"DETECTION OF PHISHING WEBSITES"

MINOR PROJECT: CS-D-03

MENTOR –
PROFESSOR SARSIJ TRIPATHI

MEMBERS –
SHASHWAT KHARE - 20214142
SOURAV PAUL - 20214056
SAYON KAR - 20214533
VAISHNAV ANISH – 20214501

INTRODUCTION

- **PROBLEM STATEMENT**

  The aim of this project is to create a system to identify phishing URLs efficiently. Using machine learning and URL pattern analysis the system should accurately detect phishing URLs to protect users from scams and data theft. This solution will also be integrated into web application and web browser extension to prevent phishing-related risks.

- **MOTIVATION**

  Phishing attacks cause billions in financial losses and widespread identity theft by tricking users into revealing sensitive information. As cybercriminals refine their tactics, detecting phishing URLs is increasingly challenging but crucial for protecting individuals, businesses, and public safety.

  Effective phishing URL detection can prevent unauthorized access, data breaches, and financial fraud. By combating deceptive URLs through advanced analysis and pattern recognition, this project addresses real-world risks, supports compliance with cybersecurity regulations, and helps maintain trust in digital services.

INTRODUCTION - contd.

- **CYBERCRIME ATTACKS**

  Phishing - Phishing involves sending deceptive emails or messages to trick recipients into revealing sensitive information or clicking on malicious links.

  Ransomware - Ransomware is a type of malware that encrypts a victim's data, rendering it inaccessible.

  Denial of Service (DoS) and Distributed Denial of Service (DDoS) - A DoS attack overwhelms a system or network with excessive traffic or requests, rendering it unavailable to legitimate users.

  SQL Injection - SQL injection attacks target web applications with poorly secured databases. Attackers insert malicious SQL code into database queries, allowing them to access or manipulate data, or even gain administrative privileges.

  Man-in-the-Middle (MitM) Attack - In a MitM attack, an attacker intercepts and potentially alters communication between two parties without their knowledge.

  Credential Stuffing - Credential stuffing involves using lists of leaked usernames and passwords to gain unauthorized access to multiple accounts.

  Brute Force Attacks - Brute force attacks systematically try all possible combinations of passwords or encryption keys until the correct one is found.

  Malware - Malware encompasses a wide range of malicious software, including viruses, worms, Trojans, spyware, and more.

INTRODUCTION - contd.

- **PHISHING URLS**

  Phishing URLs are malicious web links designed to deceive users into revealing sensitive information, such as login credentials, financial data, or personal details.

  These URLs often mimic legitimate websites, using similar domain names or disguised links to trick users into believing they are interacting with a trusted source.

- **CHARACTERISTICS**

  Spoofed Domains: Phishing URLs often use misspelled or similar-looking domain names to mimic trusted websites.

  Obfuscated Links: Phishing URLs may use shortened URLs or complex URL structures to hide their true destination.

  Fraudulent Landing Pages: The websites these URLs lead to may look nearly identical to legitimate ones but are designed to steal information.

INTRODUCTION - contd.

- **DETECTING PHISHING URLs**

  Anti-Phishing Software: Use security tools that automatically scan websites for suspicious links.

  Education and Awareness: Teach users to recognize the signs of phishing, such as unexpected emails, urgent requests, or suspicious URLs.

  Regular Security Audits: Conduct regular security audits and penetration testing to identify vulnerabilities and improve defenses.

- **IMPORTANCE OF DETECTING PHISHING URLs**

  Prevent Data Theft: Detecting phishing URLs helps prevent unauthorized access to sensitive information, such as usernames, passwords, and credit card numbers.

  Avoid Financial Loss: Phishing attacks can lead to identity theft, unauthorized transactions, and significant financial damage.

  Protect Personal and Corporate Reputation: Falling victim to phishing can harm personal reputation or damage corporate brands.

  Maintain Cybersecurity: Early detection of phishing URLs is essential for maintaining overall cybersecurity, protecting networks, and preventing malware distribution.

INTRODUCTION - contd.

- **MACHINE LEARNING ALGORITHMS**

  Machine Learning (ML) algorithms play a pivotal role in detecting phishing URLs, helping to combat the rising threat of cyberattacks that aim to deceive users into providing sensitive information.

  ML algorithms for phishing detection are typically trained on large datasets containing both phishing and legitimate URLs. The training process involves learning patterns that distinguish phishing URLs from safe ones.

  Common ML Algorithms for Phishing Detection –

  Logistic Regression

  Decision Trees

  Random Forests

  Gradient Boosting Machines (GBM)

  Support Vector Machines (SVM)

RELATED WORK

- Yang et al. (2019) developed a method called Multidimensional Feature Phishing Detection (MFPD) using deep learning. They extracted character sequence features from URLs for classification, incorporating a dynamic category choice method to speed up detection. By combining URL statistics, webpage code, and text characteristics with deep learning results, they verified detection accuracy. Testing on a dataset with millions of URLs yielded a 98.99% accuracy with a low false positive rate of 0.59%, demonstrating strong performance in accuracy, false positives, and speed.

Link- (PDF) Phishing Website Detection Based on Multidimensional Features Driven by Deep Learning (researchgate.net)

RELATED WORK - contd.

- In 2020, Ali and Maleberry introduced a method to enhance phishing website identification accuracy using particle swarm optimization (PSO). By selecting PSO-based website features, they emphasized the importance of each feature in distinguishing phishing from legitimate sites. Applying PSO reduced unnecessary characteristics in training datasets by 7-57%. They employed various learning methods including back-propagation neural network (BPNN), support vector machine (SVM), k-nearest neighbor (KNN), decision tree (C4.5), random forest (RF), and naive Bayes classifier (NB). Compared to standalone models, PSO-based algorithms showed superior accuracy and improvements in true positive rate (TPR), true negative rate (TNR), false positive rate (FPR), and false negative rate (FNR) by utilizing a reduced set of website attributes.

Link- https://ieeexplore.ieee.org/document/9799917

RELATED WORK - contd.

- In 2021, Tang and Mahmoud introduced a deep learning-based framework for phishing website identification, implemented as a Google Chrome extension. This browser plug-in delivers real-time risk assessments and prompts warnings upon phishing detection. Their approach integrates whitelist filtering, blacklist interception, and machine learning (ML) prediction to enhance accuracy, minimize false positives (FP), and accelerate processing time. Utilizing seven datasets sourced from four distinct repositories, they employed Support Vector Machine (SVM), Logistic Regression, Random Forest, and RNN-GRU models. RNN-GRU demonstrated the highest accuracy at 99.18%, establishing it as the most effective model according to their findings.

Link- A Deep Learning-Based Framework for Phishing Website Detection | IEEE Journals & Magazine | IEEE Xplore

METHODOLOGY

- **Feature Engineering**

  get_domain_length(url):

  This function takes a single string argument url representing a URL. It imports the urlparse function from the urllib.parse module to parse the URL into its components. It extracts the domain name from the parsed URL using the netloc attribute, which contains the domain and port information. The function returns the calculated length of the domain name.

  special_char_ratio_in_url(url):

  This function calculates the ratio of special characters(excluding letters, digits, and common punctuation symbols like periods, commas, and hyphens) within a URL and stores the result in a new column of our DataFrame.

  is_domain_ip(url):

  This function effectively identifies whether the domain part of a URL is an IP address and stores the result in a new column of our DataFrame. If the conversion is successful (IP address), the function returns 1. If the conversion fails (not an IP address), the function returns 0.

METHODOLOGY - contd.

tld_length(tld):

It effectively extracts the top-level domain (TLD) from URLs, calculates its length, and stores both the TLD and its length in our DataFrame. This function takes a string argument tld (top-level domain). It checks if the tld is not empty. If it's not empty, it returns the length of the tld string. If it's empty, it returns -1.

url_character_prob(url):

It effectively calculates the average probability of each character appearing in a URL and stores it in a new column of our DataFrame. This feature provides additional information about the character composition of URLs, potentially aiding in further analysis and insights. It Returns the calculated average character probability.

number_of_subdomains(url):

The function takes a single string argument url representing a URL. It uses urlparse(url) to parse the URL into its components. It extracts the domain name from the netloc attribute of the parsed URL. It basically counts the number of occurrences of the dot character ('.') in the extracted domain name. Each dot separates a subdomain from the next, so the count of dots indicates the number of subdomains. It returns the calculated number of subdomains as an integer.

If the domain name is empty (no dots), the function returns 0.

METHODOLOGY - contd.

obfuscation_ratio(url):

It effectively calculates the ratio of characters within a URL that belong to common obfuscation patterns and stores the result in a new column of our DataFrame. It returns the calculated obfuscation_ratio, which is calculated by dividing the total number of obfuscated characters (num_obfuscated_chars) by the total number of characters in the URL (total_chars).

letter_ratio_in_url(url):

It effectively calculates the letter ratio in a URL and it is calculated as If total_chars is greater than 0, then the letter_ratio is equal to the the number of letters (num_letters) divided by the total number of characters (total_chars).

fd_length(url):

It calculates the length of the first directory level within the path component of a URL and stores the result in a new column of our DataFrame. It splits the path by forward slashes (/) using the split('/') method. It attempts to access the second element (index 1) of the split list, which should represent the first directory level after the leading slash. If successful, the function returns the length of the first directory level string.

METHODOLOGY - contd.

calculate_tld_legitimate_prop(url):

It calculates a score representing the legitimacy of a URL's Top-Level Domain (TLD) and stores the result in a new column of our DataFrame. If the extracted TLD is present in the legitimate_tlds list, the function returns 1.0, indicating a high probability of the TLD being legitimate. If the TLD is not found in the list, the function returns 0.0, suggesting a lower probability of legitimacy.

having_ip_address(url):

The function employs regular expressions to search for IP addresses in various formats including IPv4 and IPv6.

abnormal_url(url):

It checks if a URL contains its own hostname within the path and stores the result in a new column of our DataFrame.It uses the re.search(hostname, url) function to search for the extracted hostname as a substring within the entire URL. If the regular expression search finds a match (i.e., the hostname is present within the URL path), the function returns 1, indicating a potential abnormality. If no match is found, the function returns 0.

no_of_dir(url):

It calculates the number of directory levels (indicated by forward slashes) within the path component of a URL and stores the result in a new column of our DataFrame. It returns the total number of directory levels found in the URL.

METHODOLOGY - contd.

no_of_embed(url):

It calculates the number of occurrences of the substring "//" within the path component of a URL and stores the result in a new column of our DataFrame. It uses the urldir.count('//') method to count the number of occurrences of the substring "//" within the extracted path. It returns the total number of occurrences of "//" found in the URL path.

suspicious_words(url):

It checks for the presence of certain suspicious words within a URL and stores a binary indicator (1 or 0) in a new column of our DataFrame. It uses the re.search function with a regular expression pattern to search for the presence of specific words within the URL string.

google_index(url):

It utilizes the googlesearch library to search for the given URL using Google Search.

It retrieves the top 5 results returned by Google Search. If the URL appears in any of the top 5 search results, the function returns 1, indicating that the URL is likely indexed by Google.

Otherwise, it returns 0.

METHODOLOGY - contd.

- **How Random Forest Classifier Work**

  Ensemble of Decision Trees: A random forest consists of a collection ("forest") of individual decision trees. Each tree is constructed using a subset of the training data and different subsets of features. This process, called bootstrapping or bagging, introduces randomness and helps diversify the ensemble.

  Training: To build the forest, the algorithm creates multiple decision trees by selecting random samples from the dataset and choosing a subset of features at each split. This randomness helps ensure that each tree is different, reducing the risk of overfitting.

  Classification: In the classification phase, each tree in the forest predicts an outcome, and the final prediction is typically determined by a majority vote among the trees. This aggregation helps improve accuracy and robustness.

  Benefits: Random forests offer several advantages:

  Robustness: By using multiple trees, the classifier is less prone to errors caused by outliers or overfitting.

  Versatility: It can handle both numerical and categorical data.

  Feature Importance: The algorithm can rank features based on their importance, aiding in feature selection and interpretation.

METHODOLOGY - contd.

- **Why Random Forest is Important for Phishing URL Detection**

  Handling Complexity: Phishing URL detection involves complex patterns and features like URL structure, domain name, presence of special characters, length, etc. Random forests can effectively manage this complexity due to their ensemble nature and flexibility with various types of data.

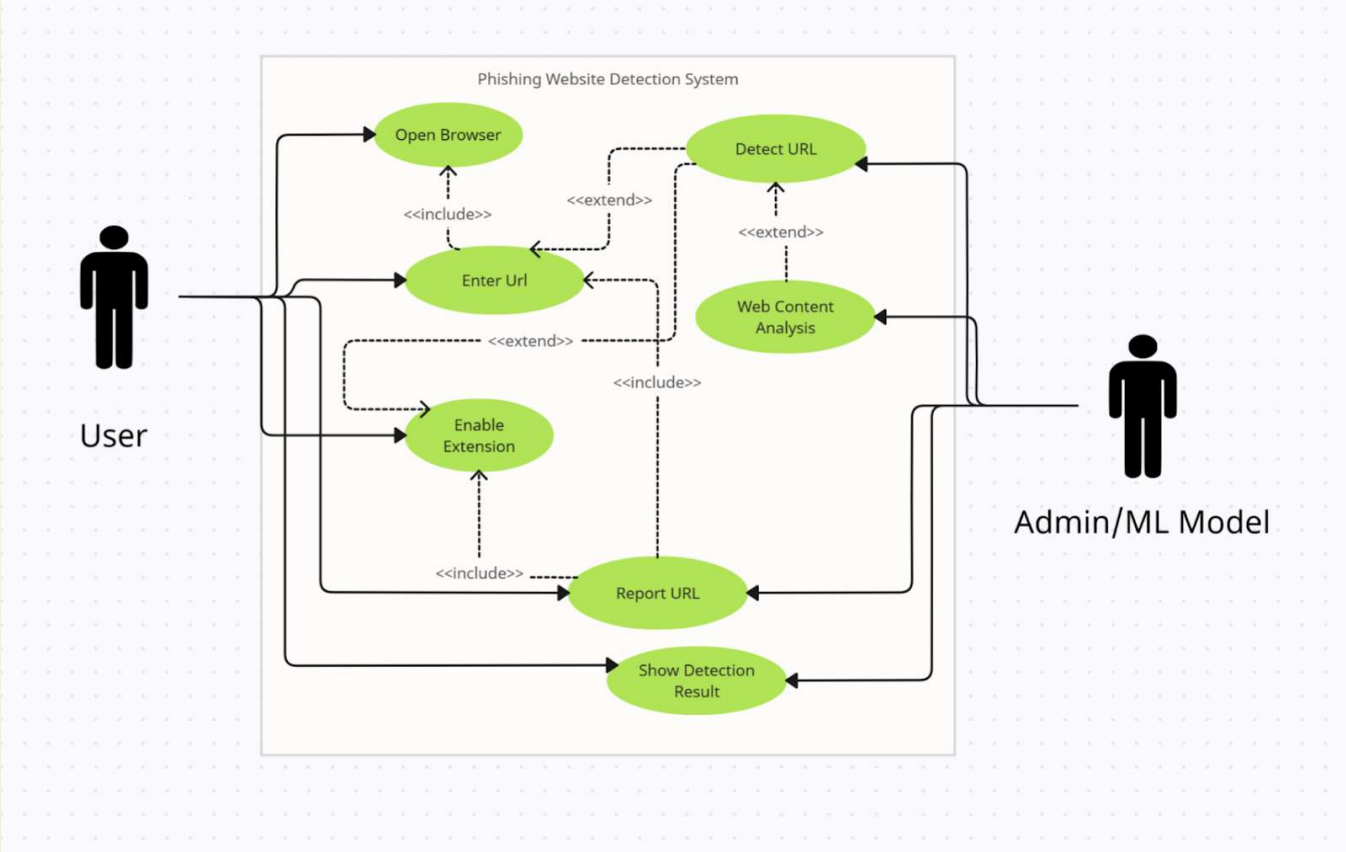
  Resilience to Noise: Phishing URLs can be diverse and sometimes mimic legitimate patterns to evade detection. The randomization in random forests helps make the classifier resilient to noise and variation in the data.

  Accuracy: Since random forests use a majority voting system across multiple decision trees, the resulting classifier tends to have high accuracy. This accuracy is crucial in phishing URL detection, where a wrong prediction can lead to security risks.

  Feature Importance and Interpretability: Phishing detection systems often need to explain why a URL is considered suspicious. Random forests can provide insights into which features contribute most to the prediction, aiding in interpretability and understanding of phishing tactics.

  Scalability: Given the vast number of URLs in use on the internet, scalability is important. Random forests can efficiently handle large datasets, making them suitable for real-time phishing detection systems.

USE CASE DIAGRAM



EXPERIMENTAL SETUP

- **PROTOCOL STACK**

Layer 1: Data Collection

Data Sources: Gather datasets containing URLs labeled as phishing or benign. These datasets can come from open-source repositories, security organizations, or data collection websites.

Open-Source Repository -

https://github.com/ESDAUNG/PhishDataset?tab=readme-ov-file

Security Organizations -

https://phishtank.org/

Data Collection Websites –

https://www.kaggle.com/

https://archive.ics.uci.edu/

Data Extraction: Extract URLs from various sources, including email archives, web scraping, or public datasets.

EXPERIMENTAL SETUP - contd.

Layer 2: Data Preprocessing

Feature Engineering: Extract meaningful features from URLs

Data Cleaning: Remove duplicates, normalize data, and handle missing values.

Labeling: Ensure that the dataset has appropriate labels indicating whether a URL is phishing or legitimate.

Layer 3: Machine Learning Model Development

Feature Selection: Selecting the most relevant features for training the model –

'url_length','domain_length', 'is_domain_ip', 'tld_length', 'char_continuation_rate', 'URLCharProb', 'NoOfSubDomain', 'HasObfuscation', 'NoOfObfuscatedChar', 'ObfuscationRatio', 'NoOfLettersInURL', 'LetterRatioInURL', 'NoOfDigitsInURL', 'DigitRatioInURL', 'NoOfAmpersandInURL', 'NoOfOtherSpecialCharsInURL', 'SpecialCharRatioInURL', 'IsHTTPS', 'TLDLegitimateProp', 'use_of_ip','abnormal_url', 'count%', 'count?', 'count-', 'count=', 'count-www','count@', 'count_dir', 'count_embed_domain', 'count-https', 'count.','count-http', 'short_url', 'hostname_length', 'sus_url', 'count-digits','count-letters', 'google_index', 'fd_length'

EXPERIMENTAL SETUP - contd.

Model Selection: Random Forest Classifier

Model Training:

Splitting the dataset to training and testing data -

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, stratify=Y, shuffle=True, random_state=5)

Model training (on training data) –

rf = RandomForestClassifier(n_estimators=100,max_features='sqrt')

rf.fit(X_train,Y_train)

Prediction(on testing data) –

Y_pred_rf = rf.predict(X_test)

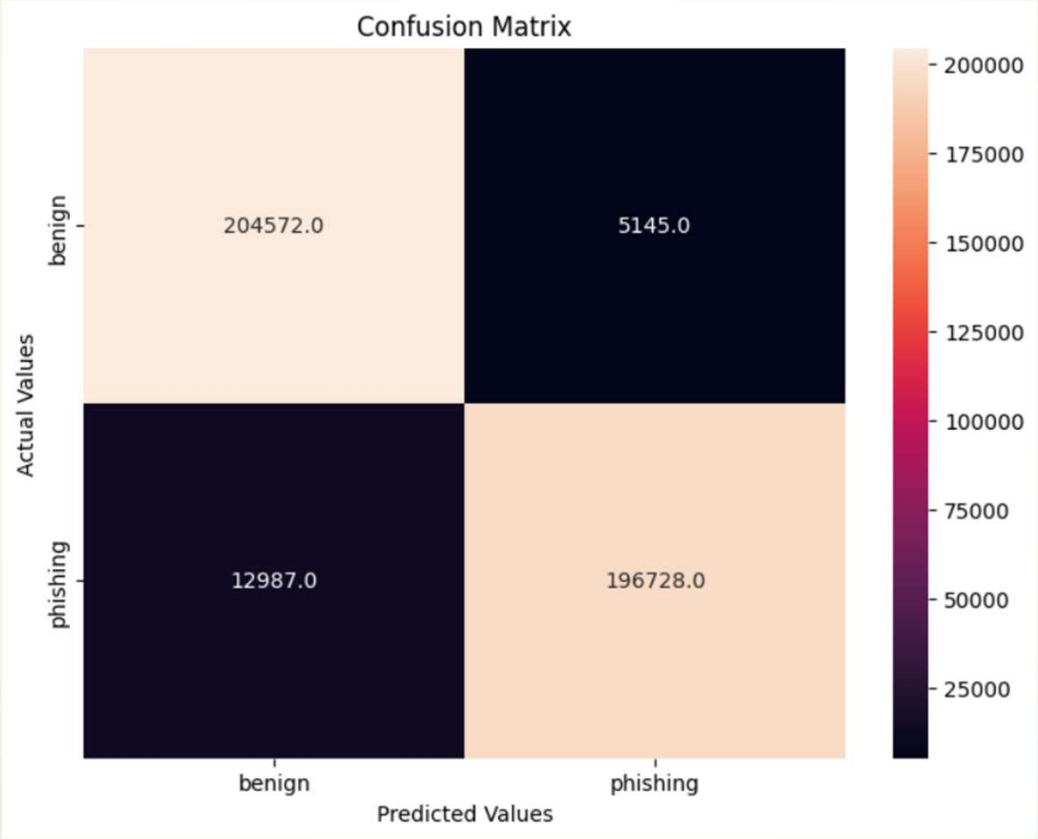
Model Evaluation:

Accuracy score of the training data: 0.9915707382667144

Accuracy score of the testing data: 0.956770108146255

EXPERIMENTAL SETUP - contd.

Confusion Matrix:



EXPERIMENTAL SETUP - contd.

Layer 4: Model Deployment

Model Export: Export the trained model to a format suitable for deployment (e.g., a serialized model file).

```
import pickle

filename = 'trained_model.sav'

pickle.dump(rf, open(filename, 'wb'))
```

API Development: Create an API to expose the model for use in real-time phishing detection. We have used FastAPI web framework along with ngrok for tunneling the local server to a public URL, and we have also employed uvicorn as the ASGI server.

Integration with Other Systems: Integrated the API with -

MERN web application

Streamlit web application

Developed a google chrome extension.

EXPERIMENTAL SETUP - contd.

Layer 5: Monitoring and Maintenance

Model Retraining: We are updating the database with new URLs (and their result) as predicted by the model.

With this updated database we can retrain our model, ensuring that the model stays updated with new trends in phishing attacks.

Feedback Loops: Report functionality is present in our website to collect data on false positives/negatives and we can adjust the model accordingly.

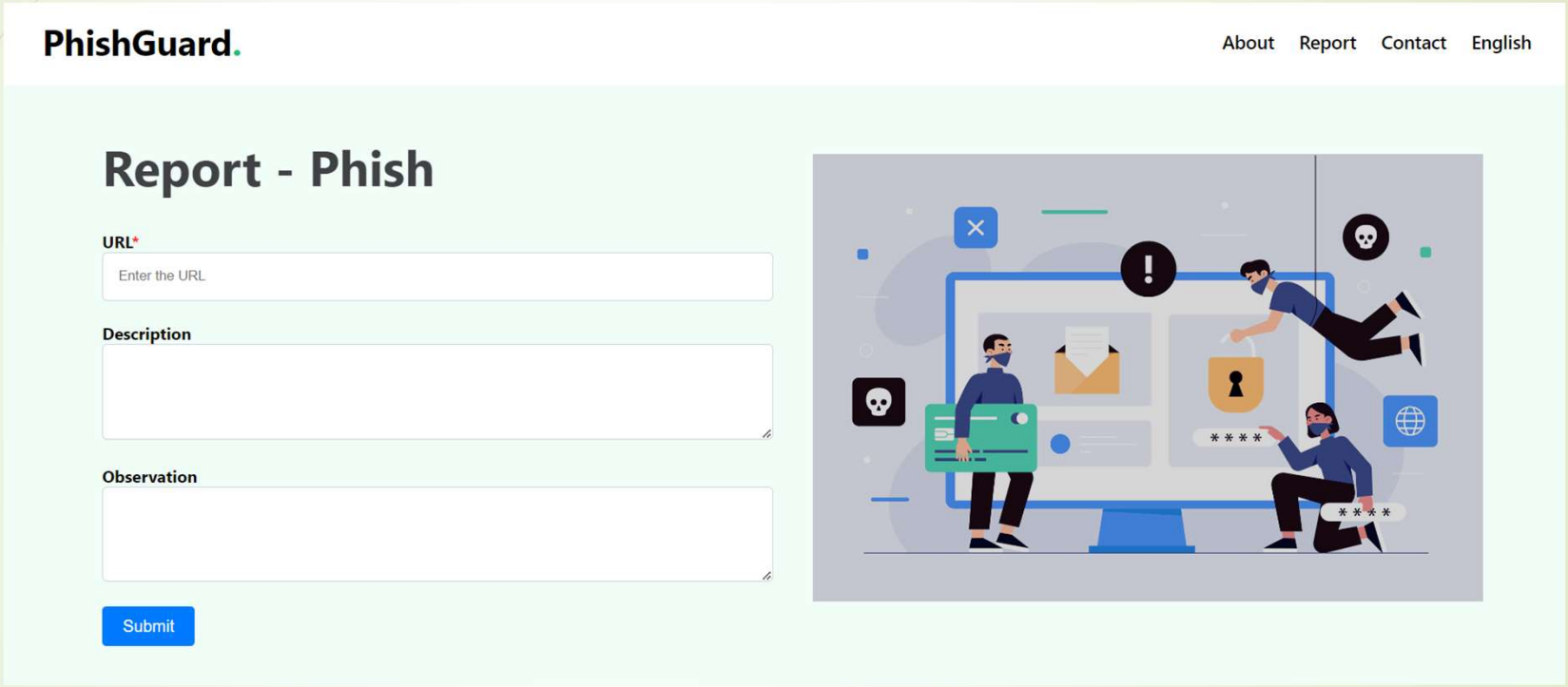
Layer 6: User Interface

Web-based Dashboard: Create a user interface to visualize detection results, allowing users to view flagged URLs and safe URLs.

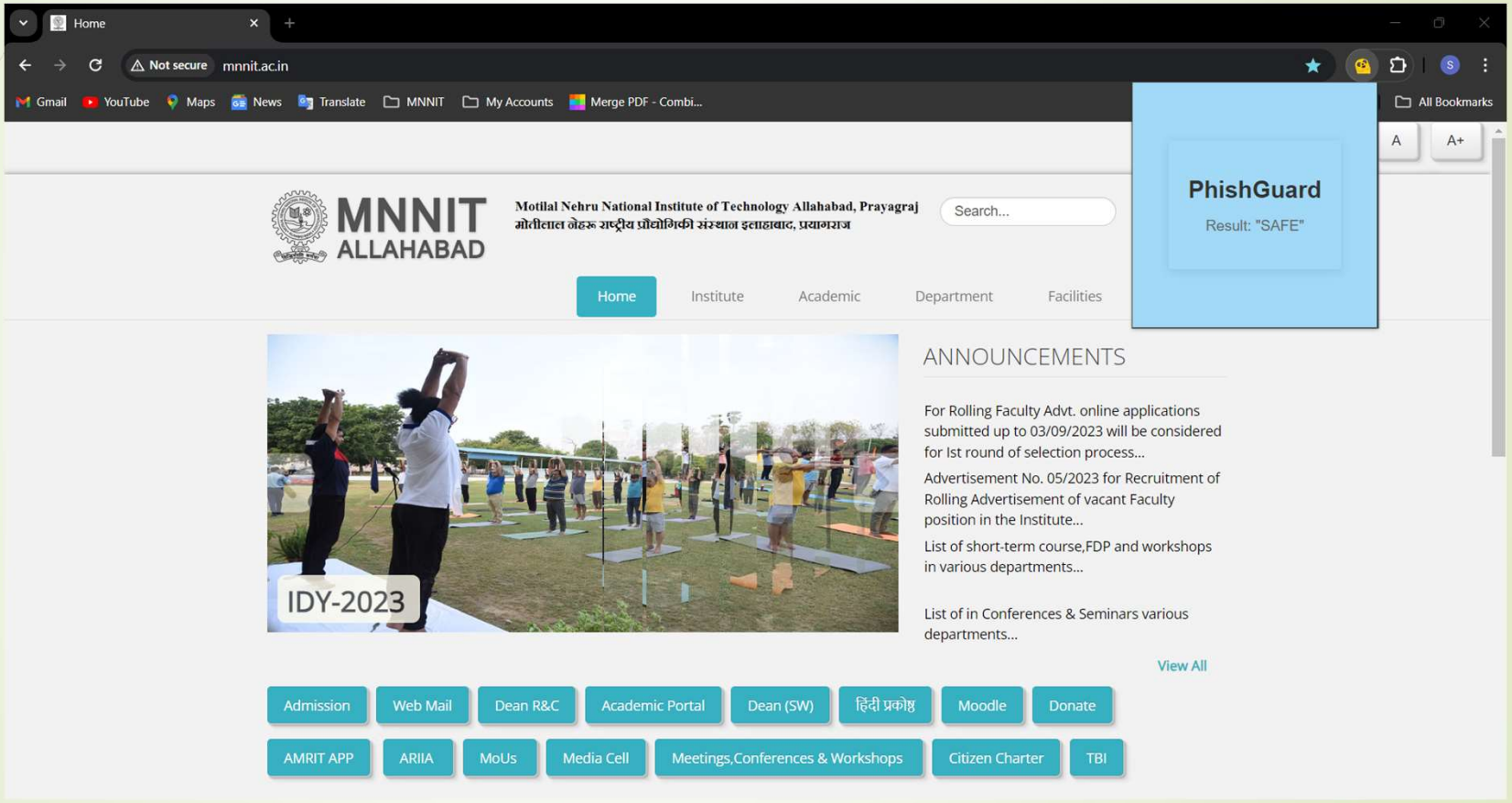
Extension: On-click extension to check if the URL entered in the search bar is phishing or safe.

Hosted Website - https://phish-guard.vercel.app/

Report URLs - https://phish-guard.vercel.app/



Google Chrome Extension



CONCLUSION

- Future work would include –
  - Enhancing Feature Engineering

    Explore new features that could improve model accuracy. This could include:

    Content-based features: Information derived from the content of the webpage.

    Behavioral features: Data on how users interact with the URL or the domain.

    Experiment with feature selection techniques to identify the most important features and reduce noise.
  - Advanced Machine Learning Techniques

    Consider using more complex machine learning algorithms or ensemble techniques to improve performance.

    Look into deep learning approaches like CNNs (Convolutional Neural Networks) or RNNs (Recurrent Neural Networks) to capture complex patterns in URLs.

    Use ensemble learning, such as Random Forests or Gradient Boosting, to combine the strengths of multiple models.
  - Data Augmentation and Expansion

    Expand the dataset with more phishing and legitimate URLs to ensure robustness.

    Use data augmentation techniques to artificially create more data, which can help prevent overfitting and increase model generalization.

    Collaborate with security experts or use external datasets to enhance the diversity and scope of your data.

CONCLUSION - contd.

- Model Adaptability and Continuous Learning

  Research methods to make the model adaptable to new phishing URLs detected by the model automatically.

  Develop a feedback loop where user-reported false positives or negatives are incorporated into model training.

- Converting from a binary classification to a multi-class classification

  Determining what classes we want to classify, for a phishing URL detection model, this could be:

  - Legitimate URLs
  - Phishing URLs
  - Malicious URLs (non-phishing)
  - Defacement URLs
  - Suspicious URLs (those requiring further analysis)

  In binary classification, ambiguous cases may be forced into one of two categories, leading to potential misclassification. Multi-class classification can reduce this risk by allowing for a broader range of categories. It provides a more detailed understanding of the data by categorizing it into more than two classes. This allows us to distinguish between various types of entities or situations, leading to deeper insights and more informed decision-making.

REFERENCES

[1] M. M. Gandomi, "Domain length," [Online]. Available: https://www.nature.com/articles/s41598-022-10841-5. [Accessed: May 2022].

[2] J. H. Kim, "Subdomain count," [Online]. Available: https://www.researchgate.net/publication/355896081_Phishing_Detection_Methods_A_Review. [Accessed: 11 November 2021].

[3] E. F. Williams, "Number of ampersands," [Online]. Available: https://www.researchgate.net/publication/358142755_Phishing_Attacks_Detection_--_A_Machine_Learning-Based_Approach. [Accessed: January 2022].

[4] N. O. Perez, "Presence of "https"," [Online]. Available: https://www.researchgate.net/publication/355896081_Phishing_Detection_Methods_A_Review. [Accessed: November 2021].

[5] Wisdom ML, (2023), Malicious URL Detection Using Machine Learning in Python, [Online]. Available: https://wisdomml.in/malicious-url-detection-using-machine-learning-in-python/.

REFERENCES – contd.

[6] S. Sheng, B. Wardman, G. Warner, L. Cranor, J. Hong, and C. Zhang,

"An Empirical Analysis of Phishing Blacklists,"

[Proc. 6th Conf. Email Anti-Spam (CEAS'09), Jul. 2009, pp. 59-78].

[7] A. K. Jain and B. B. Gupta,

"A novel approach to protect against phishing attacks at client side using auto-updated white-list,"

[Eurasip J. Inf. Secur., vol. 2016, no. 1, May. 2016].

[8] M. Zouina and B. Outtaj,

"A novel lightweight URL phishing detection system using SVM and similarity index,"

[Human-Centric Comput. Inf. Sci., vol. 7, no. 1, p. 17, Jun. 2017].

[9] Yang, Peng & Zhao, Guangzhen & Zeng, Peng, (2019), "Phishing Website Detection Based on Multidimensional Features Driven by Deep Learning," [Online]. Available:

https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8610190

[IEEE Access. PP. 1-1. 10.1109/ACCESS.2019.2892066].

[10] L. Tang and Q. H. Mahmoud, "A Deep Learning-Based Framework for Phishing Website Detection," [Online]. Available:

https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9661323

[IEEE Access, vol. 10, pp. 1509-1521, 2022, doi:10.1109/ACCESS.2021.3137636].

[11] S. Alrefaai, G. Özdemir and A. Mohamed, "Detecting Phishing Websites Using Machine Learning," [Online]. Available:

https://openaccess.iku.edu.tr/items/0f21ddac-ebf3-423c-988b-f15a555ab2ec