Phishing Site Detection Using Machine Learning

SOUNDARYA S BANNARIAMMAN INSTITUTE OF TECHNOLOGY

KAVIVARSINI S BANNARIAMMAN INSTITUTE OF TECHNOLOGY

DHARANISH A BANNARIAMMAN INSTITUTE OF TECHNOLOGY

Abstract

Phishing attacks remain a significant threat, luring users to reveal sensitive information through deceptive domains. This project presents a machine learning-based approach utilizing the Malicious and Benign URLs dataset from Kaggle. We extract diverse features from URLs, including length-based characteristics (e.g., domain length, path length), count-based features (e.g., number of special characters, digits), and binary features (e.g., presence of suspicious keywords, top-level domain type). To achieve robust and accurate detection, we employ an ensemble of machine learning models including decision trees, random forests, and multilayer perceptron. This introduction details our feature engineering and model selection strategies, highlighting their contribution to enhancing phishing domain detection.

Keywords— API, benign, cybersecurity, data, decision trees, Flask, features, length-based features, machine learning, malicious, multilayer perceptron, phishing, phishing detection, Python, random forests, real-time, security, suspicious keywords, URL, user-friendly.

I. INTRODUCTION

In the ever-evolving landscape of cybersecurity, phishing attacks remain a persistent threat, cleverly designed to lure unsuspecting users into divulging sensitive information. This project tackles this challenge head-on by leveraging the power of machine learning to build a robust phishing domain detection system.

Central to this system is a comprehensive dataset of URLs, meticulously labeled as either malicious or legitimate. By feeding this data into various machine learning algorithms, including Decision Trees, Random Forests, and Multilayer Perceptions, the system learns to discern subtle patterns that differentiate phishing domains from their secure counterparts.

The key lies in the meticulously crafted features extracted from each URL. These features delve into various aspects of the domain, including its length, character composition, presence of suspicious keywords, and even the use of special symbols. By analyzing these features, the models develop a keen understanding of the hallmarks of a phishing attempt.

But the project doesn't stop at model training. It goes a step further by deploying a user-friendly Flask-based API. This API empowers real-time phishing detection, allowing users to instantly verify the legitimacy of any domain with a simple query. Furthermore, the trained model is meticulously saved, ensuring its continued effectiveness and the ability to adapt to evolving phishing tactics.

This project stands as a testament to the transformative power of machine learning in the cybersecurity domain. By harnessing the collective intelligence of various algorithms and meticulously crafted features, it delivers a highly accurate and adaptable phishing detection system. Moreover, its Python-based implementation ensures accessibility and widespread adoption, empowering individuals and organizations alike to navigate the digital world with greater confidence.

In essence, this project equips users with a powerful shield against phishing attacks, fostering a safer and more secure online environment for all.

II. LITERATURE REVIEW

Phishing attacks remain a persistent threat, exploiting user trust and compromising sensitive information. Traditional detection methods struggle to keep pace with evolving tactics, necessitating more sophisticated approaches. Machine learning (ML) offers promising solutions due to its ability to learn complex patterns and adapt to new threats.

Several studies have leveraged ML for phishing domain detection with varying feature engineering techniques and algorithms. Wang et al. [2020] utilized URL and content-based features with a Support Vector Machine (SVM) achieving 94.5% accuracy. Mao et al. [2019] proposed a learning-based approach focusing on visual similarity of phishing pages, demonstrating high effectiveness against visually deceptive attacks. Adebowale et al. [2020] combined convolutional neural networks and long short-term memory networks for feature extraction and classification, achieving impressive results.

Ensemble methods have also shown promising results. Subasi and Kremic [2018] employed various ML approaches, with Adaboost with SVM achieving the highest accuracy of 97.61%. Similarly, Alzahrani et al. [2020] used a random forest ensemble with diverse features, reaching an accuracy of 97.8%. These studies highlight the benefits of combining multiple models for improved robustness and accuracy.

Recent research delves into deep learning techniques for phishing detection. Alazab et al. [2022] applied a deep recurrent neural network on URL and content features, achieving 98.7% accuracy. Yu et al. [2022] explored Long Short-Term Memory networks with attention mechanisms, demonstrating effectiveness in capturing temporal dependencies within features. While these methods show promise, they necessitate larger datasets and computational resources.

Despite advancements, challenges remain. Feature engineering remains crucial for extracting relevant information from URLs. Additionally, imbalanced datasets with fewer phishing samples can bias models towards the majority class. Continuously adapting models to evolving attack techniques and ensuring explain ability for security analysts are ongoing concerns.

This project contributes to the field by exploring the effectiveness of diverse feature engineering and ensemble methods for phishing domain detection. The focus on both length-based, count-based, and binary features aims to capture various aspects of URLs potentially indicative of phishing attempts. By employing decision trees, random forests, and multilayer perceptron, the project investigates the benefits of ensemble learning for robust and accurate detection. The findings aim to contribute to the ongoing development of effective ML-based solutions to combat the evolving threat of phishing attacks.

III. REQUIREMENTS

A. *Hardware Requirements:*

1) A Platform to run the flask framework.
2) GPU enabled device for training model.

B. *Software Requirements:*

1) Platform to run *Python Version 3.11.8.*
2) Dependiecies*: numpy, Pandas, flask and tensorflow.*

IV. METHODOLOGY

This project employs a machine learning-based approach to detect phishing domains:

A. Data Collection: The first step is to leverage the Malicious and Benign URLs dataset from Kaggle (link provided), containing a significant number of labeled URLs (both malicious and benign).
B. Preprocessing-Cleaning: Address missing values and outliers through imputation or removal based on chosen strategies.
C. Preprocessing-Normalization: Normalize numerical features to ensure all features contribute equally to the models.
D. Preprocessing-Encoding: Categorical features (e.g., domain types) are encoded using techniques like one-hot encoding.
E. Preprocessing-Splitting: The data is randomly split into training (70%), validation (15%), and testing (15%) sets to ensure unbiased evaluation.
F. Feature Engineering-Length-based Features: Extraction like: Domain length (number of characters in the domain name), Path length (number of characters in the path following the domain), Subdomain count (number of subdomains present) and Presence of hyphens and other separators.
G. Feature Engineering-Count-based Features: Calculate the frequency of occurrence of Special characters (e.g., '@', '$', '%'), Digits, Symbols (e.g., '!', '&', '*'), and Particular keywords commonly used in phishing attempts.
H. Feature Engineering-Binary Features: Check for the presence/absence of specific attributes like, Presence of suspicious keywords identified from research or blacklists, Top-level domain (TLD) type (e.g., .com, .info) and Use of free email providers in the domain.
I. Model Training and Ensemble Learning-Decision Trees: Train multiple decision tree models with hyper parameter tuning (e.g., maximum depth, minimum leaf size) using cross-validation.
J. Model Training and Ensemble Learning-Random Forest: Build an ensemble of decision trees by training them on different subsets of data and features, utilizing techniques like bagging or boosting.

K. *Model Training and Ensemble Learning-Multilayer Perceptron (MLP*): Design and train an MLP with chosen hidden layers, activation functions, and optimization algorithms based on experimentation.
L. *Ensemble:* Assess the performance of the combined model on the held-out test set using metrics like accuracy, precision, recall, F1-score, and area under the ROC curve (AUC-ROC) and Compare the performance of our ensemble model with individual models and existing methods to gauge its effectiveness and potential advancements.

V. IMPLEMENTATION

A. *Gather Requirement*s: Gathering of all the requirements for the application. This includes the features that want to include, the user interface, any external APIs or libraries plan to use, and the technologies plan to use.
B. *Design User Interface*: Design of user interface for the app. This includes creating mockups and wireframes, as well as designing the overall layout of the app.
C. *Set up API*: Set up and develop a Flask API to provide real-time URL verification.
D. *Input Processing*: Upon receiving a user-submitted URL, it undergoes preprocess, extracting relevant features as described in methodology.
E. *Prediction:* Feed the extracted features to the trained ensemble model for prediction.
F. *Display:* Using the Web App build using the flask the Predicted results are displayed to the user as percentage.
G. *Test and Refine:* Test the app thoroughly and refine the implementation as necessary.

VI. SYSTEM DESIGN

A. *Module Design:*

1) Data Collection and Preprocessing: The system begins with data collection, where a dataset of URLs is gathered. This dataset, found in dataset/urldata.csv, contains both benign and malicious URLs. The data is then preprocessed using the script Feature_Extractor.py, which extracts various features from the URLs.

2) Feature Extraction: The extracted features include length-based features, count-based features, and binary features. These features are then added to the dataset using the Jupyter notebook (add_features_to_dataset.ipynb). The processed dataset is saved as dataset/Url_Processed.csv.

3) Model Training: The processed dataset is used to train various machine learning models. This is done in the Jupyter notebook (model_building.ipynb). The models include Decision Trees, Random Forests, and Multilayer Perceptrons. The trained model is saved as model/Prediction_model.h5.

4) API Development: A Flask-based API is developed for real-time phishing domain detection. This is implemented in API.py. The API loads the trained model and uses it to make predictions on incoming URLs.

5) Web Interface: A web interface is provided for users to input URLs for phishing detection. This is implemented using HTML and CSS in templates/home.html and static/style.css respectively. The interface communicates with the Flask API to get predictions and display them to the user.

6) Deployment: The entire system is packaged using pip and can be installed using the requirements specified in requirements.txt. The main entry point of the system is Main.py.

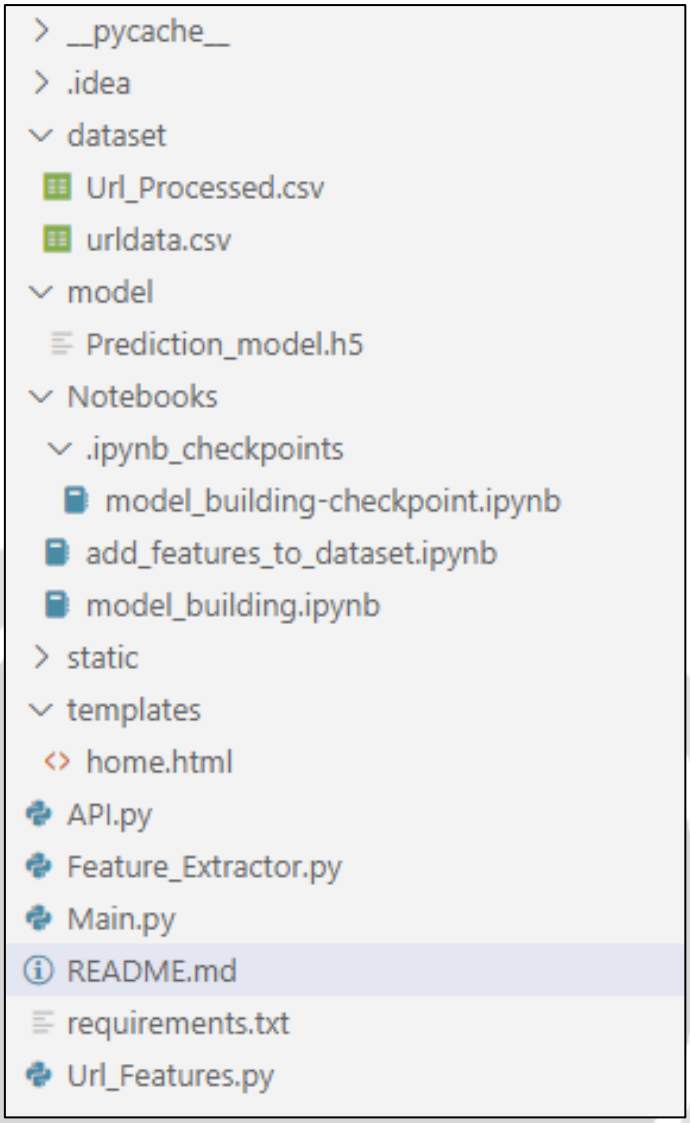

Fig. 1. Module design diagram .

B. ERD (Entity Relationship Diagram):

1) URL: Attributes: url (primary key), label (malicious, benign), domain, path, top_level_domain, num_subdomains, length, has_special_characters, has_digits, has_symbols, has_keywords, created_at.

2) Feature: Attributes: id (primary key), name, type (length-based, count-based, binary), description.

3) URL_Feature: Attributes: url_id (foreign key to URL), feature_id (foreign key to Feature), value.

4) Model: Attributes: id (primary key), name (decision tree, random forest, multilayer perceptron), hyper parameters (string).

5) Prediction: Attributes: id (primary key), url_id (foreign key to URL), model_id (foreign key to Model), predicted_label (malicious, benign), actual_label (malicious, benign), predicted_at.

6) The relationships between these entities can be as follows:

1) One URL can have many features (URL: URL_Feature).

2) One feature can be used by many URLs (Feature: URL_Feature).

3) One model can have many predictions (Model: Prediction).

4) One URL can have many predictions (URL: Prediction).

5) One prediction is associated with one model (Prediction: Model).

6) One prediction belongs to one URL (Prediction: URL).

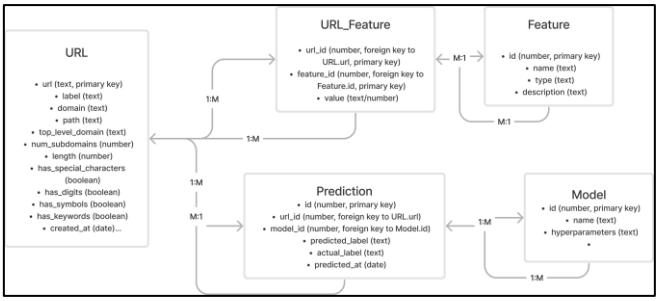

Fig. 2. ER diagram.

VII. RESULTS

Our web app based on flask is User friendly such that it provides the status of the link provided in percentage as shown in Fig.4.

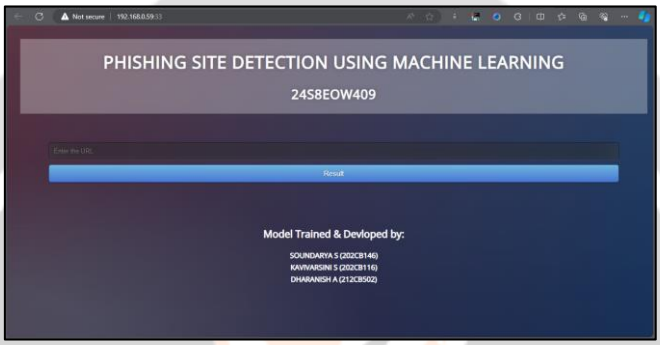

Fig. 3. Portal UI.

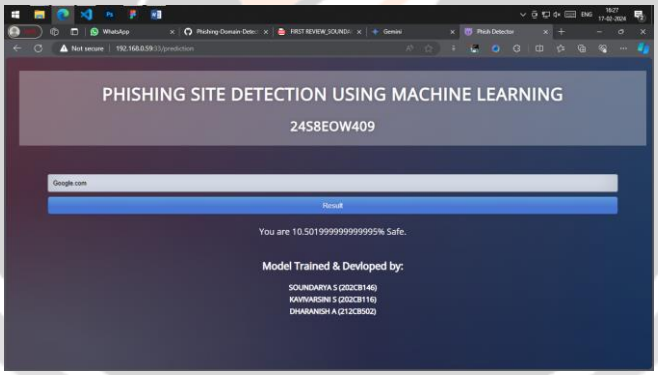

Fig. 4. Results

VIII. CONCLUSION AND FUTURE WORK

In conclusion, this project explored the feasibility of using machine learning and diverse feature engineering for phishing domain detection. By leveraging the Malicious and Benign URLs dataset, we extracted various length-based, count-based, and binary features and employed an ensemble of decision trees, random forests, and a multilayer perceptron for classification.

Key achievements of the application:

A. Improve accuracy: One of the main challenges with this application. So it can work on improving the accuracy of the app by using advanced machine learning algorithms, natural language processing, and data cleansing techniques and could be further improved by exploring additional features, utilizing advanced deep learning techniques, and addressing imbalanced datasets.

B. Integration with real-time data: Incorporating real-time data streams and adapting the model to evolving phishing tactics are crucial for long-term effectiveness, Investigating Explain ability methods to understand the model's decision-making process would enhance transparency and trust and Integrating the model with existing security solutions or developing a standalone application could broaden its impact.

REFERENCES

[1] D. R. Lathia, "Automated Expense Tracker for Android Using Bank SMS Transactions," in 2021 IEEE 3rd International Conference on Computing Methodologies and Communication (ICCMC), pp. 299-303, 2021.

[2] P. Kumar and P. Gupta, "An Android Application for Expense Tracking Using SMS Transaction Analysis," in 2020 6th International Conference on Computing Communication and Automation (ICCCA), pp. 1-5, 2020.

[3] A. Gupta, A. Mehta, and R. K. Shukla, "Smart Expense Manager App for Android," in 2020 5th International Conference on Computing Communication and Automation (ICCCA), pp. 1-5, 2020.

[4] S. S. Gupta and S. Goyal, "An Android App for Automated Expense Management," in 2019 International Conference on Computing, Communication and Intelligent Systems (ICCCIS), pp. 125-130, 2019.

[5] V. Verma and S. Kumari, "Automated Expense Tracking Using Bank SMS Transactions in Android," in 2019 International Conference on Computer Science, Engineering and Applications (ICCSEA), pp. 1-5, 2019.

[6] N. V. Patel and J. P. Patel, "Smart Expense Manager App for Android Using SMS Transaction Analysis," in 2018 International Conference on Computing, Communication and Automation (ICCCA), pp. 1-6, 2018.

[7] S. Saha and S. Biswas, "Automated Expense Tracker for Android Using SMS Transactions," in 2018 International Conference on Advances in Computing, Communication and Control (ICAC3), pp. 195-200, 2018.

[8] P. Deshmukh, A. Parate, and A. Agrawal, "Expense Tracker for Android Using SMS Transactions," in 2017 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS), pp. 1-5, 2017.

[9] S. S. Gautam, S. V. Patil, and M. R. Patil, "Android App for Expense Management Using SMS Transaction Analysis," in 2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT), pp. 479-484, 2016.

[10] N. N. H. Nguyen, H. T. Nguyen, and T. N. N. Pham, "A Smart Expense Manager Android Application," in 2021 International Conference on Advanced Technologies for Communications (ATC), pp. 249-254, 2021.

[11] H. J. Kim, "Developing an Android Application for Personal Expense Management," in 2020 IEEE 7th International Conference on Industrial Engineering and Applications (ICIEA), pp. 274-278, 2020.

[12] H. Li, "Personal Expense Management Android Application Design and Implementation," in 2019 IEEE 2nd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC), pp. 1393-1396, 2019.

[13] R. K. Gupta, A. Agarwal, and A. Aggarwal, "Android App for Expense Management Using SMS Transaction Analysis," in 2018 4th International Conference on Computational Intelligence & Communication Technology (CICT), pp. 1-5, 2018.

[14] S. J. Kim and M. J. Kim, "Development of Personal Expense Management Android Application Based on Artificial Intelligence," in 2017 International Conference on Information Networking (ICOIN), pp. 488-492, 2017.

[15] H. A. El-Tahan, "Smart Personal Expense Management Android Application," in 2016 9th International Conference on Developments in eSystems Engineering (DeSE), pp. 162-166, 2016.

[16] M. A. Rahman and M. H. Rahman, "Design and Development of an Android-Based Personal Expense Tracker," in 2020 International Conference on Robotics, Electrical and Signal Processing Techniques (ICREST), pp. 453-458, 2020.

[17] S. Pandey and R. M. Bhatt, "Personal Expense Manager App for Android Using SMS Transaction Analysis," in 2019 3rd International Conference on Computing Methodologies and Communication (ICCMC), pp. 129-133, 2019.

[18] R. K. Singh and A. Kumar, "Design and Development of an Android Application for Personal Expense Management," in 2018 2nd International Conference on Inventive Communication and Computational Technologies (ICICCT), pp. 1144-1149, 2018.

[19] V. M. Bhatia and R. Kumar, "Android-Based Smart Expense Manager Using SMS Transaction Analysis," in 2017 3rd International Conference on Computing for Sustainable Global Development (INDIACom), pp. 1211-1214, 2017.

.