**Project Report: AI-Powered Phishing URL Detection System**

**Student Name:** S.Srinubabu

**Date:** August 13, 2025

**Abstract**

Phishing remains one of the most pervasive and damaging threats in cybersecurity, leveraging social engineering to deceive users and harvest sensitive information. Traditional defense mechanisms, such as blacklisting, are fundamentally reactive and struggle to keep pace with the ephemeral nature of modern phishing websites. This report details the design, development, and evaluation of a proactive, multi-layered system for detecting phishing URLs in real-time. The system employs a hybrid methodology, integrating (1) a business logic "allow list" for trusted brands, (2) real-time domain reputation checks via the VirusTotal threat intelligence API, and (3) a custom-trained Random Forest machine learning model as a fallback for zero-day threat detection. A key innovation of this project is the development of a sophisticated feature set, including the implementation of the Levenshtein distance algorithm to specifically combat typosquatting attacks. The final system is deployed as a full-stack web application with a Flask backend, demonstrating high accuracy and, more critically, a recall rate exceeding 98% on a diverse test dataset of over 320,000 URLs.

**1. Introduction**

Phishing is a form of social engineering and a common cyber-attack where attackers trick naive online users into revealing confidential information, often with the intent of fraudulent use. These attacks have grown in sophistication, moving beyond simple email scams to include highly convincing counterfeit websites that mimic legitimate services. The financial and personal data loss resulting from successful phishing attacks is substantial, making their detection a critical area of cybersecurity research.

Traditional methods for combating phishing, such as maintaining blacklists of known malicious sites, are insufficient. Attackers can register and deploy new phishing domains in minutes, and these sites often have a lifespan of only a few hours—a window too short for blacklists to be updated and propagated effectively. Therefore, a modern solution must be proactive and intelligent, capable of identifying malicious intent from the characteristics of a URL itself, even if it has never been seen before.

This project addresses this challenge by developing a machine learning-based detection system. The objective is to train a model that can predict phishing websites based on a set of engineered features, moving beyond reactive methods towards a predictive and preventative security posture.

**2. Project Objectives**

The primary objectives of this project were aligned with the goal of creating a comprehensive and effective detection tool:

* **Develop a High-Accuracy Model:** To train, evaluate, and select a machine learning model capable of classifying URLs as either "phishing" or "legitimate" with a high degree of accuracy and reliability.
* **Engineer an Advanced Feature Set:** To move beyond basic URL features by designing and implementing a robust set of structural, keyword-based, and semantic features to identify suspicious patterns. A key objective was to specifically address typosquatting attacks.
* **Build a Full-Stack Application:** To create a functional and user-friendly web application with a Python Flask backend and an HTML/CSS frontend, allowing users to submit URLs for real-time analysis.
* **Integrate Threat Intelligence:** To enhance the system's accuracy by integrating a third-party, real-time threat intelligence API (VirusTotal), creating a hybrid system that leverages both internal model predictions and external security data.

**3. System Architecture and Design**

The system was designed as a multi-layered, sequential decision-making process to ensure both speed and accuracy. A request is handled by a Flask backend API, which routes the input URL through a three-tier decision engine.

*(A simple block diagram should be inserted here showing: User Input -> Flask Backend -> Decision Engine -> Final Verdict)*

The Decision Engine operates with the following prioritized logic:

1. **Business Logic Rule (Allow List):** Upon receiving a URL, the system first calculates the Levenshtein distance between its primary domain part and a curated list of known, trusted brands (e.g., 'google', 'amazon'). If the distance is zero, it signifies a perfect match. The URL is immediately classified as "Legitimate" and given a 100% confidence score. This rule acts as a powerful "allow list" that prevents the system from incorrectly flagging major legitimate sites (a False Positive).
2. **Threat Intelligence API Call:** If the URL is not a verified brand, the system then queries the VirusTotal API with the domain. VirusTotal checks the domain's reputation against over 70 real-time security vendor databases. If any vendor flags the domain as malicious, the system classifies the URL as "Phishing" and reports the number of security vendors that found a threat. This provides a rapid and highly reliable verdict for known threats.
3. **AI Model Prediction:** If, and only if, the URL is not a verified brand and is unknown to VirusTotal (i.e., has a clean reputation score), it is passed to our custom-trained Random Forest model. The model analyzes the URL's feature vector and calculates a probability score. This serves as our primary defense against new, zero-day phishing attacks that have not yet been cataloged by threat intelligence services.

This architecture ensures that fast, low-computation checks (rules and API calls) are performed first, reserving the AI model for the most ambiguous cases, leading to an efficient and robust system.

**4. Data Acquisition and Preprocessing**

A large and diverse dataset is fundamental to training an effective machine learning model. A hybrid sourcing strategy was employed:

* **Legitimate URLs:** A list of 1 million legitimate domains was sourced from the **Tranco list**, which ranks the world's most popular websites. This provided a strong and clean baseline for "good" data. The approach of using a large list of legitimate URLs is similar to research methods that use sources like the University of New Brunswick's dataset.
* **Phishing URLs:** To ensure both volume and freshness, a dual-source approach was used. A large, static dataset of labeled URLs was obtained from **Kaggle**. This was supplemented with a real-time, verified feed of active phishing URLs from

**PhishTank**, a community-driven anti-phishing service.

These sources were combined, cleaned of duplicates and null values, and randomly shuffled to create a final, balanced "super dataset" containing over 320,000 URLs (approximately 166,000 phishing and 156,000 legitimate). This large, balanced dataset formed the foundation for training the final model. The process involved shuffling the data to ensure an even distribution for training and testing sets, a standard practice in data preprocessing.

**5. Feature Engineering**

The process of converting a raw URL string into a meaningful numerical vector is the most critical part of the project. Our approach evolved to include not just structural features but also an intelligent semantic feature to counter modern attack techniques. The features can be categorized as follows:

* 5.1. Structural & Keyword-Based Features:

These features, inspired by common address bar and domain-based analysis, capture the syntactic properties of the URL. They include:

* + url\_len: The total length of the URL.
  + has\_ip\_address: A binary flag indicating if the domain is an IP address.
  + num\_dots, num\_dashes\_domain, num\_slashes: Counts of various characters.
  + path\_len: The length of the URL's path component.
  + has\_login\_keyword: A binary flag for the presence of words like 'login' or 'signin'.
  + num\_at\_symbol: A binary flag for the '@' symbol in the URL.
  + num\_query\_params: The number of query parameters in the URL.
* **5.2. Intelligent Typosquatting Detection (Levenshtein Distance):** The most significant innovation in our feature set was the addition of a feature to specifically detect typosquatting (e.g., amazonn.in). To achieve this, we implemented the **Levenshtein distance** algorithm. Levenshtein distance is a string metric that measures the minimum number of single-character edits (insertions, deletions, or substitutions) required to change one word into another. A low distance indicates high similarity. The min\_levenshtein\_distance feature was created by calculating the Levenshtein distance between the URL's primary domain part and a list of well-known brand names. A distance of 0 indicates a perfect match, while a distance of 1 or 2 is a strong indicator of a typosquatting attempt. This feature proved to be the decisive factor in correctly classifying subtle lookalike domains.

**6. Model Training and Evaluation**

* **Model Selection:** The **Random Forest Classifier** from the Scikit-learn library was chosen for this task. It is an ensemble model well-suited for high-dimensional, tabular data and is robust against overfitting. While multiple models were considered, the project focused on refining a single, powerful model through iterative improvements. This contrasts with the reference project which compared six different models and found XGBoost to be the best performer.
* **Iterative Training:** The model was not trained once, but three times (V1, V2, V3). Each iteration corresponded to a significant improvement in the project:
  + **V1:** Trained on the initial dataset and basic features.
  + **V2:** Retrained on the enriched "super" dataset.
  + **V3:** Retrained on the "super" dataset with the new, intelligent Levenshtein distance feature.
* **Evaluation:** The model was split into an 80% training set and a 20% test set. Performance was measured using standard classification metrics. The final v3 model achieved an accuracy of over 98%. More importantly, it achieved a **Recall score of over 98%**, indicating that it successfully identifies more than 98 out of every 100 phishing websites in the test set, resulting in a very low number of dangerous False Negatives.

**7. Full-Stack Application Implementation**

To make the trained model usable, a full-stack web application was developed.

* **Backend:** A lightweight backend server was built using the **Flask** framework in Python. This server is responsible for loading the saved phishing\_model\_v3.joblib file, handling incoming HTTP requests, and executing the complete multi-layered decision logic.
* **Frontend:** A simple, intuitive user interface was created with **HTML and CSS**. The frontend provides an input field for the user to submit a URL and an area where the final verdict and confidence score are displayed in a clear, color-coded format.

**8. Conclusion**

This project successfully demonstrates the design and implementation of a modern, effective phishing detection system. By combining an advanced, custom-engineered feature set with a robust machine learning model and real-time threat intelligence, the system overcomes the limitations of traditional, static methods. The iterative development process, particularly the identification and resolution of the model's "blind spot" regarding typosquatting via the Levenshtein feature, was a critical learning experience and the key to the project's success. The final application is a powerful proof-of-concept for a real-world, multi-layered cybersecurity tool.

**9. Future Work**

While the current system is a robust proof-of-concept, several enhancements could be made in the future:

* **Browser Extension:** The system's logic could be packaged into a browser extension to provide automatic, real-time protection by scanning URLs before a user visits them.
* **Web Content Analysis:** The system could be expanded to not just analyze the URL, but to also fetch and analyze the HTML content of the webpage itself using Natural Language Processing (NLP) to look for suspicious text, forms, or scripts.
* **Automated Retraining Pipeline:** A CI/CD (Continuous Integration/Continuous Deployment) pipeline could be established to automatically fetch the latest data from PhishTank daily, retrain the model, and deploy the updated version, ensuring the system is constantly learning and adapting to new threats.

**10. Technology Stack**

* **Programming Language:** Python
* **Backend Framework:** Flask
* **Machine Learning:** Scikit-learn, Pandas
* **Data Handling & Model Storage:** Joblib
* **Key Libraries:** Levenshtein (for string distance), Requests (for API calls)
* **Frontend:** HTML, CSS
* **Development Environment:** Google Colab (for model training), VS Code (for application development)