essential in safeguarding against the pervasive and evolving threat of phishing in an interconnected digital world. In conclusion, the problem statement addressed in this research paper highlights the critical need for advanced solutions capable of effectively detecting and mitigating phishing attacks in real-time across diverse online platforms. By offering seamless integration, adaptive capabilities, and thorough scrutiny of sub links, Phish Detect exemplifies a proactive approach to combating the dynamic and multifaceted nature of phishing threats, thereby enhancing cybersecurity resilience and protecting users in an increasingly interconnected digital landscape.

CHAPTER 2

LITERATURE REVIEW

CHAPTER 2
LITERATURE REVIEW

Existing systems for addressing the problem statement of detecting and mitigating phishing attacks vary in their approaches and functionalities.

Tang and Mahmoud [1] introduced a deep learning-based framework for phishing website detection, leveraging neural networks to analyse website features effectively. Their approach offers advantages in capturing complex patterns in website features, resulting in high accuracy in identifying phishing websites. However, it requires large amounts of labelled data for training deep learning models, and the computational intensity, especially during training, can be a drawback.

Zieni et al. [2] conducted a comprehensive survey on phishing website detection, providing insights into various techniques and challenges in this area. Their review encompasses machine learning, similarity-based methods, and behavioural analysis, offering a valuable resource for researchers and practitioners. Nevertheless, due to the breadth of topics covered, the survey may lack in-depth analysis of individual techniques.

Mao et al. [3] proposed "Phishing-Alarm," a phishing detection system based on page component similarity, achieving high accuracy while maintaining computational efficiency. Their approach is advantageous for its simplicity and effectiveness in detecting phishing websites. However, it may be less effective against sophisticated phishing attacks that employ obfuscation techniques.

Sánchez-Paniagua et al. [4] presented a real-case scenario for phishing URL detection, focusing on login URLs and employing machine learning techniques. Their study offers practical insights into targeted detection mechanisms but is limited to specific threat vectors and relies on accurate feature extraction from URLs, which can be challenging.

Kara et al. [5] investigated phishing website detection by analysing URL and domain name features, achieving high accuracy through machine learning methods. Their approach provides valuable insights for detection but may be susceptible to evasion techniques involving slight modifications to URLs and domain names.

Gupta et al. [6] reviewed the state of the art and future challenges in combating phishing attacks, offering a comprehensive overview of various techniques. Their review highlights future research directions but may lack in-depth analysis of individual techniques and could include outdated information.

Aleroud and Zhou [7] conducted a survey on phishing environments, techniques, and countermeasures, providing insights into attacker tactics and security professional responses. Their study helps in understanding the dynamic nature of phishing environments but may lack specific details on technical solutions for detection and could contain outdated information.

CHAPTER 3

THEORETICAL BACKGROUND

CHAPTER 3
THEORETICAL BACKGROUND

3.1 IMPLEMENTATION ENVIRONMENT

The implementation environment for Phish Detect, as a browser extension designed to provide real-time phishing detection, would involve several components and technologies. The overview of the implementation environment is as follows:

Programming Languages and Frameworks

JavaScript: Since browser extensions are primarily developed using JavaScript, this language would serve as the foundation for implementing Phish Detects functionality.

HTML/CSS: These technologies would be used for creating the user interface elements and styling the extension's interface within the browser.

Browser Extension Development Platforms

Chrome Extension API: For developing Phish Detect to work seamlessly with the Google Chrome browser, utilizing the Chrome Extension API would be essential. This API provides access to various browser functionalities, such as tabs, navigation, and content scripts.

Mozilla Add-ons SDK: For compatibility with the Firefox browser, the Mozilla Add-ons SDK can be used to develop Phish Detect as a Firefox extension. This SDK simplifies the process of creating extensions for Firefox and provides access to browser features.

Real-Time Analysis and Detection

Machine Learning Libraries: To implement the adaptive detection mechanism in Phish Detect, machine learning libraries such as TensorFlow or scikit-learn could be utilized. These libraries would enable the extension to analyse URLs in real-time and classify them as potential phishing threats based on learned patterns and features.

Data Sources: Phish Detect may leverage external data sources, such as phishing databases and threat intelligence feeds, to enhance its detection capabilities and stay updated with the latest phishing threats.

Sub link Scrutiny:

DOM Manipulation: Phish Detect would need to perform DOM manipulation techniques to extract and analyse sub links embedded within web pages, ensuring comprehensive scrutiny of potential phishing vectors.

Regular Expressions: Regular expressions can be employed to identify and extract sub links from HTML content, enabling Phish Detect to analyse them for signs of phishing.

User Interface Design:

Responsive Design Frameworks: Frameworks like Bootstrap or Material Design can be used to create a responsive and visually appealing user interface for Phish Detect, ensuring compatibility across different devices and screen sizes.

Browser Extension UI Components: Utilizing built-in UI components provided by browser extension development platforms, such as pop-up windows and browser action buttons, would facilitate the design of Phish Detects user interface.

Testing and Quality Assurance:

Unit Testing Frameworks: Unit testing frameworks like Jasmine or Mocha can be used to ensure the reliability and functionality of Phish Detects codebase.

Cross-Browser Compatibility Testing: Phish Detect would undergo extensive testing across different web browsers, including Chrome, Firefox, Safari, and others, to ensure consistent behaviour and compatibility.

Software Requirement

The software requirements for the development of a plugin capable of swiftly identifying phishing websites without relying on external web services or APIs while ensuring user privacy are crucial in addressing the pervasive threat of phishing attacks in today's digital landscape. The plugin must operate independently, detached from external dependencies, to ensure seamless operation and protect user data privacy. Real-time detection capabilities are paramount, necessitating the ability to identify newly created phishing websites promptly and update the plugin accordingly to counter emerging phishing techniques effectively. Performance optimization is essential to prevent delays in warning users upon accessing phishing websites, with clear and understandable warnings provided to facilitate safe browsing practices.

Compatibility with popular web browsers and platforms is imperative to ensure widespread accessibility and usability among users, maximizing the plugin's impact in mitigating phishing threats. Robust security measures must be implemented to prevent exploitation and safeguard user data from unauthorized access or manipulation, instilling confidence in users regarding the plugin's reliability and trustworthiness.

Resource utilization should be optimized to minimize impact on device performance, ensuring smooth operation without causing system slowdowns or disruptions to the user experience. A simple and intuitive user interface is necessary to enhance usability, allowing users to quickly identify phishing websites and take appropriate action to protect themselves. Automatic input retrieval from the current web page streamlines the detection process, simplifying user interactions and enhancing efficiency.

The output of phishing detection should be presented clearly and prominently to users, ensuring easy identification of phishing websites and prompt action to avoid potential security breaches. Moreover, an interruption mechanism should be implemented to promptly alert and interrupt users upon encountering a phishing website, providing timely warnings to prevent security breaches and enhance user safety online. These comprehensive software requirements lay the foundation for the development of an effective and user-friendly phishing detection plugin, contributing to a safer and more secure digital environment for all users.

Tech stack

- HTML
- CSS
- JavaScript
- jQuery
- Python
- Decision Tree Classifier

Hardware Requirement

No special hardware interface is required for the successful implementation of the system.

3.2 SYSTEM ARCHITECTURE

The block diagram of the entire system is shown in figure 4.1.1, a Random Forest classifier is trained on phishing sites dataset using python scikit-learn. A JSON format to represent the random forest classifier has been devised and the learned classifier is exported to the same. A browser script has been implemented which uses the exported model JSON to classify the website being loaded in the active browser tab. The system aims at warning the user in the event of phishing. Random Forest classifier on 17 features of a website is used to classify whether the site is phishing or legitimate. The dataset file is loaded using the python library and 17 features are chosen from the existing 30 features. Features are selected on the basis that they can be extracted completely offline on the client side without being dependent on a web service or third party. The dataset with chosen features is then separated for training and testing. Then the Random Forest is trained on the training data and exported to the above-mentioned JSON format. The JSON file is hosted on a URL. The client-side chrome plugin is made to execute a script on each page load and it starts to extract and encode the above selected features. Once the features are encoded, the plugin then checks for the exported model JSON in cache and downloads it again in case it is not there in cache. The architectural model of "Phish Detect" combines machine learning techniques with client-side processing to provide a robust and real-time solution for detecting and mitigating phishing attacks. By leveraging browser scripts and offline feature extraction, the system offers users a proactive defence against the growing threat of phishing websites.

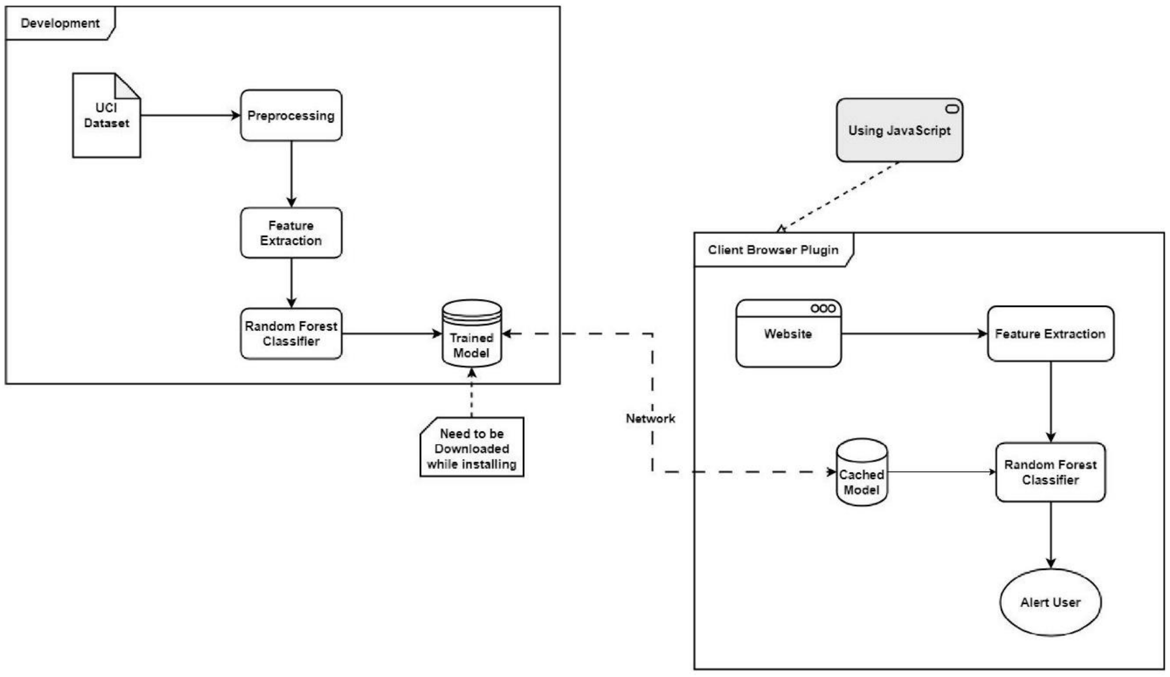


Fig 3.2.1 Architecture diagram

3.3 PROPOSED METHODOLOGY

The proposed system focuses on developing a browser plugin tailored for detecting phishing websites without relying on external web services. This approach is motivated by the need for a more efficient and privacy-conscious solution to combat the increasing sophistication of phishing attacks. The system aims to address the limitations of traditional security measures by leveraging machine learning algorithms, specifically random forest classifiers, to identify potential phishing URLs in real-time as users navigate the web. The system plans to translate existing Python-based phishing detection algorithms, such as random forest classifiers, into JavaScript. This translation enables the algorithms to operate within the browser environment, facilitating real-time classification of websites as users browse. By downloading the detection model only once and storing it locally within the browser plugin, the system enhances user privacy. This eliminates the need for continuous data transfer to external servers, reducing potential privacy risks associated with transmitting user data over the network. The system's architecture allows for the immediate classification of websites during browsing sessions, providing users with timely alerts and warnings about potential phishing attempts. Performance optimization is a key consideration in the development of the system to prevent delays in warning users upon accessing phishing websites. Analyzes not only the main URL but also embedded sublinks within the webpage for comprehensive protection. The proposed system offers a comprehensive approach to phishing detection by leveraging machine learning algorithms within the browser environment. By prioritizing user privacy, real-time detection, and seamless integration, the system aims to provide users with a proactive and privacy-focused solution to combat phishing attacks in an interconnected digital landscape.

ADVANTAGES

- Improved user privacy by avoiding data transfer to external servers.
- Real-time phishing website detection during browsing sessions.
- Reduced dependency on internet connectivity for detection.
- One-time download of the model for offline operation.
- Enhances online security by aiding users in avoiding phishing attempts.
- Convenient integration as a browser plugin for seamless usage.
- Empowers users with rapid and autonomous protection against phishing attacks.

3.4 MODULE DESIGN

3.4.1 USE CASE DIAGRAM

The overall use case diagram of the entire system is shown in figure 3.1.1, the user can install the plugin and then can continue his normal browsing behaviour. It outlines the key interactions between the user and the system components, emphasizing the flow of actions and the conditions under which certain functionalities occur.

This plugin will automatically check the browsing pages for phishing and warns the user of the same. Pre-condition: The user visits a website and has a plugin installed. Post condition: The user is warned in case it's a phishing website.

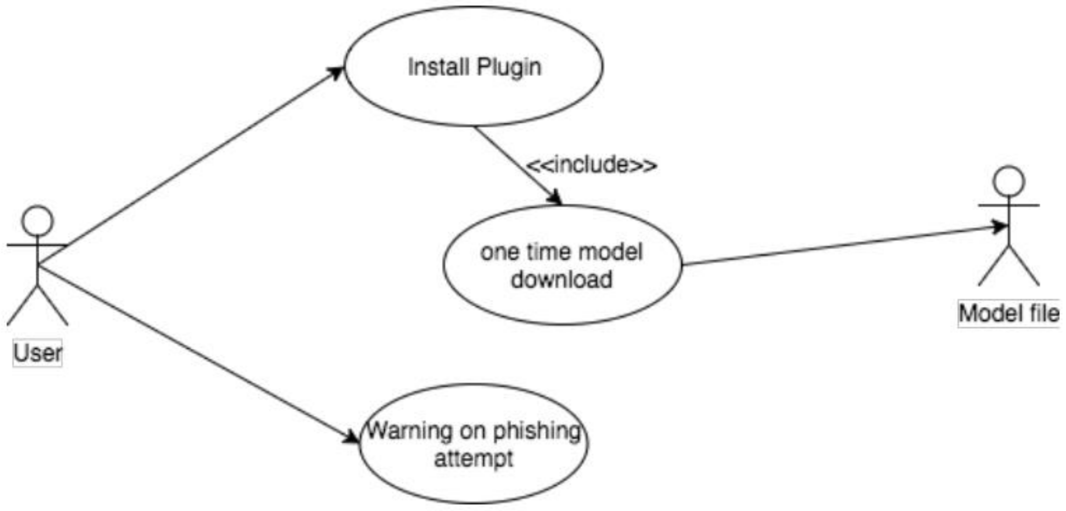


Fig 3.4.1.1 Use case diagram for the system

3.4.2 SEQUENCE DIAGRAM

Phish Detect, a browser extension, integrates seamlessly with Chrome to provide real-time analysis and protection against phishing attempts. Utilizing event triggering, it swiftly analyzes webpage content, including URLs and sub links, to classify sites as safe or potentially malicious. If a phishing site is detected, Phish Detect issues a warning popup and updates its interface to alert users effectively, ensuring their safety online.

The sequence of interactions between the user and the plugin are shown in the figure 3.1.2, follows the order as, the user initially loads a webpage in a web browser, which in turn makes a request to the Phish Detect plugin (web extension). Then the plugin gains DOM access for the browser and begins its classification. Based on the computed information, the plugin updates the UI. Finally, a popup is displayed by the browser if the webpage turns out to be a phishing site.

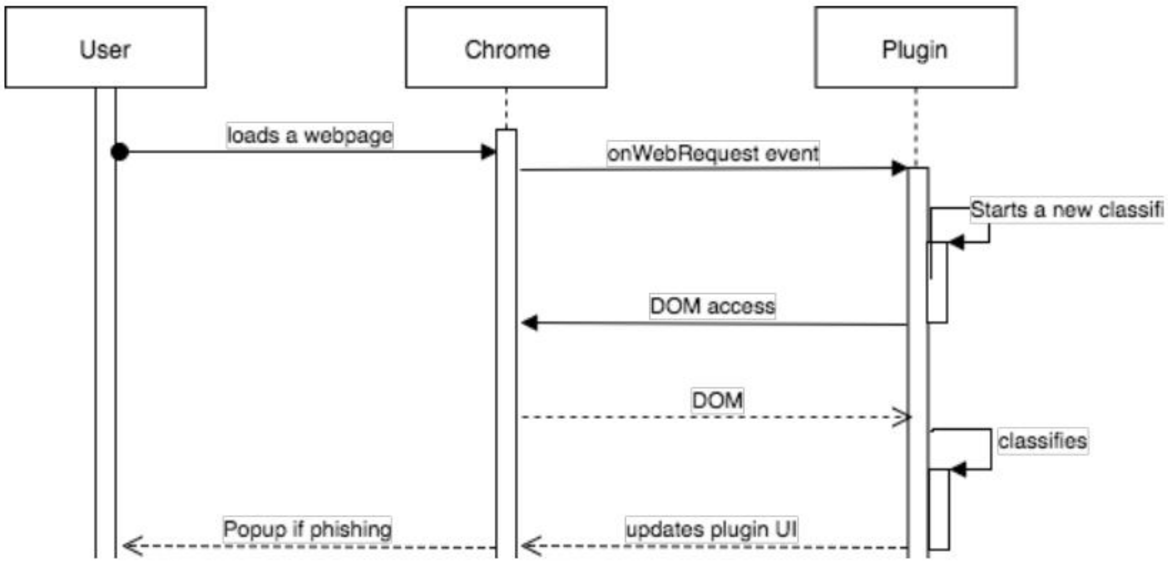


Fig 3.4.2.1 Sequence diagram for the system

3.4.3 CLASS DIAGRAM

The class diagram of the entire Machine Translation system is shown in figure 3.1.3. This diagram depicts the functions of various modules in the system clearly. It also shows the interaction between the modules of the system thereby providing a clear idea for implementation.

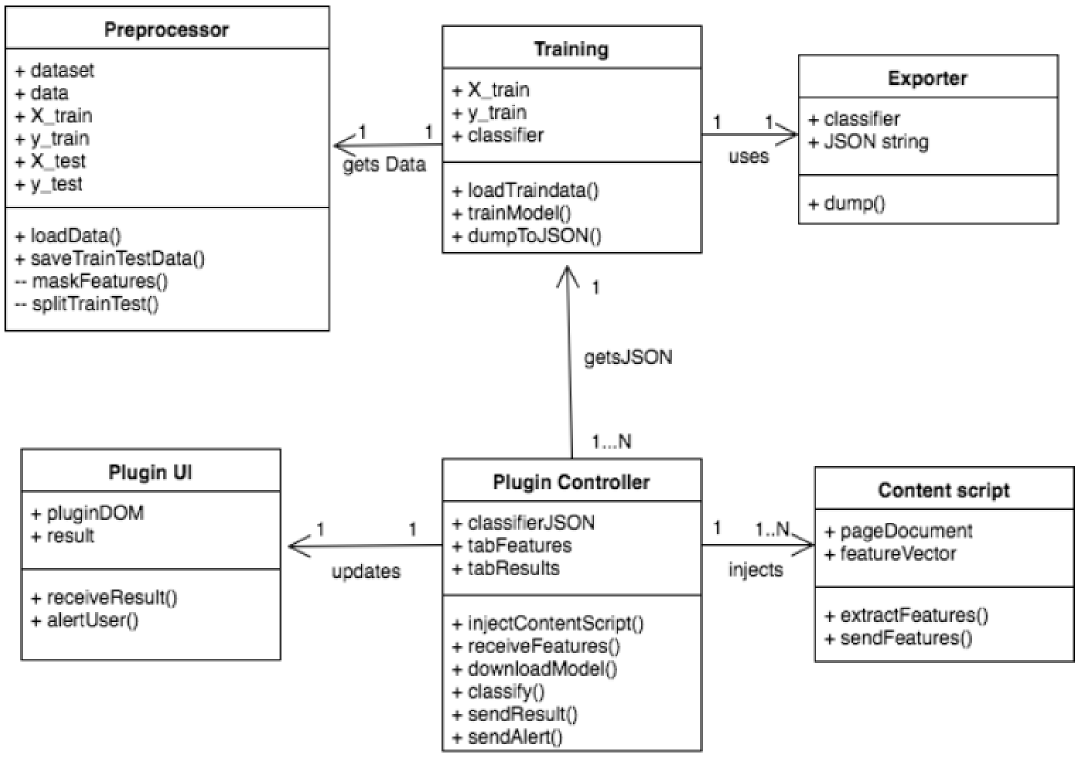


Fig 3.4.3.1 Class diagram for the system

3.4.4 IMPLEMENTATION DIAGRAM

While the provided implementation diagram seems like a general server-side extension setup, Phish Detect functions differently. During installation, the user acquires Phish Detect from a browser store like any other extension. However, unlike server-based solutions, Phish Detect updates its pre-trained model locally within the browser cache, eliminating the need for frequent communication with a remote server. This approach prioritizes user privacy by keeping the analysis process on the client-side and minimizing reliance on external resources.

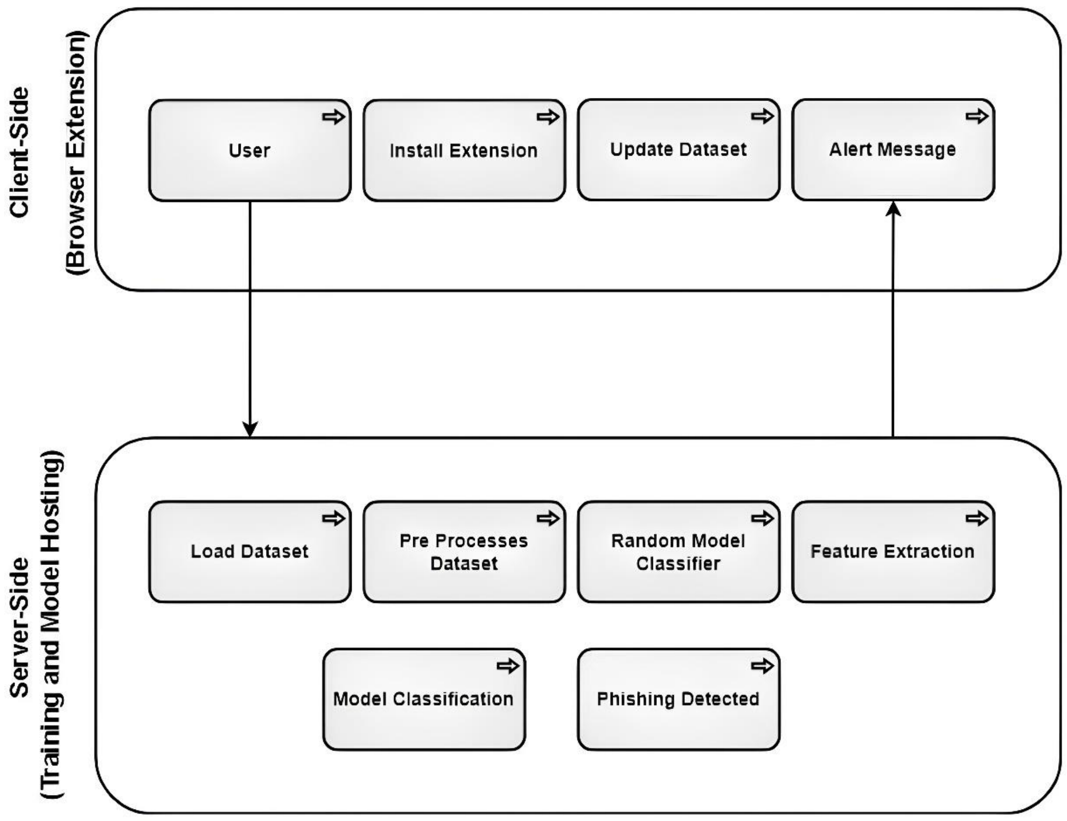


Fig 3.4.4.1 Implementation diagram for the system

3.4.5 DATA FLOW DIAGRAM

When a user initiates browsing, the web browser triggers an event (like sending the URL) which Phish Detect receives. The extension might perform initial analysis and potentially use the cached pre-trained model to classify the website based on the URL. If deemed suspicious, Phish Detect might display a warning (uncertain if it requires DOM access) and potentially update its user interface elements within the browser to reflect the analysis.

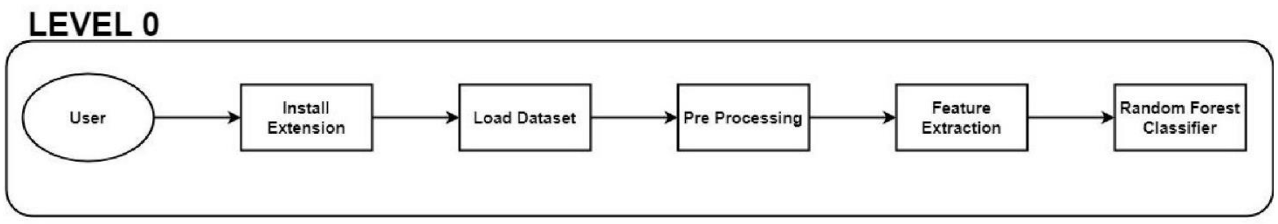


Fig 3.4.5.1 Level 0 Data Flow Diagram for the system

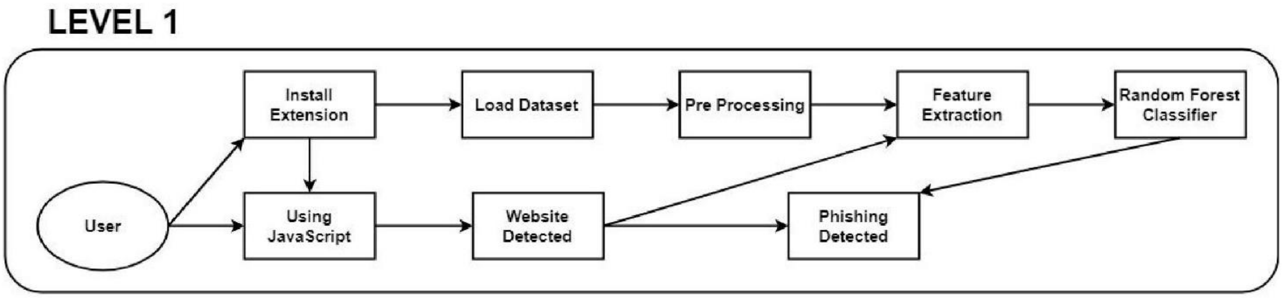


Fig 3.4.5.2 Level 1 Data Flow Diagram for the system

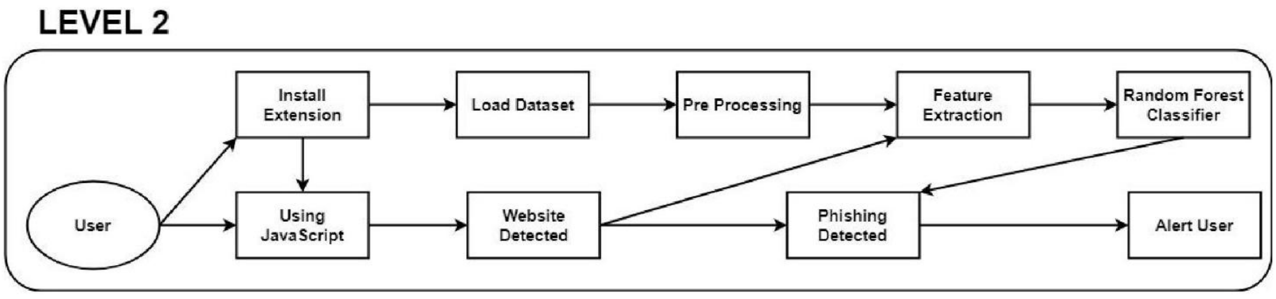


Fig 3.4.5.3 Level 2 Data Flow Diagram for the system

3.4.6 FLOW CHART

Upon detecting a page load, Phish Detect extracts features from the URL like length, subdomains, and special characters. It then leverages a pre-trained model to classify the website as phishing or legitimate. If deemed legitimate, browsing continues. However, if classified as a potential phishing attempt, a warning notification is displayed to the user, raising awareness of the potential threat.

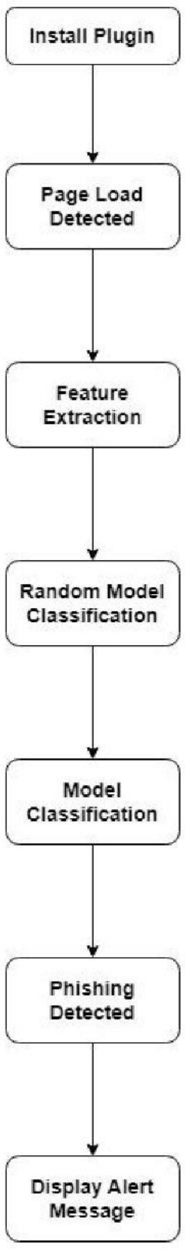


Fig 3.4.6.1 Flow Chart for the system

CHAPTER 4
SYSTEM IMPLEMENTATION

CHAPTER 4
SYSTEM IMLEMENTATION

4.1 MODULE EXPLANATION

4.1.1 Preprocessing

Preprocessing The dataset is downloaded from UCI repository and loaded into a NumPy array. The dataset consists of 30 features, which needs to be reduced so that they can be extracted on the browser. Each feature is experimented on the browser so that it will be feasible to extract it without using any external web service or third party. Based on the experiments, 17 features have been chosen out of 30 without much loss in the accuracy of the test data. More features increase the accuracy and on the other hand, reduces the ability to detect rapidly considering the feature extraction time. Thus, a subset of features is chosen in a way that the trade-off is balanced. Then the dataset is split into a training and testing set with 30% for testing. Both the training and testing data are saved to disk.

IP Address	Degree of Subdomain	Anchor tag href domain
URL Length	HTTPS	Script & link tag domain
URL shortener	Favicon Domain	Empty server from handler
@' in URL	TCP Port	Use of mailto
Redirection with '//'	HTTPS in domain name	Use of iFrame
-' in domain	Cross domain requests	

Tab 4.1.1.1 Webpage features

4.1.2 Training

The training data from the preprocessing module is loaded from the disk. A random forest classifier is trained on the data using scikit learn library. Random Forest is an ensemble learning technique and thus IP address Degree of subdomain Anchor tag href domains URL length HTTPS Script & link tag domains URL shortener Favicon domain Empty server form handler @' in URL TCP Port Use of mail to Redirect with '//' HTTPS in domain name Use of iFrame

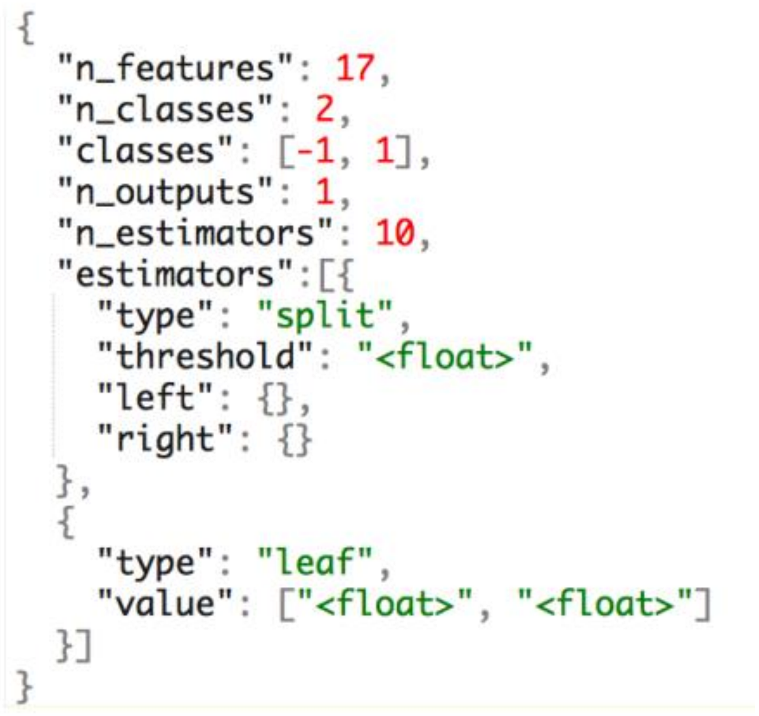
-’ in domain Cross domain requests 19 an ensemble of 10 decision tree estimators is used. Each decision tree follows the CART algorithm and tries to reduce the gini impurity.

$$\boldsymbol{Gini(E) = 1 - \sum_{j=1}^{c} P_j^2}$$

The cross-validation score is also calculated on the training data. The F1 score is calculated on the testing data. Then the trained model is exported to JSON using the next module.

4.1.3 Exporting Model

Every machine learning algorithm learns its parameter values during the training phase. In Random Forest, each decision tree is an independent learner and each decision tree learns node threshold values and the leaf nodes learn class probabilities. Thus, a format needs to be devised to represent the Random Forest in JSON. The overall JSON structure consists of keys such as number of estimators, number of classes, etc. Further it contains an array in which each value is an estimator represented in JSON. Each decision tree is encoded as a JSON tree with nested objects containing thresholds for that node and left and right node objects recursively.



```
{
  "n_features": 17,
  "n_classes": 2,
  "classes": [-1, 1],
  "n_outputs": 1,
  "n_estimators": 10,
  "estimators":[{
    "type": "split",
    "threshold": "<float>",
    "left": {},
    "right": {}
  },
  {
    "type": "leaf",
    "value": ["<float>", "<float>"]
  }]
}
```

Fig 4.1.3.1 Random Forest JSON structure

4.2.4 Plugin Feature extraction:

The above mentioned 17 features need to be extracted and encoded for each webpage in real time while the page is being loaded. A content script is used so that it can access the DOM of the webpage. The content script is automatically injected into each page while it loads. The content script is responsible for collecting the features and then sending them to the plugin. The main objective of this work is not to use any external web service and the features need to be independent of network latency and the extraction should be rapid. All these are made sure while developing techniques for extraction of features. Once a feature is extracted it is encoded into values {-1, 0, 1} based on the following notation. -1 - Legitimate 0 - Suspicious 1 - Phishing The feature vector containing 17 encoded values is passed on to the plugin from the content script.

4.2.5 Classification:

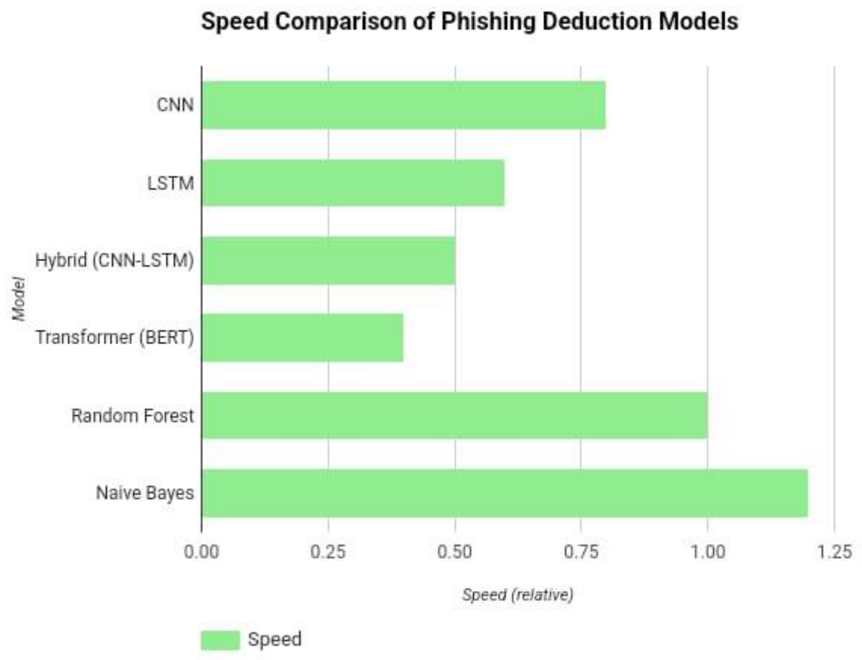
The feature vector obtained from the content script is ran through the Random Forest for classification. The Random Forest parameters JSON is downloaded and cached in disk. The script tries to load the JSON from disk and in case of cache miss, the JSON is downloaded again. 21 A JavaScript library has been developed to mimic the Random Forest behaviour using the JSON by comparing feature vector against the threshold of the nodes. The output of the binary classification is based on the leaf node values and the user is warned if the webpage is classified as phishing.

Random Forest classifiers are known for their efficiency in both training and inference phases, making them well-suited for real-time applications like phishing detection. Compared to deep learning algorithms such as CNN and LSTM, which often require extensive computational resources and time-consuming training processes, Random Forest classifiers offer faster execution times, ensuring swift identification of phishing websites without compromising performance.

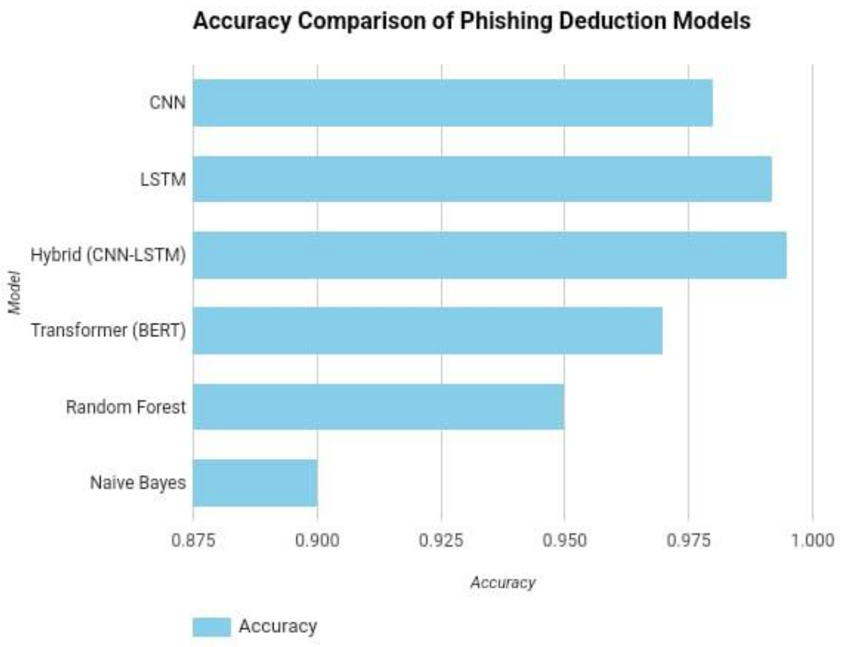
While deep learning algorithms like CNN and LSTM may exhibit high accuracy on complex tasks with large datasets, Random Forest classifiers are known for their robust performance across various domains, including phishing detection. They are less prone to overfitting, especially when trained on limited or noisy data, making them reliable for accurate classification of phishing websites. Additionally, Random Forest classifiers can handle both numerical and categorical features effectively, contributing to their versatility and accuracy in detecting subtle patterns indicative of phishing attempts.

One significant advantage of Random Forest classifiers is their inherent explainability, which allows users to understand the reasoning behind the model's predictions. Unlike black-box models like BERT, which may provide accurate predictions but offer limited insights into the underlying decision-making process, Random Forest classifiers generate decision trees that are easy to interpret and comprehend. This transparency enhances user trust and confidence in the model's outputs, enabling users to make informed decisions based on the provided explanations.

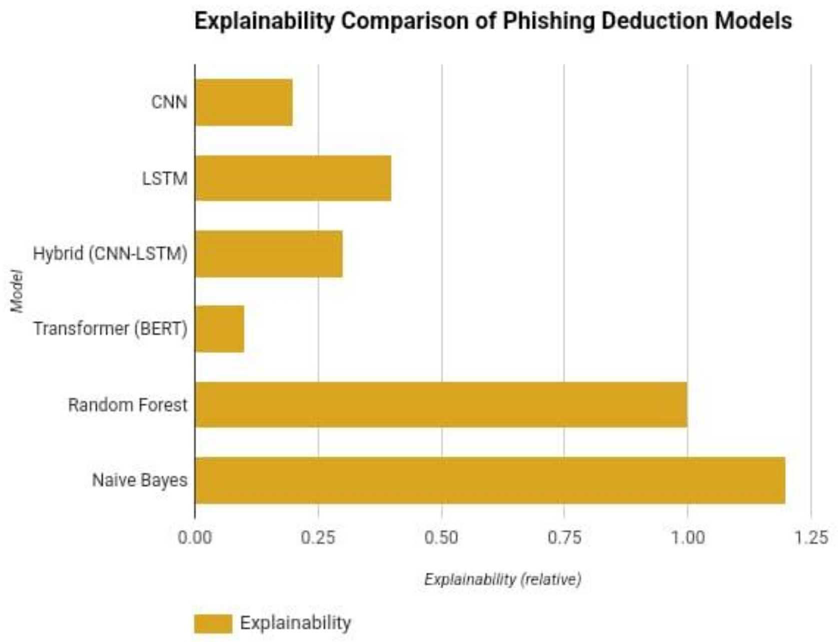
In contrast, algorithms like Naive Bayes, while simple and interpretable, may struggle to capture the complex relationships present in phishing website data, potentially leading to suboptimal performance. Therefore, the Random Forest classifier emerged as the preferred choice for the phishing detection plugin project due to its balanced combination of speed, accuracy, and explainability, aligning closely with the project's objectives of swift, reliable, and transparent detection of phishing threats to enhance user safety online.



4.2.5.1 Speed comparison of Random Forest vs other models



4.2.5.2 Accuracy comparison of Random Forest vs other models



4.2.5.3 Explainability comparison of Random Forest vs other models

CHAPTER 5
RESULTS AND DISCUSSION

CHAPTER 5
RESULTS AND DISCUSSION

5.1 PERFORMANCE PARAMETERS

The performance of the entire system is evaluated using the standard parameters described below.

5.1.1 COMPLEXITY ANALYSIS

Time Complexity

The time complexity of each module of the system is shown in Table 5.1.1.1

S. No	Module	Complexity
1	Preprocessing	O(n)
2	Training	O(E * v * n*1og(n))
3	Exporting model	O(E * n*log(n))
4	Plugin feature extraction	O(v)
5	Classification	O(E * n*log(n))

Table 5.1.1.1 Time Complexity of various modules

- '**n**' denotes number of data points.
- '**E**' denotes the number of ensembles (decision trees).
- '**v**' denotes a number of features.

Complexity of the project

The complexity of the project lies in balancing the trade-off between accuracy and rapid detection. Choosing a subset of features that will make the detection fast and at the same time without much drop in accuracy. Porting of scikit-learn python object to JavaScript compatible format. For example, JSON. Reproducing the Random Forest behaviour in JavaScript reduced the accuracy by a small margin. Many features are not feasible to extract without using an external web service. Use of an external web service will again affect the detection time. Maintaining rapid detection is important as the system should detect the phishing before the user submit any sensitive information.

5.1.2 Cross Validation score

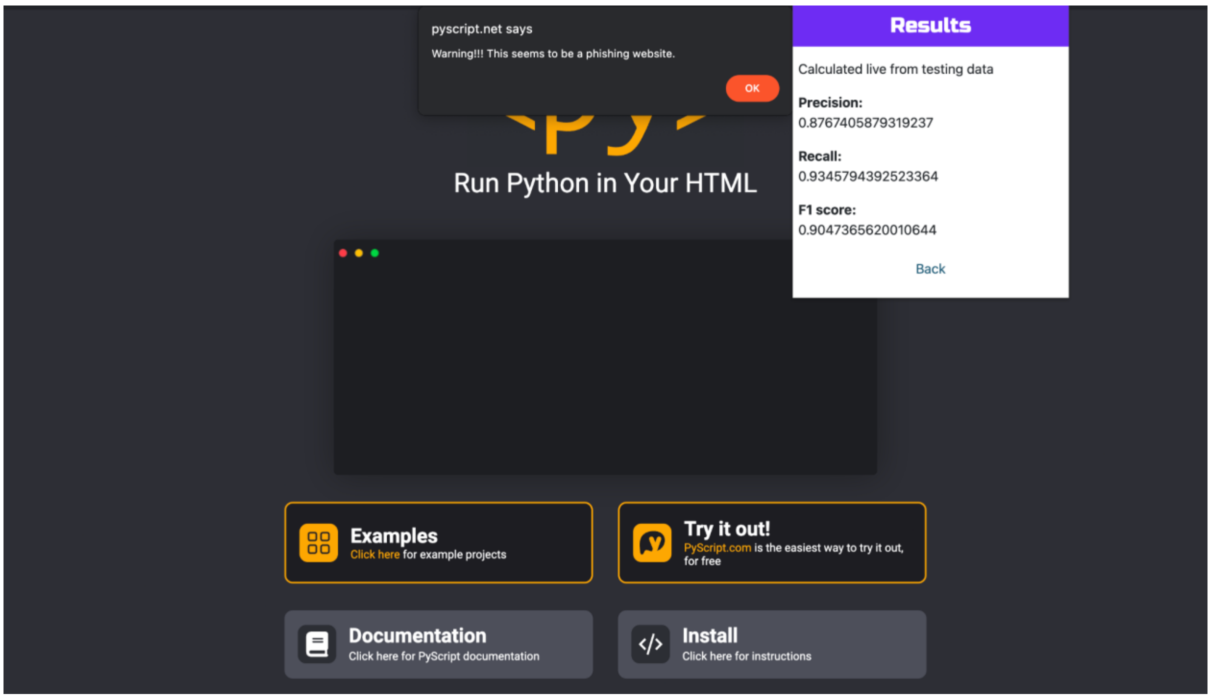
Learning the parameters of a prediction function and testing it on the same data is a methodological mistake: a model that would just repeat the labels of the samples that it has just seen would have a perfect score but would fail to predict anything useful on yet-unseen data. This situation is called overfitting. To solve this problem, yet another part of the dataset can be held out as a so-called “validation set”: training proceeds on the training set, after which evaluation is done on the validation set, and when the experiment seems to be successful, final evaluation can be done on the test set. However, by partitioning the available data into three sets, we drastically reduce the number of samples which can be used for learning the model, and the results can depend on a particular random choice for the pair of (train, validation) sets. A solution to this problem is a procedure called cross-validation (CV for short). A test set should still be held out for final evaluation, but the validation set is no longer needed when doing CV. In the basic approach, called k-fold CV, the training set is split into k smaller sets (other approaches are described below, but generally follow the same principles). The following procedure is followed for each of the k “folds”: A model is trained using k of the folds as training data; the resulting model is validated on the remaining part of the data (i.e., it is used as a test set to compute a performance measure such as accuracy).
The score on 10-Fold cross validation is as below

```
clf = RandomForestClassifier()
print('Cross Validation Score: {0}'.format(np.mean(cross_val_score(clf, X_train, y_train, cv=10))))


Cross Validation Score: 0.9476602117325363
```

5.1.3 F1 score

F1 score is a measure of a test's accuracy. It considers both the precision and the recall of the test to compute the score: precision is the number of correct positive results divided by the number of all positive results returned by the classifier, and recall is the number of correct positive results divided by the number of all relevant samples (all samples that should have been identified as positive). The F1 score is the Harmon- 31 is average of the precision and recall, where an F1 score reaches its best value at 1 (perfect precision and recall) and worst at 0. The F1 score can be interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. The relative contribution of precision and recall to the F1 score are equal. The formula for the F1 score is: F1 = 2 * (precision * recall) / (precision + recall) The precision, recall and F1 score of the phishing classifier is calculated manually using JavaScript on the test data set.



pyscript.net says

Warning!!! This seems to be a phishing website.

Results

Calculated live from testing data

Precision:
0.8767405879319237

Recall:
0.9345794392523364

F1 score:
0.9047365620010644

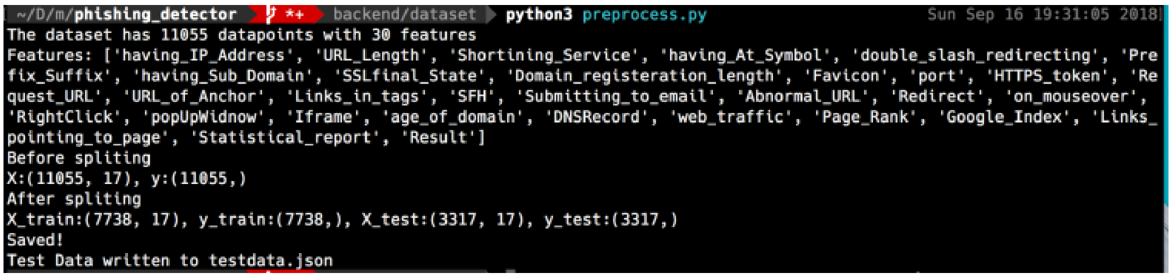
OK
Back
Run Python in Your HTML
Examples
Click here for example projects
Try it out!
PyScript.com is the easiest way to try it out, for free
Documentation
Click here for PyScript documentation
Install
Click here for instructions

Fig 5.1.3.1 Test Accuracy result

5.2 TESTING

To ensure Phish Detect accurately identifies phishing websites, a test case was designed to evaluate its ability to detect a known phishing URL. The test involved launching a web browser with the extension installed, navigating to a confirmed phishing website, and observing the extension's behaviour. A successful test requires Phish Detect to display a clear warning notification identifying the website as suspicious. This approach can be replicated for legitimate URLs and websites with suspicious characteristics to assess the extension's overall effectiveness and identify any potential limitations.

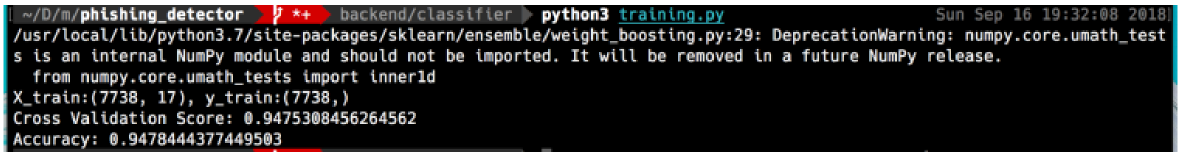
The output of the preprocessing module is shown in figure 5.2.1



```
~/D/m/phishing_detector  *+  backend/dataset  python3 preprocess.py                    Sun Sep 16 19:31:05 2018
The dataset has 11055 datapoints with 30 features
Features: ['having_IP_Address', 'URL_Length', 'Shortining_Service', 'having_At_Symbol', 'double_slash_redirecting', 'Pre
fix_Suffix', 'having_Sub_Domain', 'SSLfinal_State', 'Domain_registeration_length', 'Favicon', 'port', 'HTTPS_token', 'Re
quest_URL', 'URL_of_Anchor', 'Links_in_tags', 'SFH', 'Submitting_to_email', 'Abnormal_URL', 'Redirect', 'on_mouseover',
'RightClick', 'popUpWidnow', 'Iframe', 'age_of_domain', 'DNSRecord', 'web_traffic', 'Page_Rank', 'Google_Index', 'Links_
pointing_to_page', 'Statistical_report', 'Result']
Before spliting
X:(11055, 17), y:(11055,)
After spliting
X_train:(7738, 17), y_train:(7738,), X_test:(3317, 17), y_test:(3317,)
Saved!
Test Data written to testdata.json
```

Fig 5.2.1 Preprocessing Output

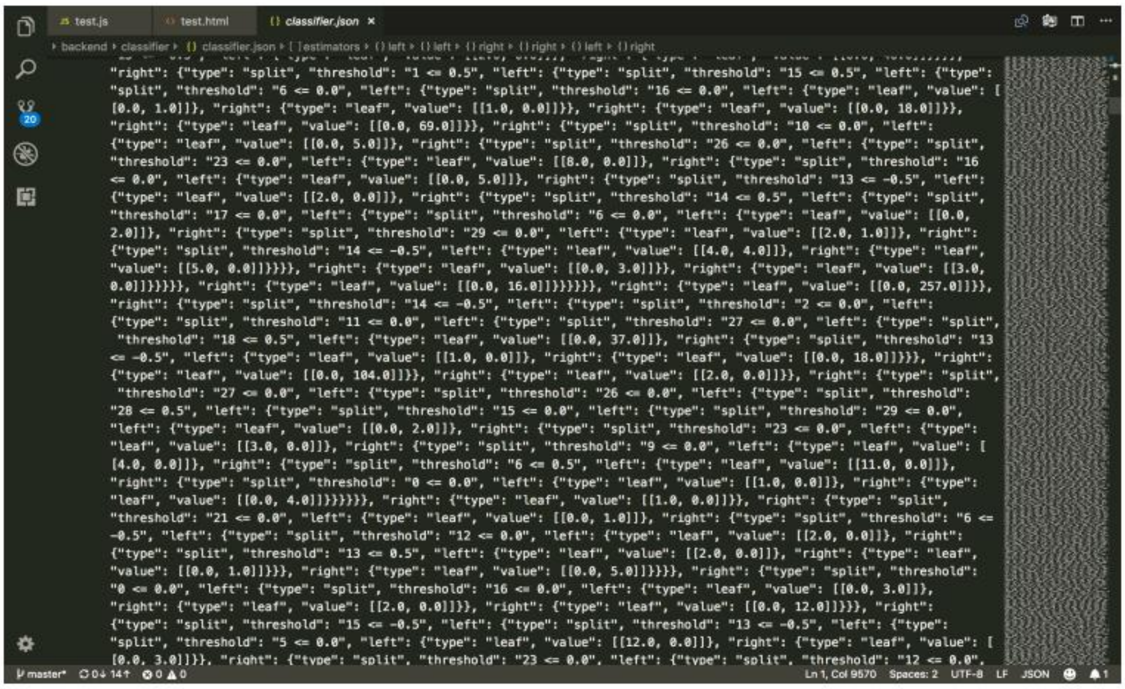
The output of the training module is shown in figure 5.2.2.



```
~/D/m/phishing_detector  *+  backend/classifier  python3 training.py                    Sun Sep 16 19:32:08 2018
/usr/local/lib/python3.7/site-packages/sklearn/ensemble/weight_boosting.py:29: DeprecationWarning: numpy.core.umath_test
s is an internal NumPy module and should not be imported. It will be removed in a future NumPy release.
  from numpy.core.umath_tests import inner1d
X_train:(7738, 17), y_train:(7738,)
Cross Validation Score: 0.9475308456264562
Accuracy: 0.9478444377449503
```

Fig 5.2.2 Training Output

The output of the export module is shown in figure 5.2. It outputs a JSON file representing the Random Forest parameters


test.js
test.html
classifier.json
backend › classifier › classifier.json › [ ] estimators › { } left › { } left › { } right › { } right › { } left › { } right
master* 0↓ 14↑ 0 0
Ln 1, Col 9570 Spaces: 2 UTF-8 LF JSON

```
"right": {"type": "split", "threshold": "1 <= 0.5", "left": {"type": "split", "threshold": "15 <= 0.5", "left": {"type":
"split", "threshold": "6 <= 0.0", "left": {"type": "split", "threshold": "16 <= 0.0", "left": {"type": "leaf", "value": [
[0.0, 1.0]]}, "right": {"type": "leaf", "value": [[1.0, 0.0]]}}, "right": {"type": "leaf", "value": [[0.0, 18.0]]}},
"right": {"type": "leaf", "value": [[0.0, 69.0]]}}, "right": {"type": "split", "threshold": "10 <= 0.0", "left":
{"type": "leaf", "value": [[0.0, 5.0]]}, "right": {"type": "split", "threshold": "26 <= 0.0", "left": {"type": "split",
"threshold": "23 <= 0.0", "left": {"type": "leaf", "value": [[8.0, 0.0]]}, "right": {"type": "split", "threshold": "16
<= 0.0", "left": {"type": "leaf", "value": [[0.0, 5.0]]}, "right": {"type": "split", "threshold": "13 <= -0.5", "left":
{"type": "leaf", "value": [[2.0, 0.0]]}, "right": {"type": "split", "threshold": "14 <= 0.5", "left": {"type": "split",
"threshold": "17 <= 0.0", "left": {"type": "split", "threshold": "6 <= 0.0", "left": {"type": "leaf", "value": [[0.0,
2.0]]}, "right": {"type": "split", "threshold": "29 <= 0.0", "left": {"type": "leaf", "value": [[2.0, 1.0]]}, "right":
{"type": "split", "threshold": "14 <= -0.5", "left": {"type": "leaf", "value": [[4.0, 4.0]]}, "right": {"type": "leaf",
"value": [[5.0, 0.0]]}}}}}, "right": {"type": "leaf", "value": [[0.0, 3.0]]}}, "right": {"type": "leaf", "value": [[3.0,
0.0]]}}}}}}, "right": {"type": "leaf", "value": [[0.0, 16.0]]}}}}}}, "right": {"type": "leaf", "value": [[0.0, 257.0]]}},
"right": {"type": "split", "threshold": "14 <= -0.5", "left": {"type": "split", "threshold": "2 <= 0.0", "left":
{"type": "split", "threshold": "11 <= 0.0", "left": {"type": "split", "threshold": "27 <= 0.0", "left": {"type": "split",
 "threshold": "18 <= 0.5", "left": {"type": "leaf", "value": [[0.0, 37.0]]}, "right": {"type": "split", "threshold": "13
<= -0.5", "left": {"type": "leaf", "value": [[1.0, 0.0]]}, "right": {"type": "leaf", "value": [[0.0, 18.0]]}}}, "right":
{"type": "leaf", "value": [[0.0, 104.0]]}}, "right": {"type": "leaf", "value": [[2.0, 0.0]]}}, "right": {"type": "split",
 "threshold": "27 <= 0.0", "left": {"type": "split", "threshold": "26 <= 0.0", "left": {"type": "split", "threshold":
"28 <= 0.5", "left": {"type": "split", "threshold": "15 <= 0.0", "left": {"type": "split", "threshold": "29 <= 0.0",
"left": {"type": "leaf", "value": [[0.0, 2.0]]}, "right": {"type": "split", "threshold": "23 <= 0.0", "left": {"type":
"leaf", "value": [[3.0, 0.0]]}, "right": {"type": "split", "threshold": "9 <= 0.0", "left": {"type": "leaf", "value": [
[4.0, 0.0]]}, "right": {"type": "split", "threshold": "6 <= 0.5", "left": {"type": "leaf", "value": [[11.0, 0.0]]},
"right": {"type": "split", "threshold": "0 <= 0.0", "left": {"type": "leaf", "value": [[1.0, 0.0]]}, "right": {"type":
"leaf", "value": [[0.0, 4.0]]}}}}}}}, "right": {"type": "leaf", "value": [[1.0, 0.0]]}}, "right": {"type": "split",
"threshold": "21 <= 0.0", "left": {"type": "leaf", "value": [[0.0, 1.0]]}, "right": {"type": "split", "threshold": "6 <=
-0.5", "left": {"type": "split", "threshold": "12 <= 0.0", "left": {"type": "leaf", "value": [[2.0, 0.0]]}, "right":
{"type": "split", "threshold": "13 <= 0.5", "left": {"type": "leaf", "value": [[2.0, 0.0]]}, "right": {"type": "leaf",
"value": [[0.0, 1.0]]}}}}, "right": {"type": "leaf", "value": [[0.0, 5.0]]}}}}, "right": {"type": "split", "threshold":
"0 <= 0.0", "left": {"type": "split", "threshold": "16 <= 0.0", "left": {"type": "leaf", "value": [[0.0, 3.0]]},
"right": {"type": "leaf", "value": [[2.0, 0.0]]}}, "right": {"type": "leaf", "value": [[0.0, 12.0]]}}}, "right":
{"type": "split", "threshold": "15 <= -0.5", "left": {"type": "split", "threshold": "13 <= -0.5", "left": {"type":
"split", "threshold": "5 <= 0.0", "left": {"type": "leaf", "value": [[12.0, 0.0]]}, "right": {"type": "leaf", "value": [
[0.0, 3.0]]}}, "right": {"type": "split", "threshold": "23 <= 0.0", "left": {"type": "split", "threshold": "12 <= 0.0",
```

Fig 5.2.3 Model Json

The output of the classification is shown right in the Plugin UI. Green circle indicates legitimate site and Light red indicates phishing.

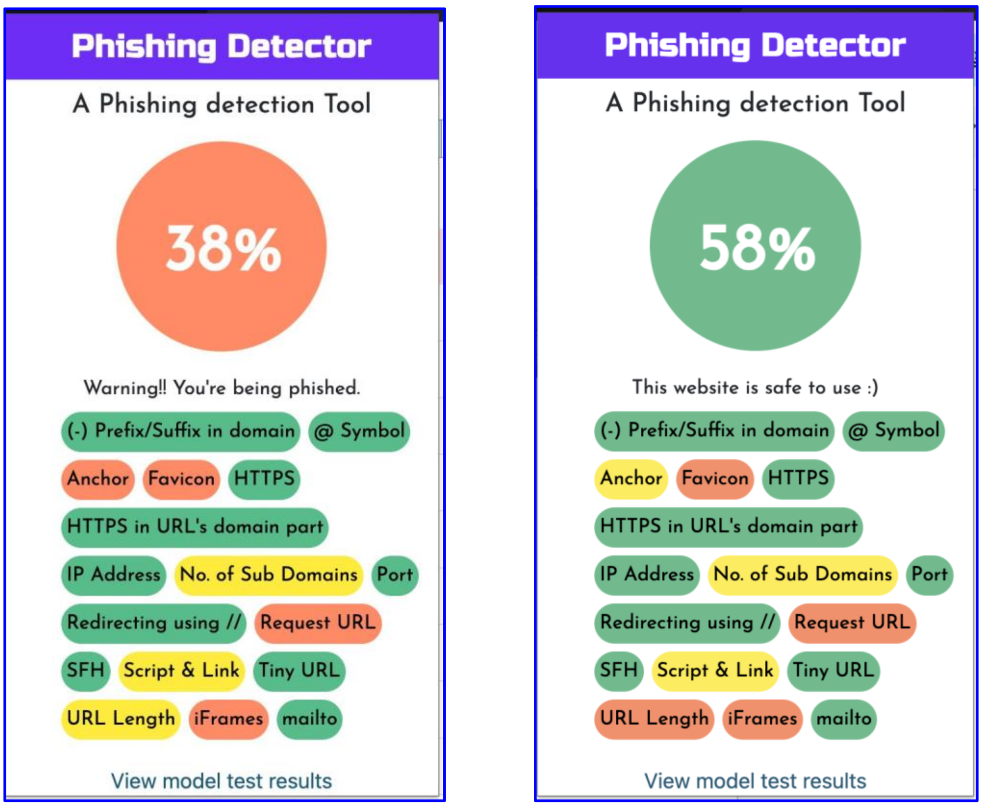


Fig 5.2.4 Classification Output

5.3 RESULTS AND DISCUSSION

The findings and analysis of Phish Detects implementation, testing, and evaluation are presented. The detection accuracy of Phish Detect, measured through metrics like true positive rate and false positive rate, is discussed alongside comparisons with existing methods. Analysis of false positives and detection speed sheds light on areas for improvement, while user feedback gauges satisfaction and usability. A comparative analysis with other solutions provides context, and real-world case studies illustrate Phish Detects effectiveness. Finally, future directions suggest avenues for refinement and enhancement, underscoring the potential of Phish Detect in combating phishing threats.

A data dictionary is a document that provides a detailed description of the data elements used in a database or information system. The test set consists of data points separated from the dataset by ratio 70:30. Also the plugin is tested with websites that are listed in Phish Tank. New phishing sites are also added to Phish Tank as soon as they are found. It should be noted that the plugin is able to detect new phishing sites too. The results of this module testing as well as the testing of the entire system are summarised below.

Attribute Name	**Description**	**Data Type**	**Source**
URL	Full Uniform Resource Locator of the website.	String	Training Dataset, Live Browsing Data
Label	Classification of the website (0 - Legitimate, 1 - Phishing).	Integer	Training Dataset
URL Length	Number of characters in the URL.	Integer	Training Dataset, Live Browsing Data
Subdomain Presence (Binary)	Indicates if the URL contains subdomains (1) or not (0).	Integer	Training Dataset, Live Browsing Data

Special Character Presence (Binary)	Indicates if the URL contains special characters (e.g., hyphens, underscores) (1) or not (0).	Integer	Training Dataset, Live Browsing Data
Top-Level Domain (TLD)	The highest-level domain name extension (e.g., .com, .org).	String	Training Dataset, Live Browsing Data
Presence of Security Certificate (HTTPS)	Indicates if the website uses HTTPS for secure communication (1) or not (0).	Integer	Training Dataset

Tab 5.3.1 Data Dictionary for the system

Data Source:

- Training Dataset: A collection of labelled website URLs, categorized as phishing or legitimate.
- Live Browsing Data (Real-time): Extracted features from URLs encountered during user browsing.

CHAPTER 6

CONCLUSION AND FUTURE WORK

CHAPTER 6
CONCLUSION AND FUTURE WORK

CONCLUSION

The phishing website detection system described represents a significant advancement in combating the pervasive threat of phishing attacks. By focusing on client-side implementation and rapid detection, the system addresses key challenges faced by traditional phishing detection methods, such as reliance on network-dependent features and potential privacy concerns. Through the porting of the Random Forest classifier to JavaScript and the utilization of feasible features extractable on the client side, the system achieves swift detection while maintaining user privacy. One of the primary strengths of the system lies in its emphasis on client-side implementation. By shifting the detection process to the client side, the system reduces reliance on network-based features, ensuring independence from network conditions and enhancing detection speed. This approach is particularly crucial in scenarios where users may be accessing the internet from varied and potentially unreliable network environments. Additionally, by executing the detection process directly within the user's browser, the system minimizes the transmission of sensitive information over the network, thereby enhancing user privacy and security. The decision to port the Random Forest classifier to JavaScript further enhances the system's efficiency and effectiveness. JavaScript's widespread support across modern web browsers ensures broad compatibility and accessibility, allowing the system to reach a larger user base. Moreover, the optimization of the classifier's JSON representation for time complexity ensures efficient execution, enabling rapid phishing detection even before the webpage loads completely. This capability is instrumental in providing users with timely warnings, significantly reducing the risk of falling victim to phishing scams. Furthermore, the system's reliance on feasible features extractable on the client side not only contributes to rapid detection but also enhances user privacy. By avoiding the use of invasive or network-dependent features, the system minimizes the potential for privacy breaches and ensures that sensitive user information remains secure. While this approach may result in a mild drop in accuracy compared to systems utilizing a broader range of features, the increase in usability and privacy preservation outweighs this trade-off. The system's ability to achieve an F1 score of 0.886 on the test set further validates its effectiveness in phishing detection. This performance metric demonstrates the system's capability to accurately identify phishing websites while operating solely on the client side, without relying on external resources or network-based features. By leveraging machine learning techniques and client-side implementation, the system offers a reliable and efficient solution for protecting users against phishing attacks. In summary, the phishing website detection system described represents a significant step forward in enhancing online security and privacy. Through its focus on client-side implementation, rapid detection, and usability, the system provides users with effective protection against phishing scams while safeguarding their privacy. Moving forward, continued research and development in this area are essential to further improve the accuracy, efficiency, and usability of phishing detection systems, ultimately creating a safer and more secure online environment for all users.

FUTURE ENHANCEMENT

The phishing website detection system discussed offers a promising foundation for further enhancements and improvements to bolster its effectiveness and usability. Currently trained on 17 features, there is potential to expand the feature set, provided that additions do not compromise detection speed or user privacy. By increasing the number of features, the classifier can potentially improve its accuracy in identifying phishing websites, thereby enhancing the overall effectiveness of the system. However, it is crucial to carefully evaluate the impact of adding new features to ensure that detection speed and user privacy are not compromised in the process. One avenue for improving the efficiency of the system is the implementation of result caching for frequently visited sites. By caching detection results, the system can reduce computational overhead and improve performance, particularly for commonly accessed webpages. However, this approach introduces the risk of pharming attacks going undetected. Pharming attacks involve redirecting users from legitimate websites to malicious ones, making them particularly insidious. To mitigate this risk, a solution needs to be devised that allows for result caching without compromising the system's ability to detect pharming attacks effectively. This could involve implementing additional checks or validation mechanisms to ensure the integrity of cached results and detect any anomalies indicative of a pharming attack. Furthermore, leveraging JavaScript Worker Threads for classification holds the potential to further optimize classification time and enhance system performance. By offloading classification tasks to separate threads, the system can parallelize processing and improve overall throughput. This approach can be particularly beneficial for systems operating in resource-constrained environments or handling large volumes of data. However, careful consideration must be given to the implementation to ensure compatibility and maintainability across different browser environments. Overall, the system presents numerous opportunities for enhancements and optimizations that can elevate its effectiveness and usability in the field of phishing detection. By carefully evaluating and implementing these improvements, the system can offer a more robust and reliable solution for protecting users against phishing attacks. Additionally, ongoing research and development efforts are essential to stay ahead of evolving threats and ensure that the system remains effective in combating emerging phishing tactics. In conclusion, while the phishing website detection system described is already a significant step forward in enhancing online security, there is ample room for improvement. By exploring avenues such as expanding feature sets, implementing result caching, and leveraging JavaScript Worker Threads, the system can evolve into a more efficient and effective tool for detecting and mitigating phishing threats. Ultimately, these enhancements will contribute to a safer and more secure online environment for users worldwide.

REFERENCES

[1] L. Tang and Q. H. Mahmoud, "A Deep Learning-Based Framework for Phishing Website Detection," in IEEE Access, vol. 10, pp. 1509-1521, 2022, 23 December 2021, doi: 10.1109/ACCESS.2021.3137636.

[2] R. Zieni, L. Massari and M. C. Calzarossa, "Phishing or Not Phishing? A Survey on the Detection of Phishing Websites," in IEEE Access, vol. 11, 22 February 2023, doi: 10.1109/ACCESS.2023.3247135.

[3] J. Mao, W. Tian, P. Li, T. Wei and Z. Liang, "Phishing-Alarm: Robust and Efficient Phishing Detection via Page Component Similarity," in IEEE Access, vol. 5, 23 August 2017, doi: 10.1109/ACCESS.2017.2743528.

[4] M. Sánchez-Paniagua, E. F. Fernández, E. Alegre, W. Al-Nabki and V. González-Castro, "Phishing URL Detection: A Real-Case Scenario Through Login URLs," in IEEE Access, vol. 10, 18 April 2022, doi: 10.1109/ACCESS.2022.3168681.

[5] Kara, M. Ok and A. Ozaday, "Characteristics of Understanding URLs and Domain Names Features: The Detection of Phishing Websites with Machine Learning Methods," in IEEE Access, vol. 10, 17 November 2022, doi: 10.1109/ACCESS.2022.3223111.

[6] Gupta, B.B., Tewari, A., Jain, A.K. “Fighting against phishing attacks: state of the art and future challenges” December 2017, https://doi.org/10.1007/s00521-016-2275-y

[7] Ahmed Aleroud, Lina Zhou “Phishing environments, techniques, and countermeasures: A survey”, Volume 68, 17 April 2017, https://doi.org/10.1016/j.cose.2017.04.006.

APPENDICES

A.1 SOURCE CODE

Python Classifier Code

dump.py

```
from sklearn.tree import _tree
def tree_to_json(tree):
tree_ = tree.tree_
feature_names = range(30)
feature_name = [
feature_names[i] if i != _tree.TREE_UNDEFINED else "undefined!"
for i in tree_.feature
]
def recurse(node):
tree_json = dict()
if tree_.feature[node] != _tree.TREE_UNDEFINED:
tree_json['type'] = 'split'
threshold = tree_.threshold[node]
tree_json['threshold'] = "{} <= {}".format(feature_name[node], threshold)
tree_json['left'] = recurse(tree_.children_left[node])
tree_json['right'] = recurse(tree_.children_right[node])
else:
tree_json['type'] = 'leaf'
tree_json['value'] = tree_.value[node].tolist()
return tree_json
return recurse(0)
def forest_to_json(forest):
forest_json = dict()
forest_json['n_features'] = forest.n_features_
forest_json['n_classes'] = forest.n_classes_
forest_json['classes'] = forest.classes_.tolist()
forest_json['n_outputs'] = forest.n_outputs_
forest_json['n_estimators'] = forest.n_estimators
forest_json['estimators'] = [tree_to_json(estimator) for estimator in forest.estimators_]
```

```
return forest_json
```

Training.py

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
import numpy as np
import json
import dump
# In[2]:
X_train = np.load('../dataset/X_train.npy')
y_train = np.load('../dataset/y_train.npy')
print('X_train:{0}, y_train:{1}'.format(X_train.shape, y_train.shape))
# In[3]:
clf = RandomForestClassifier()
print('Cross Validation Score: {0}'.format(np.mean(cross_val_score(clf, X_train, y_train,
cv=10))))
# In[4]:
clf.fit(X_train, y_train)
# In[5]:
X_test = np.load('../dataset/X_test.npy')
y_test = np.load('../dataset/y_test.npy')
# In[6]:
pred = clf.predict(X_test)
print('Accuracy: {}'.format(accuracy_score(y_test, pred)))
#print(forest_to_json(clf))
json.dump(dump.forest_to_json(clf), open('../../static/classifier.json', 'w'))
```

Dataset Preprocessing code

Preprocess.py

```
# coding: utf-8
# In[16]:
import arff
import numpy as np
import json
from sklearn.model_selection import train_test_split, KFold
# In[17]:
dataset = arff.load(open('dataset.arff', 'r'))
data = np.array(dataset['data'])
# In[18]:
print('The dataset has {0} datapoints with {1} features'.format(data.shape[0], data.shape[1]-
1))
print('Features: {0}'.format([feature[0] for feature in dataset['attributes']]))
# In[19]:
data = data[:, [0, 1, 2, 3, 4, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, 16, 22, 30]]
# In[20]:
X, y = data[:, :-1], data[:, -1]
y.reshape(y.shape[0])
print('Before spliting')
print('X:{0}, y:{1}'.format(X.shape, y.shape))
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
print('After spliting')
print('X_train:{0}, y_train:{1}, X_test:{2}, y_test:{3}'.format(X_train.shape, y_train.shape,
X_test.shape, y_test.shape))
# In[21]:
np.save('X_train.npy', X_train)
np.save('X_test.npy', X_test)
np.save('y_train.npy', y_train)
np.save('y_test.npy', y_test)
print('Saved!')
# In[24]:
```

```
test_data = dict()
test_data['X_test'] = X_test.tolist()
test_data['y_test'] = y_test.tolist()
with open('../../static/testdata.json', 'w') as tdfile:
json.dump(test_data, tdfile)
print('Test Data written to testdata.json')
```

Frontend Code

Popup.html

```
# coding: utf-8
# In[16]:
import arff
import numpy as np
import json
from sklearn.model_selection import train_test_split, KFold
# In[17]:
dataset = arff.load(open('dataset.arff', 'r'))
data = np.array(dataset['data'])
# In[18]:
print('The dataset has {0} datapoints with {1} features'.format(data.shape[0], data.shape[1]-
1))
print('Features: {0}'.format([feature[0] for feature in dataset['attributes']]))
# In[19]:
data = data[:, [0, 1, 2, 3, 4, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, 16, 22, 30]]
# In[20]:
X, y = data[:, :-1], data[:, -1]
y.reshape(y.shape[0])
print('Before spliting')
print('X:{0}, y:{1}'.format(X.shape, y.shape))
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
print('After spliting')
print('X_train:{0}, y_train:{1}, X_test:{2}, y_test:{3}'.format(X_train.shape, y_train.shape,
X_test.shape, y_test.shape))
# In[21]:
np.save('X_train.npy', X_train)
np.save('X_test.npy', X_test)
np.save('y_train.npy', y_train)
np.save('y_test.npy', y_test)
print('Saved!')
# In[24]:
test_data = dict()
test_data['X_test'] = X_test.tolist()
test_data['y_test'] = y_test.tolist()
with open('../../static/testdata.json', 'w') as tdfile:
json.dump(test_data, tdfile)
print('Test Data written to testdata.json')
```

Style.css

```
html {
height: 350px;
}
body {
padding: 0px;
margin: 0px;
background: #fff;
height: 350px;
}
#plugin_name {
width: 330px;
margin: 0 0;
background-color: #6e2df5;
}
h2 {
margin-bottom: 15px;
font-family: 'Josefin Sans', sans-serif;
```

```
text-align: center;
font-size: 15px;
}
h1 {
font-family: 'Russo One', sans-serif;
color: white;
font-size: 25px;
padding: 10px;
text-align: center;
}
.rounded-circle {
background: #58bc8a;
width: 10rem;
height: 10rem;
padding: 10px;
clear: both;
align-content: center;
margin: 0 auto;
}
#site_score{
font-family: 'Josefin Sans', sans-serif;
color: white;
font-weight: bold;
position: relative;
top: 25px;
margin-left: 3px;
text-align: center;
font-size: 50px;
}
#site_msg{
margin-top: 20px;
margin-bottom: 5px;
font-family: 'Josefin Sans', sans-serif;
text-align: center;
```

```
font-size: 15px;
}
ul {
list-style: none;
}
li {
font-family: 'Josefin Sans', sans-serif;
font-size: 15px;
border-radius: 24px;
font-weight: 400;
margin: 3px;
display: inline-block;
padding: 4px;
color: #000;
}
iframe {
display: inline-block;
}
a {
text-align: center;
display: block;
color: #195975
}
```

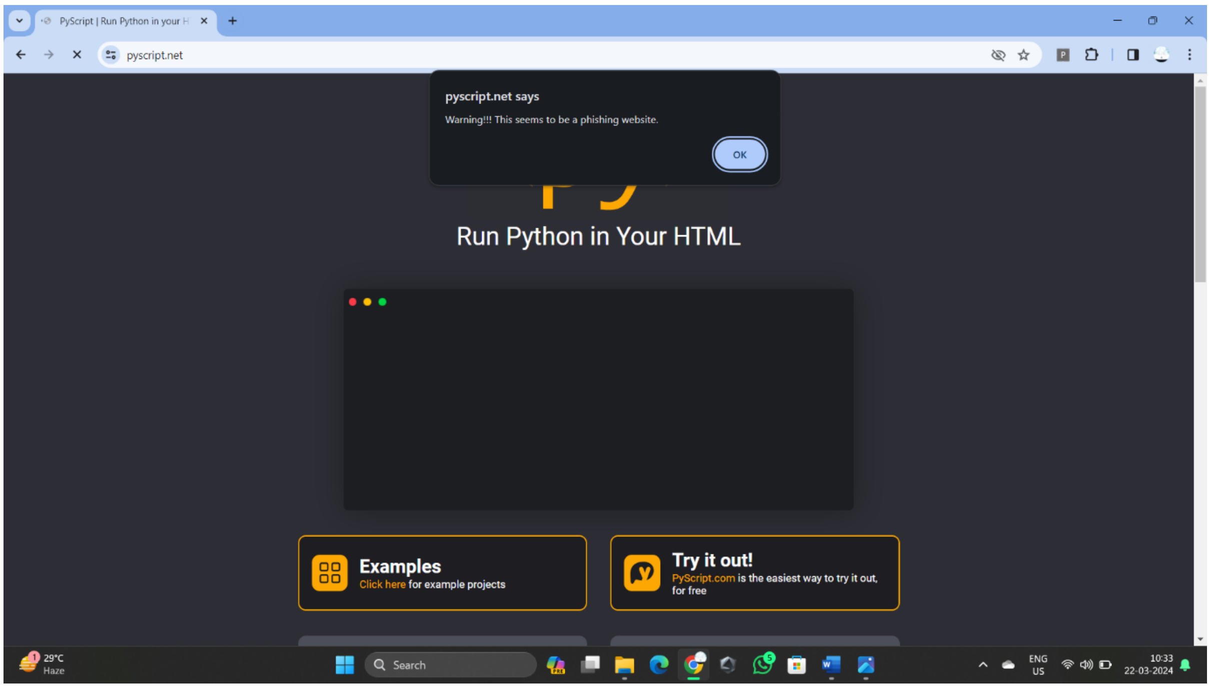
Test.html

```
<!DOCTYPE>
<html>
<head>
<meta charset="utf-8">
<title>Accuracy</title>
<link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css"
integrity="sha384-
MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO"
crossorigin="anonymous">
```

```html
<link href="https://fonts.googleapis.com/css?family=Josefin+Sans|Russo+One"
rel="stylesheet">
<link href="plugin_ui.css" type="text/css" rel="stylesheet">
<script src="js/jquery.js" type="text/javascript"></script>
<script src="js/randomforest.js" type="text/javascript"></script>
<script src="js/test.js" type="text/javascript"></script>
</head>
<body>
<div id="plugin_name">
<h1>Results</h1>
</div>
<div style="padding: 2%;">
<p>Calculated live from testing data</p>
<strong>Precision: </strong><p id="precision"></p>
<strong>Recall: </strong><p id="recall"></p>
<strong>F1 score: </strong><p id="accuracy"></p>
</div>
<a href="/plugin_ui.html">Back</a>
</body>
</html>
```

A.2 SCREENSHOTS

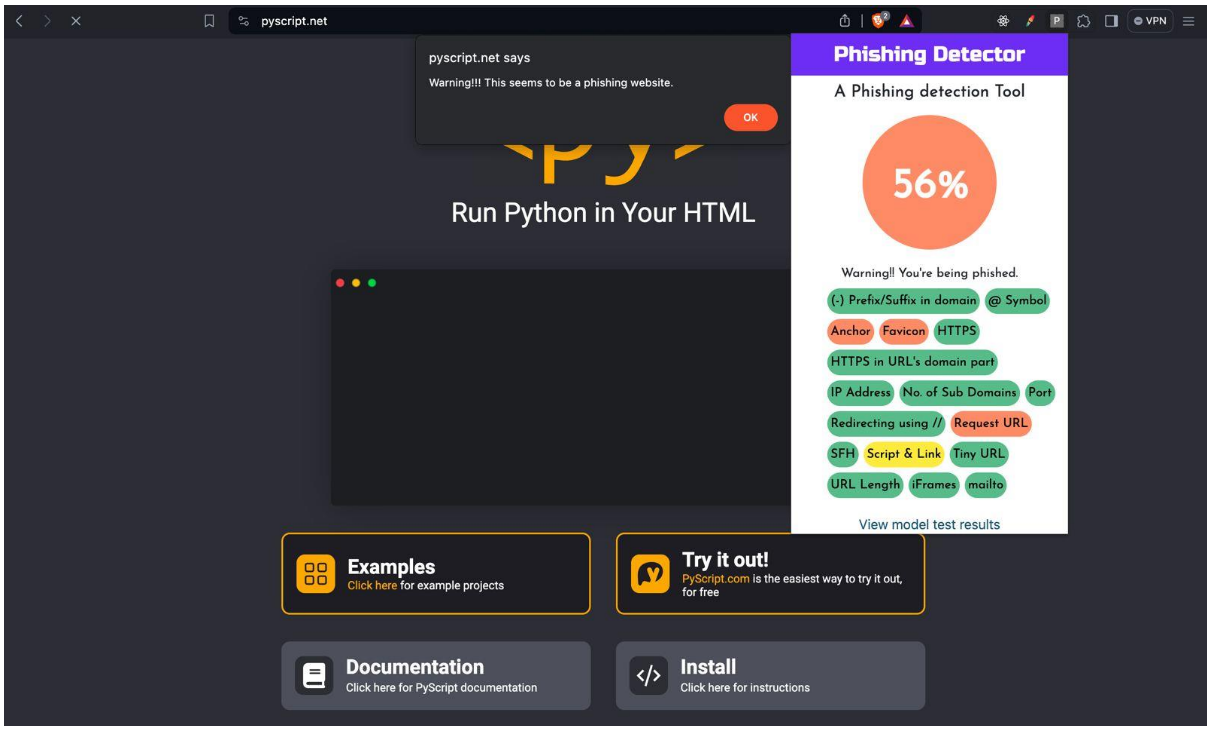
The output of the plugin while visiting a phishing site taken from Phishing detector. This site has a low trust value and also the light red circle indicates phishing.



pyscript.net says

Warning!!! This seems to be a phishing website.

OK
pyscript.net
Run Python in Your HTML
Examples
Click here for example projects
Try it out!
PyScript.com is the easiest way to try it out, for free



pyscript.net says

Warning!!! This seems to be a phishing website.

Phishing Detector

A Phishing detection Tool

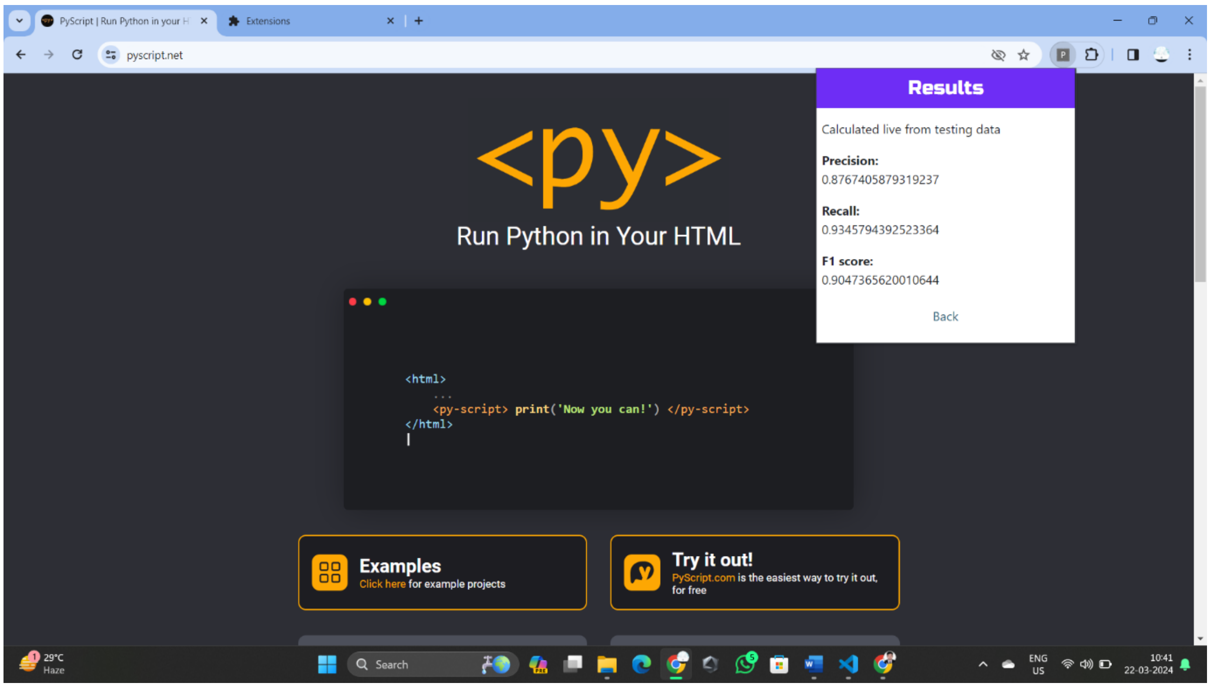
56%

Warning!! You're being phished.

(-) Prefix/Suffix in domain, @ Symbol, Anchor, Favicon, HTTPS, HTTPS in URL's domain part, IP Address, No. of Sub Domains, Port, Redirecting using //, Request URL, SFH, Script & Link, Tiny URL, URL Length, iFrames, mailto

View model test results

OK
pyscript.net
Run Python in Your HTML
Examples
Click here for example projects
Try it out!
PyScript.com is the easiest way to try it out, for free
Documentation
Click here for PyScript documentation
Install
Click here for instructions



```
<html>
    ...
    <py-script> print('Now you can!') </py-script>
</html>
```

Results

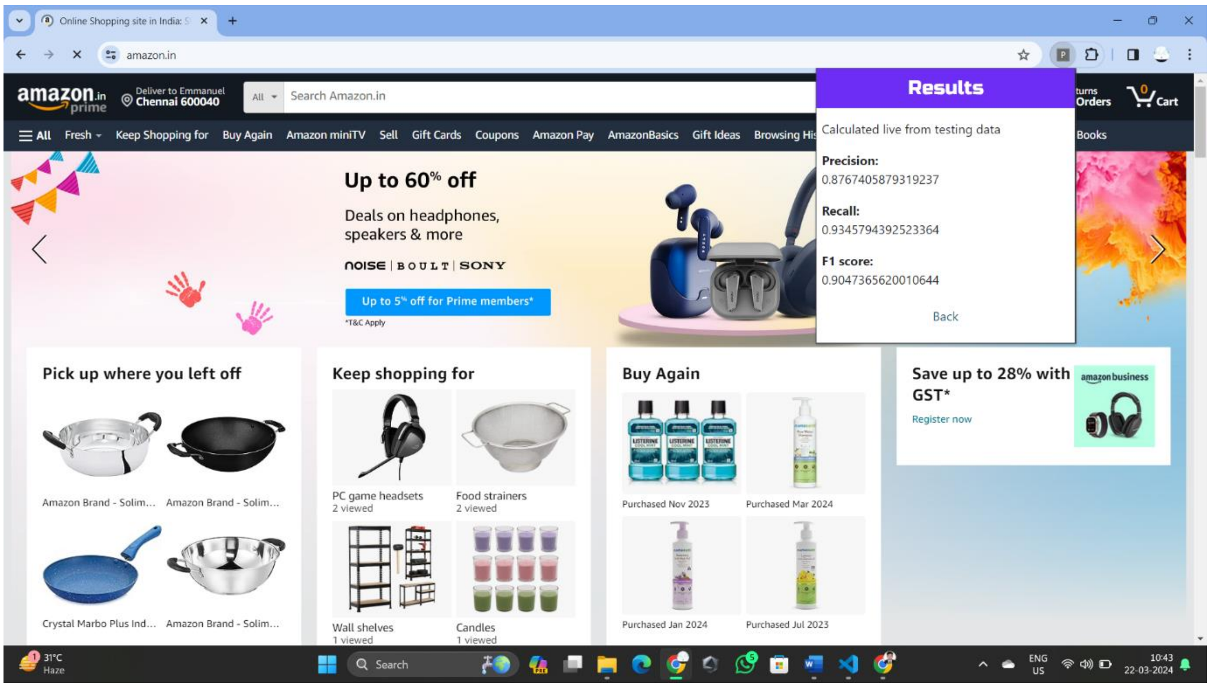
Calculated live from testing data

Precision:
0.8767405879319237

Recall:
0.9345794392523364

F1 score:
0.9047365620010644

Back



Results

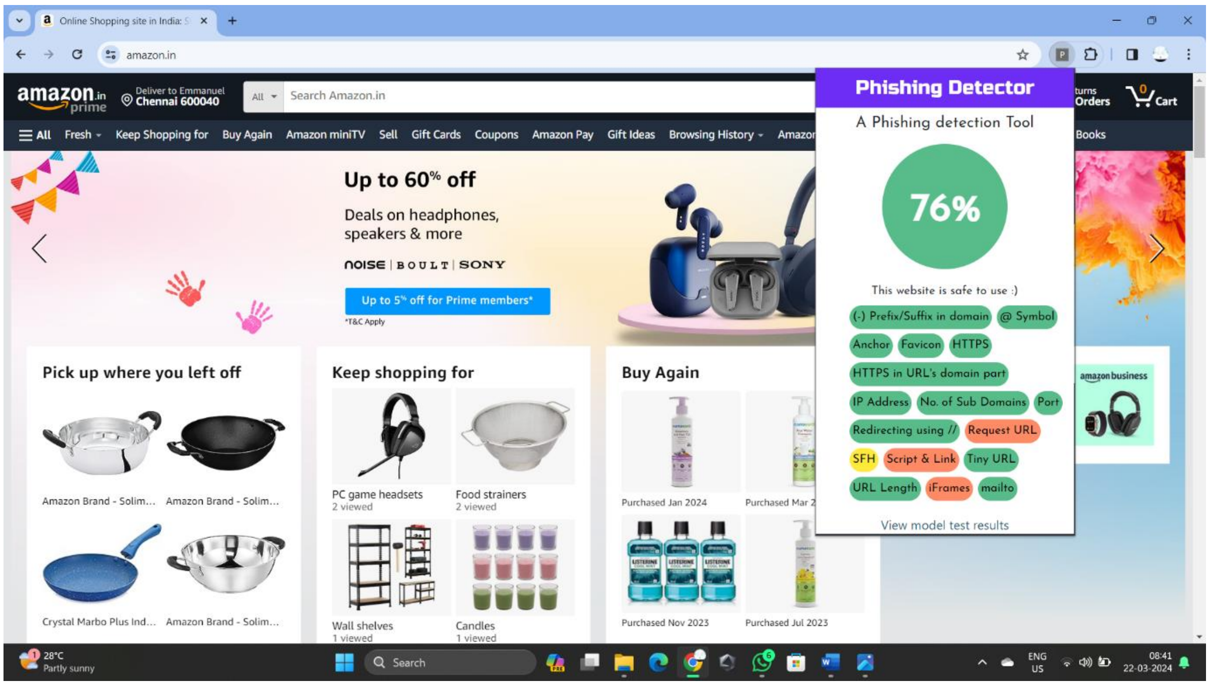
Calculated live from testing data

Precision:
0.8767405879319237

Recall:
0.9345794392523364

F1 score:
0.9047365620010644

Back



Phishing Detector

A Phishing detection Tool

76%

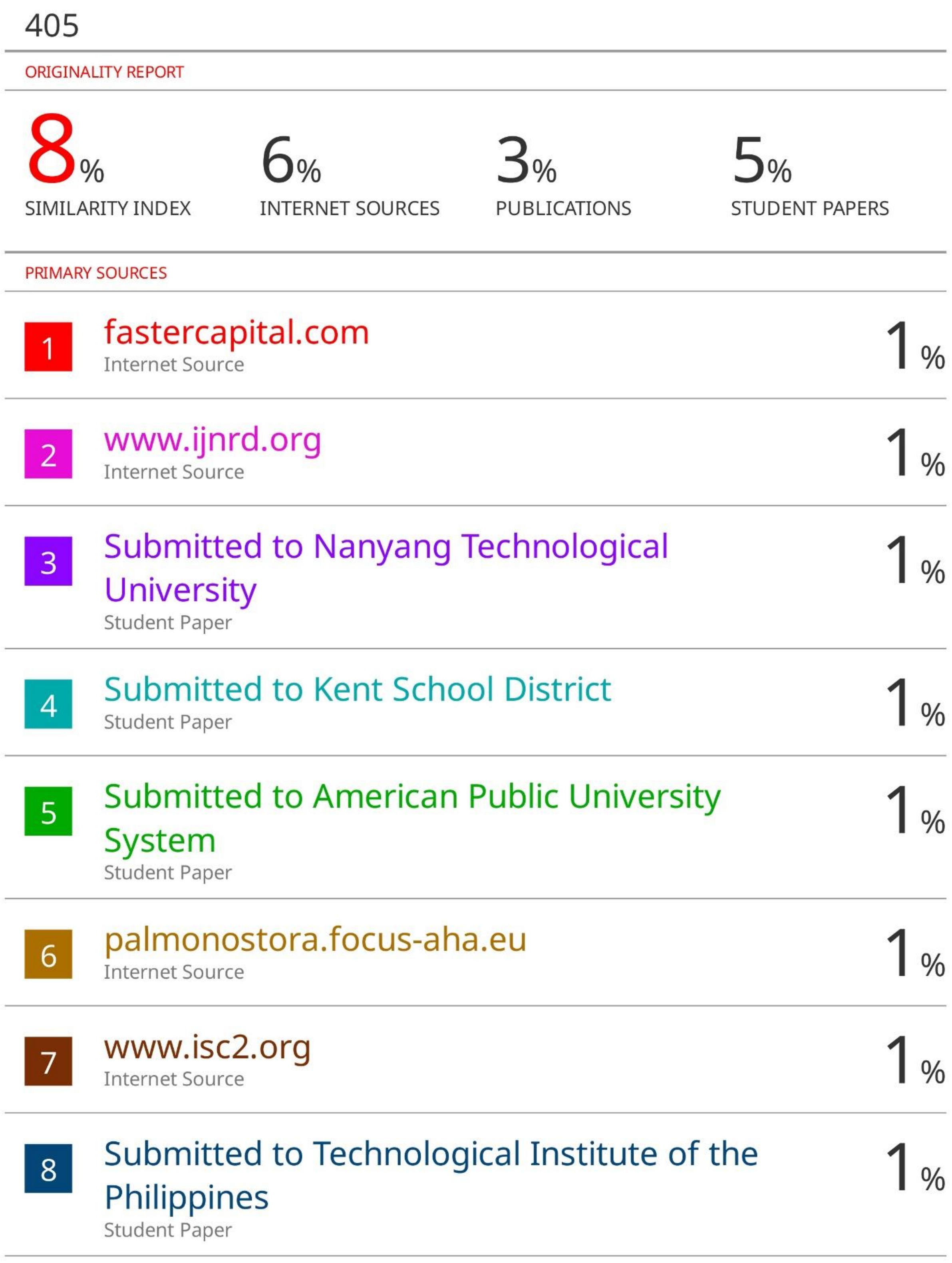
This website is safe to use :)

(-) Prefix/Suffix in domain | @ Symbol | Anchor | Favicon | HTTPS | HTTPS in URL's domain part | IP Address | No. of Sub Domains | Port | Redirecting using // | Request URL | SFH | Script & Link | Tiny URL | URL Length | iFrames | mailto

View model test results

Online Shopping site in India
amazon.in
amazon.in prime
Deliver to Emmanuel Chennai 600040
Search Amazon.in
All Fresh Keep Shopping for Buy Again Amazon miniTV Sell Gift Cards Coupons Amazon Pay Gift Ideas Browsing History
Up to 60% off
Deals on headphones, speakers & more
NOISE | BOULT | SONY
Up to 5% off for Prime members*
Pick up where you left off
Keep shopping for
Buy Again
PC game headsets
Food strainers
Wall shelves
Candles
Purchased Jan 2024
Purchased Nov 2023
Purchased Jul 2023
Orders
Cart
Books
28°C Partly sunny
08:41 22-03-2024

A.3 PLAGIARISM REPORT



405

ORIGINALITY REPORT

8%	6%	3%	5%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	fastercapital.com Internet Source	1%
2	www.ijnrd.org Internet Source	1%
3	Submitted to Nanyang Technological University Student Paper	1%
4	Submitted to Kent School District Student Paper	1%
5	Submitted to American Public University System Student Paper	1%
6	palmonostora.focus-aha.eu Internet Source	1%
7	www.isc2.org Internet Source	1%
8	Submitted to Technological Institute of the Philippines Student Paper	1%

9	Athirah Mohd Ramly, Nor Fadzilah Abdullah, Rosdiadee Nordin. "Cross-Layer Design and Performance Analysis for Ultra-Reliable Factory of the Future Based on 5G Mobile Networks", IEEE Access, 2021 Publication	<1 %
10	Submitted to University College Birmingham Student Paper	<1 %
11	thehackernews.com Internet Source	<1 %
12	Submitted to Sim University Student Paper	<1 %
13	www.csis.org Internet Source	<1 %
14	hbrppublication.com Internet Source	<1 %

Exclude quotes	On	Exclude matches	Off
Exclude bibliography	On		

A.4 PAPER PUBLICATION

Phish Detect: Building and Deploying a Phishing URL Detection System with Machine Learning

Sivakamy N[1], Pughazendi N[2], Arunkumar K. V[3], Emmanuel Joshua C[4], Dilip G[5]

Faculty[1,2], UG Scholar[3,4,5]
Department of Computer Science and Engineering, Panimalar Engineering College

sivakamyn@gmail.com[1], pughazendi@gmail.com[2], kvarunkumar23@gmail.com[3], joshjack2408@gmail.com[4], gdilip0803@gmail.com[5]

Abstract - **Phishing attacks have emerged as a pervasive and insidious cyber threat, exploiting the trust of unsuspecting users to steal sensitive information. These attacks utilize deceptive tactics, masquerading as legitimate entities such as banks or social media platforms, to deceive individuals into divulging confidential data or compromising their security. With methods continually evolving and growing in sophistication, phishing attacks have become harder to detect and mitigate. Attackers employ various techniques, including email spoofing, fake websites, social engineering, and malware distribution, to deceive users and gain unauthorized access to their accounts or sensitive data. The scale and frequency of phishing attacks have escalated, targeting individuals, businesses, and organizations across diverse sectors. The impact of these attacks is profound and far-reaching, resulting in significant financial losses, data breaches, and identity theft. Moreover, phishing attacks disrupt online services and operations, undermining the reliability and availability of digital platforms. Additionally, they erode trust in online communication and transactions, diminishing users' confidence in digital services. In response to these challenges, there is an urgent need for advanced solutions like Phish Detect. Phish Detect offers real-time analysis, seamless browser integration, adaptive detection mechanisms, and thorough scrutiny of sub links. These features empower users to promptly identify and address potential phishing threats, mitigating financial losses, data breaches, and identity theft. By bolstering online security, Phish Detect aims to restore trust in digital transactions and safeguard the integrity of the digital ecosystem.**

Keywords - Phish Detect, Online security, Phishing attacks, Real-time analysis, Browser integration, Sub links scrutiny, Adaptive detection mechanisms, Privacy compromise.

I. INTRODUCTION

In the rapidly evolving landscape of cybersecurity, one persistent and increasingly sophisticated threat stands out: phishing attacks. These deceptive manoeuvres, adept at exploiting human psychology and technological vulnerabilities, pose a significant risk to individuals, businesses, and the overall integrity of online ecosystems. Phishing attacks typically involve the impersonation of trusted entities through email, social media, or other communication channels, luring unsuspecting victims into divulging sensitive information such as passwords, financial details, or personal data. The repercussions of falling victim to phishing can be dire, ranging from financial losses and identity theft to reputational damage and operational disruptions. As the methods and techniques employed by cybercriminals continue to evolve, traditional security measures often struggle to keep pace with the dynamic nature of phishing attacks. Conventional antivirus software and spam filters, while valuable layers of defence, may prove insufficient in detecting and thwarting the ever-changing tactics of phishing perpetrators. In this context, the imperative for innovative and adaptive solutions to combat phishing has never been more pressing.

This research paper introduces "Phish Detect," a groundbreaking browser extension designed to tackle the multifaceted challenge of phishing attacks head-on. Phish Detect embodies a comprehensive approach to online security, harnessing real-time analysis, browser integration, adaptive detection mechanisms, sub link scrutiny, and user-friendly interfaces to fortify defences against phishing threats. By seamlessly integrating into popular web browsers such as Chrome, Firefox, and Safari, Phish Detect empowers users to navigate the digital landscape with confidence, offering timely alerts and insights to help discern between legitimate URLs and potential phishing scams. The objectives of this paper are twofold: firstly, to elucidate the escalating threat landscape of phishing attacks and the profound implications they entail for individuals, businesses, and society at large; and secondly, to elucidate the design, implementation, and evaluation of Phish Detect as a pioneering solution to mitigate the risks posed by phishing attacks. Through empirical analysis and user feedback, this paper aims to demonstrate the efficacy, reliability, and practical utility of Phish Detect in bolstering online security and safeguarding users against the pernicious threats posed by phishing attacks. Ultimately, the research presented herein underscores the critical importance of proactive and innovative approaches to cybersecurity in an era defined by ever-evolving digital threats.

The problem addressed by this research paper revolves around the inadequacies of current security measures in effectively detecting and mitigating phishing attacks. Traditional methods often rely on static rules or signatures, making them susceptible to evasion by attackers who frequently alter their strategies to evade detection. Moreover, as phishing attacks become more sophisticated and diversified, they increasingly target individuals across various online platforms, from email and social media to search engine results and online advertisements. The challenge lies in developing a robust and adaptive solution capable of providing real-time analysis and proactive protection against phishing threats across diverse online contexts. Such a solution must seamlessly integrate with users' existing workflows, offering intuitive interfaces and minimal disruptions while effectively identifying and alerting users to potential phishing attempts. Additionally, the solution must exhibit adaptive capabilities to stay ahead of evolving phishing tactics, continuously learning and updating its detection mechanisms to effectively counter emerging threats. Furthermore, the problem statement encompasses the need to address the growing complexity of phishing attacks, including the scrutiny of sub links embedded within online content. Many phishing attempts utilize sub links as vectors for deception, leading users to malicious websites or compromising their devices with malware. Therefore, any effective solution must thoroughly analyse not only main URLs but also sub links to provide comprehensive protection against phishing attempts. In summary, the problem statement centers on the urgency of developing advanced solutions, such as Phish Detect, capable of addressing the dynamic and multifaceted nature of phishing attacks. These solutions must offer real-time analysis, seamless integration, adaptive detection mechanisms, and thorough scrutiny of sub links to effectively safeguard users and mitigate the risks posed by phishing in an increasingly interconnected digital landscape.

II. RELATED WORK

The deep learning-based framework for phishing website detection introduced by [1], leverages neural networks to analyse website features effectively. Their approach offers advantages in capturing complex patterns in website features, resulting in high accuracy in identifying phishing websites. However, it requires large amounts of labelled data for training deep learning models, and the computational intensity, especially during training, can be a drawback.

A comprehensive survey conducted by [2], on phishing website detection, provides insights into various techniques and challenges in this area. Their review encompasses machine learning, similarity-based methods, and behavioural analysis, offering a valuable resource for researchers and practitioners. Nevertheless, due to the breadth of topics covered, the survey may lack in-depth analysis of individual techniques.

The phishing detection system proposed by [3] is based on page component similarity, achieving high accuracy while maintaining computational efficiency. Their approach is advantageous for its simplicity and effectiveness in detecting phishing websites. However, it may be less effective against sophisticated phishing attacks that employ obfuscation techniques.

A real-case scenario for phishing URL detection presented by [4], focusing on login URLs and employing machine learning techniques. Their study offers practical insights into targeted detection mechanisms but is limited to specific threat vectors and relies on accurate feature extraction from URLs, which can be challenging.

Phishing website detection by analysing URL and domain name features, investigated by [5], achieves high accuracy through machine learning methods. Their approach provides valuable insights for detection but may be susceptible to evasion techniques involving slight modifications to URLs and domain names.

The state of the art and future challenges in combating phishing attacks that has been reviewed by [6], offers a comprehensive overview of various techniques. Their review highlights future research directions but may lack in-depth analysis of individual techniques and could include outdated information.

A survey conducted by [7] on phishing environments, techniques, and countermeasures, providing insights into attacker tactics and security professional responses. Their study helps in understanding the dynamic nature of phishing environments but may lack specific details on technical solutions for detection and could contain outdated information.

III. SYSTEM ARCHITECTURE

The architectural model of "Phish Detect" combines machine learning techniques with client-side processing to provide a robust and real-time solution for detecting and mitigating phishing attacks. By leveraging browser scripts and offline feature extraction, the system offers users a proactive defence against the growing threat of phishing websites.

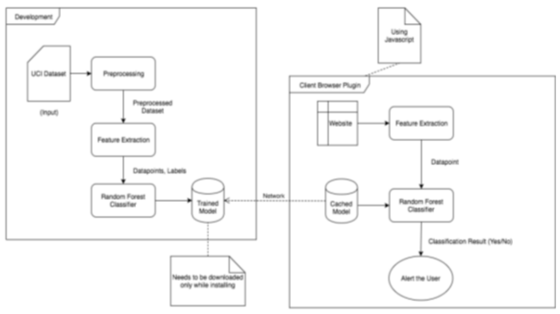


Fig 1 Architectural Model

A. Random Forest Classifier Training

The system begins by training a Random Forest classifier using a dataset of known phishing sites. This classifier is built using Python's scikit-learn library, a popular machine learning toolkit. The dataset consists of features extracted from URLs or website content that help differentiate between phishing and legitimate websites. These features might include IP address, degree of subdomain, anchor tag href domains, URL length, HTTPS usage, script & link tag domains, URL shortener, favicon domain, empty server form handler, '@' in URL, TCP port, use of mail to, redirect with '//', HTTPS in domain name, use of iFrame,'-' in domain, cross-domain requests. The Random Forest algorithm is chosen for its ability to handle large datasets and its robustness against overfitting.

B. JSON Representation of Classifier

Once the Random Forest classifier is trained, it is represented in JSON format. This representation contains the learned model parameters, such as decision trees, feature importance scores, and other metadata required to make predictions. JSON is chosen for its simplicity and ease of integration into various systems.

C. Browser Script Implementation

A browser script is developed to leverage the trained Random Forest classifier for real-time phishing detection. This script is typically implemented as a browser extension or plugin, compatible with popular browsers like Chrome or Firefox. The script's main function is to intercept website requests and classify them as phishing or legitimate based on the exported classifier model. It communicates with the JSON representation of the classifier to make predictions on the fly.

D. Phishing Detection Mechanism

The core functionality of the system lies in its ability to detect phishing websites as users browse the internet. When a user visits a website, the browser plugin executes a script that extracts relevant features from the site, such as its URL structure, content, SSL certificate details, etc. These features are then encoded and passed through the trained Random Forest classifier stored in JSON format. The classifier predicts whether the website is phishing or legitimate based on these features, and the user is alerted accordingly.

E. Offline Feature Extraction

To ensure independence from external services and preserve user privacy, the system performs feature extraction entirely offline on the client side. This means that all necessary information for phishing detection is extracted locally within the user's browser, without relying on external APIs or services. By extracting features offline, the system minimizes latency and reduces the risk of data leakage.

F. Testing and Deployment

Before deployment, the system undergoes rigorous testing to ensure its effectiveness in detecting phishing websites accurately and efficiently. Testing involves evaluating the system's performance on various datasets, including known phishing URLs and legitimate websites. Additionally, the system is designed to adapt to evolving threats by continuously updating its classifier with new phishing data from sources like PhishTank. The JSON representation of the classifier is hosted on a URL accessible to the browser plugin, ensuring seamless integration and easy updates.

IV. IMPLEMENTATION ENVIRONMENT

The implementation environment for Phish Detect, as a browser extension designed to provide real-time phishing detection, would involve several components and technologies.

A. Programming Languages:

1. Python: Utilized for training the Random Forest classifier on a dataset of phishing websites using the scikit-learn library. Python is chosen for its robust machine learning capabilities and ease of integration with data processing tasks.

2. JavaScript: Translated the Python-based phishing detection algorithms, including the Random Forest classifier, into JavaScript for client-side operation within the browser environment. JavaScript enables real-time website classification during browsing sessions.

B. Tools and Libraries:

1. Scikit-learn: A popular Python library for machine learning tasks, used for training the Random Forest classifier on the phishing dataset. It provides efficient implementations of various machine learning algorithms.

2. Browser Plugin: Developed a browser plugin to enable real-time classification of websites as users browse, contributing to better privacy and faster phishing detection. The plugin integrates the JavaScript-based detection algorithms seamlessly into the browsing experience.

3. JSON Format: Used to represent the trained Random Forest classifier in a structured format, facilitating its integration into the browser plugin for efficient classification of web pages.

4. Chrome Extension: Implemented client-side scripts within a Chrome extension to extract and encode selected features from web pages for phishing detection. This extension enhances the user's browsing security by providing timely warnings about potential phishing websites.

C. Hardware Requirement:

No special hardware interface is required for the successful implementation of the system, ensuring compatibility with standard computing devices without the need for additional resources.

By leveraging a combination of Python for training the machine learning model and JavaScript for client-side implementation within the browser environment, the "Phish Detect" system offers a comprehensive and effective solution for detecting phishing websites in real-time, enhancing user privacy, and providing rapid protection against phishing attacks

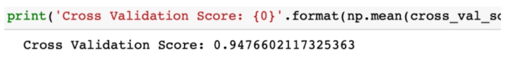
V. PERFORMANCE EVALUATION

The performance of the entire system is evaluated using the standard parameters described below.

A. Cross Validation score

Learning the parameters of a prediction function and testing it on the same data is a methodological mistake: a model that would just repeat the labels of the samples that it has just seen would have a perfect score but would fail to predict anything useful on yet-unseen data. This situation is called overfitting. To solve this problem, yet another part of the dataset can be held out as a so-called "validation set": training proceeds on the training set, after which evaluation is done on the validation set, and when the experiment seems to be successful, final evaluation can be done on the test set. However, by partitioning the available data into three sets, we drastically reduce the number of samples which can be used for learning the model, and the results can depend on a particular random choice for the pair of (train, validation) sets. A solution to this problem is a procedure called cross-validation (CV for short). A test set should still be held out for final evaluation, but the validation set is no longer needed when doing CV. In the basic approach, called k-fold CV, the training set is split into k smaller sets (other approaches are described below, but generally follow the same principles). The following procedure is followed for each of the k "folds": A model is trained using k of the folds as training data; the resulting model is validated on the remaining part of the data (i.e., it is used as a test set to compute a performance measure such as accuracy).

The score on 10-Fold cross validation is as below



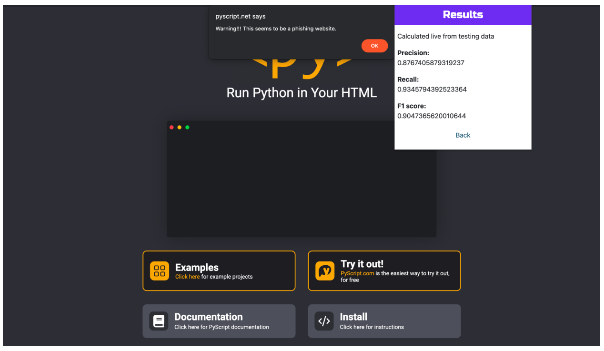
```
print('Cross Validation Score: {0}'.format(np.mean(cross_val_sc
```

```
 Cross Validation Score: 0.9476602117325363
```

Fig 2 Cross Validation Score

B. F1 score

F1 score is a measure of a test's accuracy. It considers both the precision and the recall of the test to compute the score: precision is the number of correct positive results divided by the number of all positive results returned by the classifier, and recall is the number of correct positive results divided by the number of all relevant samples (all samples that should have been identified as positive). The F1 score is the Harmon- 31 is average of the precision and recall, where an F1 score reaches its best value at 1 (perfect precision and recall) and worst at 0. The F1 score can be interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. The relative contribution of precision and recall to the F1 score are equal. The formula for the F1 score is: F1 = 2 * (precision * recall) / (precision + recall) The precision, recall and F1 score of the phishing classifier is calculated manually using JavaScript on the test data set.



pyscript.net says

Warning!!! This seems to be a phishing website.

Results

Calculated live from testing data

Precision:
0.8767405879319237

Recall:
0.9345794392523364

F1 score:
0.9047365620010644

OK
Run Python in Your HTML
Back
Examples
Click here for example projects
Try it out!
PyScript.com is the easiest way to try it out, for free
Documentation
Click here for PyScript documentation
Install
Click here for instructions

Fig 3 Test Accuracy result

VI. CONCLUSION

The utilisation of a machine learning algorithm, specifically the Random Forest classifier, in the "Phish Detect" system demonstrates promising results in the rapid detection of phishing URLs. By porting the classifier to JavaScript and focusing on client-side implementation, the system provides timely warnings to users before falling victim to phishing attacks. Despite a slight decrease in accuracy due to using a reduced set of features, the system's emphasis on rapid detection and enhanced privacy enhances its usability. The F1 score of 0.886 on the test set indicates the effectiveness of the machine learning technique in distinguishing between legitimate and phishing URLs.

Furthermore, the successful implementation of the Random Forest classifier in JavaScript showcases the potential for efficient phishing detection without relying heavily on network-based features. This approach not only improves detection speed but also enhances user privacy. Future enhancements could involve expanding the feature set without compromising detection speed, implementing result caching for frequently visited sites, and exploring the use of Worker Threads for faster classification.

VII. REFERENCE

[8] L. Tang and Q. H. Mahmoud, "A Deep Learning-Based Framework for Phishing Website Detection," in IEEE Access, vol. 10, pp. 1509-1521, 2022, 23 December 2021, doi: 10.1109/ACCESS.2021.3137636.

[9] R. Zieni, L. Massari and M. C. Calzarossa, "Phishing or Not Phishing? A Survey on the Detection of Phishing Websites," in IEEE Access, vol. 11, 22 February 2023, doi: 10.1109/ACCESS.2023.3247135.

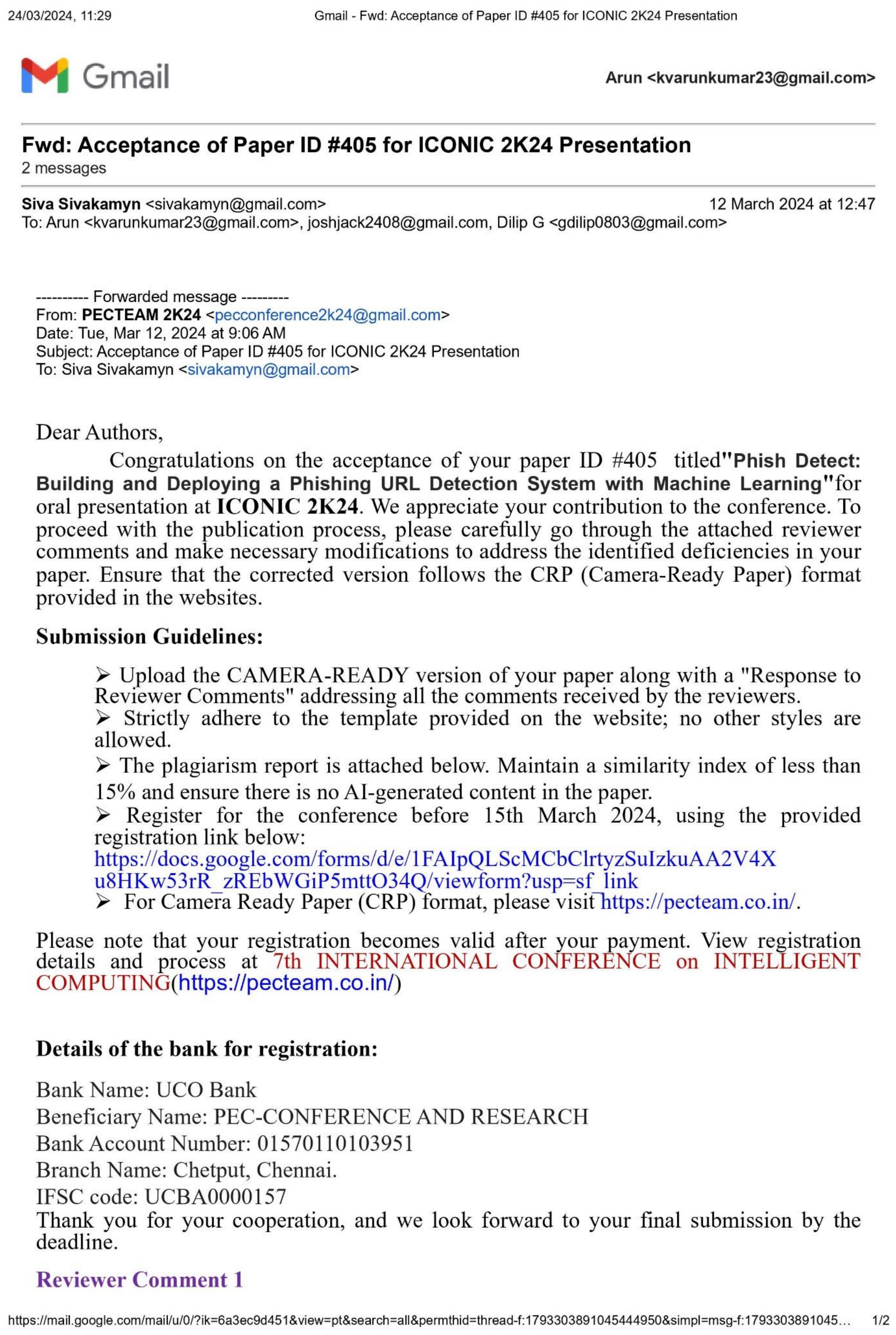
[10] J. Mao, W. Tian, P. Li, T. Wei and Z. Liang, "Phishing-Alarm: Robust and Efficient Phishing Detection via Page Component Similarity," in IEEE Access, vol. 5, 23 August 2017, doi: 10.1109/ACCESS.2017.2743528.

[11] M. Sánchez-Paniagua, E. F. Fernández, E. Alegre, W. Al-Nabki and V. González-Castro, "Phishing URL Detection: A Real-Case Scenario Through Login URLs," in IEEE Access, vol. 10, 18 April 2022, doi: 10.1109/ACCESS.2022.3168681.

[12] Kara, M. Ok and A. Ozaday, "Characteristics of Understanding URLs and Domain Names Features: The Detection of Phishing Websites with Machine Learning Methods," in IEEE Access, vol. 10, 17 November 2022, doi: 10.1109/ACCESS.2022.3223111.

[13] Gupta, B.B., Tewari, A., Jain, A.K. “Fighting against phishing attacks: state of the art and future challenges” December 2017, https://doi.org/10.1007/s00521-016-2275-y

[14] Ahmed Aleroud, Lina Zhou “Phishing environments, techniques, and countermeasures: A survey”, Volume 68, 17 April 2017, https://doi.org/10.1016/j.cose.2017.04.006.



24/03/2024, 11:29 Gmail - Fwd: Acceptance of Paper ID #405 for ICONIC 2K24 Presentation

Arun <kvarunkumar23@gmail.com>

Fwd: Acceptance of Paper ID #405 for ICONIC 2K24 Presentation
2 messages

Siva Sivakamyn <sivakamyn@gmail.com> 12 March 2024 at 12:47
To: Arun <kvarunkumar23@gmail.com>, joshjack2408@gmail.com, Dilip G <gdilip0803@gmail.com>

---------- Forwarded message ---------
From: **PECTEAM 2K24** <pecconference2k24@gmail.com>
Date: Tue, Mar 12, 2024 at 9:06 AM
Subject: Acceptance of Paper ID #405 for ICONIC 2K24 Presentation
To: Siva Sivakamyn <sivakamyn@gmail.com>

Dear Authors,

Congratulations on the acceptance of your paper ID #405 titled"**Phish Detect: Building and Deploying a Phishing URL Detection System with Machine Learning**"for oral presentation at **ICONIC 2K24**. We appreciate your contribution to the conference. To proceed with the publication process, please carefully go through the attached reviewer comments and make necessary modifications to address the identified deficiencies in your paper. Ensure that the corrected version follows the CRP (Camera-Ready Paper) format provided in the websites.

Submission Guidelines:

- Upload the CAMERA-READY version of your paper along with a "Response to Reviewer Comments" addressing all the comments received by the reviewers.
- Strictly adhere to the template provided on the website; no other styles are allowed.
- The plagiarism report is attached below. Maintain a similarity index of less than 15% and ensure there is no AI-generated content in the paper.
- Register for the conference before 15th March 2024, using the provided registration link below: https://docs.google.com/forms/d/e/1FAIpQLScMCbClrtyzSuIzkuAA2V4Xu8HKw53rR_zREbWGiP5mttO34Q/viewform?usp=sf_link
- For Camera Ready Paper (CRP) format, please visit https://pecteam.co.in/.

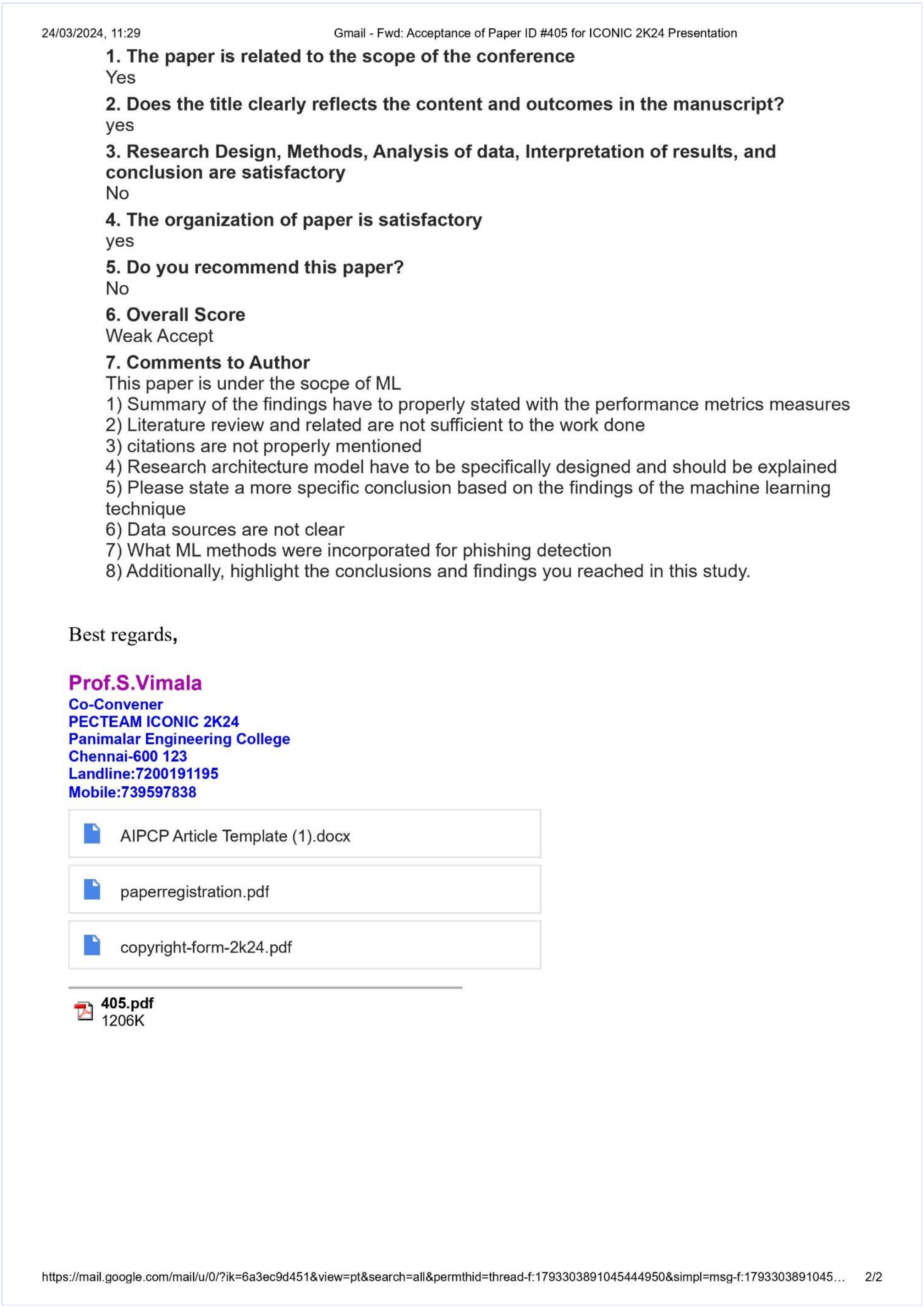
Please note that your registration becomes valid after your payment. View registration details and process at 7th INTERNATIONAL CONFERENCE on INTELLIGENT COMPUTING(https://pecteam.co.in/)

Details of the bank for registration:

Bank Name: UCO Bank
Beneficiary Name: PEC-CONFERENCE AND RESEARCH
Bank Account Number: 01570110103951
Branch Name: Chetput, Chennai.
IFSC code: UCBA0000157
Thank you for your cooperation, and we look forward to your final submission by the deadline.

Reviewer Comment 1

https://mail.google.com/mail/u/0/?ik=6a3ec9d451&view=pt&search=all&permthid=thread-f:1793303891045444950&simpl=msg-f:1793303891045… 1/2



24/03/2024, 11:29 Gmail - Fwd: Acceptance of Paper ID #405 for ICONIC 2K24 Presentation

1. The paper is related to the scope of the conference
Yes

2. Does the title clearly reflects the content and outcomes in the manuscript?
yes

3. Research Design, Methods, Analysis of data, Interpretation of results, and conclusion are satisfactory
No

4. The organization of paper is satisfactory
yes

5. Do you recommend this paper?
No

6. Overall Score
Weak Accept

7. Comments to Author
This paper is under the socpe of ML
1) Summary of the findings have to properly stated with the performance metrics measures
2) Literature review and related are not sufficient to the work done
3) citations are not properly mentioned
4) Research architecture model have to be specifically designed and should be explained
5) Please state a more specific conclusion based on the findings of the machine learning technique
6) Data sources are not clear
7) What ML methods were incorporated for phishing detection
8) Additionally, highlight the conclusions and findings you reached in this study.

Best regards,

Prof.S.Vimala
Co-Convener
PECTEAM ICONIC 2K24
Panimalar Engineering College
Chennai-600 123
Landline:7200191195
Mobile:739597838

AIPCP Article Template (1).docx

paperregistration.pdf

copyright-form-2k24.pdf

405.pdf
1206K

https://mail.google.com/mail/u/0/?ik=6a3ec9d451&view=pt&search=all&permthid=thread-f:1793303891045444950&simpl=msg-f:1793303891045… 2/2