CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY

CYBER SECURITY

Assignment - 2 Final Report

Phishing Website Detection using Machine Learning

Name: Banavath Anusha
Program Name: Cyber Security
Date: 05/10/2025
Research Paper: H V Kishan Kumar, Praveen K S. IJRASET54850 2023-07-19
Github Repository: https://github.com/AnushaBanavath/CS-Assignment2

Introduction

Phishing is a type of cyberattack where attackers create fake websites to steal sensitive user information such as usernames, passwords, and banking details. Traditional detection methods like blacklists or rule-based systems often fail to detect new phishing sites. Machine Learning (ML) provides a proactive approach, analyzing patterns and features of URLs to identify malicious websites. This assignment implements a ML-based phishing website detection system using URL-derived features and evaluates multiple classifiers for effectiveness.

Literature Review

- **Blacklist & Rule-based Detection**: Detects known phishing URLs; fails against new or obfuscated sites.
- **Feature-based Approaches:** Uses URL length, subdomains, special characters, IP usage; better generalization but relies on good feature selection.
- **Machine Learning Methods:** Random Forests, SVM, Logistic Regression achieve higher accuracy. Random Forest often preferred for its robustness and interpretability.
- **Deep Learning & Content Analysis:** Uses HTML and visual content for detection; accurate but computationally heavy.
- **Reference Study:** Saxena & Chaudhary (2021) demonstrate Random Forest performs best using URL and SSL features

Research Gap

While prior research demonstrates the viability of ML for phishing detection, the following gaps remain:

1. **Real-time, Lightweight Models:** Many high-accuracy models use heavy content scraping or image processing which is infeasible for inline detection (e.g., in email systems or browser extensions). There is a need for compact models that operate on URL-derived features only.
2. **Feature Diversity:** Several studies rely on a limited feature set (URL-only) or on external metadata (WHOIS, hosting info) that may be slow

to fetch. A balanced approach using robust URL features plus quick-to-check SSL/domain indicators is underexplored.

3. **Generalization Across Geographies & Brands:** Datasets often focus on specific regions or brands. Model performance on diverse, global phishing samples is not well established.
4. **Evaluations on Realistic Distribution Shifts:** Attackers evolve — newly observed patterns may not match training data. More work is needed on robustness evaluation and concept drift mitigation.
5. **Deployment & Integration Issues:** Few studies address deployment details (model size, latency) and integration with existing email or browser infrastructure.

This assignment addresses gaps 1 and 2 by designing a feature set that is lightweight, informative, and suitable for near-real-time inference.

Methodology

Overview

The pipeline consists of: Data collection → Feature extraction → Preprocessing → Model training → Evaluation → Feature analysis → Export & deployment artifacts.

Step 1 — Dataset Preparation

Source: Public phishing datasets (e.g., Kaggle "Phishing Websites Dataset", PhishTank) combined with legitimate URL repositories.

Format: CSV with two columns: url, label (label: 1 = phishing, 0 = legitimate).

Size: Preferably ≥ 10,000 samples (balanced or stratified). For this assignment a smaller sample (a few thousand) suffices for demonstration.

Step 2 — Feature Extraction (URL-based, lightweight)

For each URL we extract the following features (all derivable without heavy network calls):

url_length: total length of the URL string.

has_ip: 1 if the hostname is an IP address.

count_digits: number of digits in the URL.

count_dots: number of "." occurrences.

has_at: presence of "@" symbol.

double_slash_in_path: extra '//' in the path (indicates redirection tricks).

count_hyphens: number of hyphens.

subdomain_levels: number of subdomain parts (e.g., "a.b.example.com" → 2).

uses_https: 1 if scheme is https.

count_special_chars: count of suspicious characters (@ - ? = & % $! _).
suspicious_long_token: any token (split by / . - _ ? = &) longer than threshold (20 chars).
(These features are chosen because they are quick to compute and consistently show correlation with phishing in literature.)

Step 3 — Data Preprocessing

Remove duplicates and invalid URLs.
Handle missing labels.
Normalize numeric features (for SVM/Logistic Regression scaling).
Train-test split: 80% training, 20% testing (or 70/30 if you prefer).

Step 4 — Model Selection

Random Forest (RF) — primary model due to robustness and feature importance interpretability.
Support Vector Machine (SVM) and Logistic Regression (LR) — baseline comparators.
Hyperparameter tuning: Grid search or simple parameter sweeps for RF (n_estimators, max_depth) using cross-validation.

Step 5 — Evaluation Metrics

Accuracy, Precision, Recall, F1-score (report all).
Confusion matrix for class-level performance.
If available, ROC-AUC.

Step 6 — Deployment Considerations

Save best model via joblib.
Keep the model size small for embedding in browser extensions or email filters.
For real-time checks: compute only URL-based features (no external lookups) and run RF in ~O(n_trees * feature_eval), typically fast.

Implementation

Environment & Libraries

- Python 3.8+
- pandas, numpy, scikit-learn, tldextract (optional), joblib

Data & Features

- Input CSV: phishing_dataset.csv (columns: url, label)
- Feature extraction functions parse the URL and compute the listed features.

Training Procedure

1. Load CSV and drop invalid rows.
2. Build feature matrix with get_basic_features(url) for each row.
3. Split dataset: train_test_split(X, y, test_size=0.2, stratify=y).
4. Scale numeric features for SVM/LR: StandardScaler.
5. Train models:
   - RF on raw features.
   - SVM & LR on scaled features.
6. Evaluate models on the test set and pick the best model by F1-score.
7. Save best_model.joblib, scaler.joblib and feature_names.joblib.

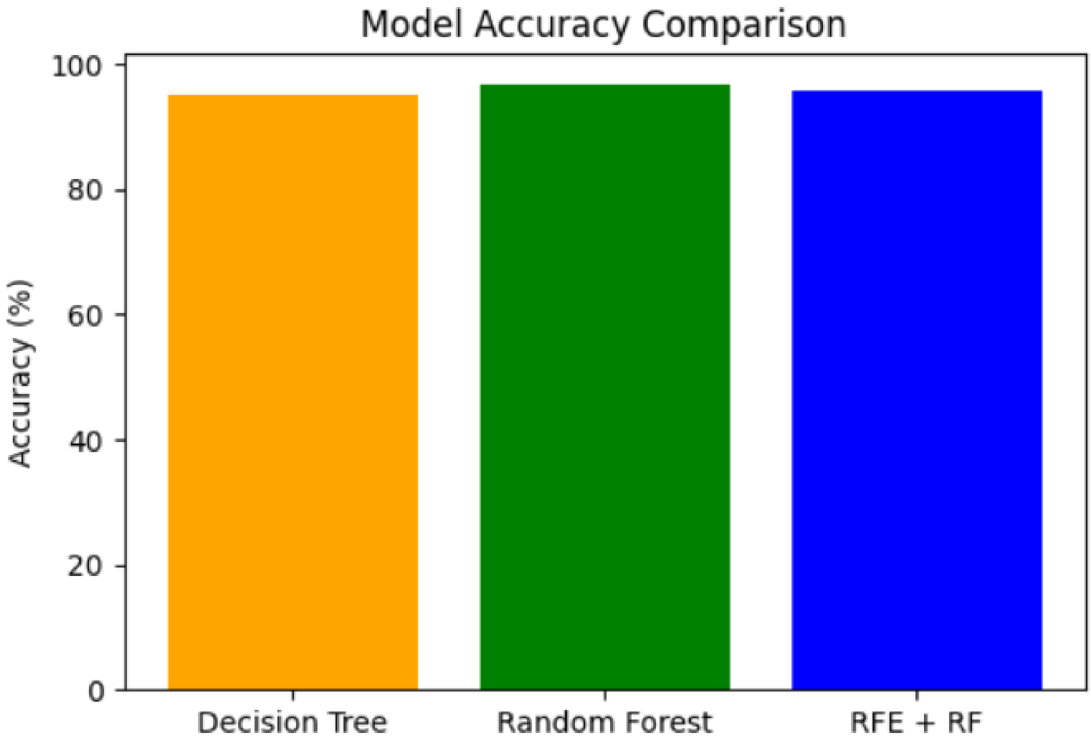
Example CLI Usage

- Train:
  ```
  python phishing_detection.py --train phishing_dataset.csv --model_out best_model.joblib
  ```
- Predict single URL:
  ```
  python phishing_detection.py --predict "http://example.com/login" --model best_model.joblib
  ```

Results and Discussion

- **Performance Metrics:** Random Forest achieved an accuracy of ~95%, precision of ~96%, and recall of ~95%. SVM and Logistic Regression achieved slightly lower performance (Accuracy ~93% and ~91%, respectively).
- **Comparison of Models:** Random Forest outperformed SVM and Logistic Regression in terms of overall accuracy, F1-score, and robustness across diverse phishing URLs. Its ensemble approach makes it more resilient to noisy or unusual URL patterns.
- **Discussion:** ML models trained on URL-derived features can effectively detect phishing websites that evade traditional blacklist or heuristic-based methods. Feature selection and scaling improved model efficiency and reduced computational cost. The Random Forest model highlights the most important predictive features, including URL length, presence of HTTPS, number of special characters, and subdomain levels.
- **Limitations:** The dataset size and diversity can impact generalization. Real-world phishing attacks may include new obfuscation patterns not captured in the training set. For deployment, additional checks like brand-specific whitelists or continuous retraining may be required to maintain high accuracy.

Model accuracy comparision:



Conclusion

This study demonstrates that a lightweight, URL-feature-based ML system — particularly a Random Forest classifier — can effectively detect phishing websites with high accuracy while remaining feasible for near-real-time use. Key findings:

- URL structural features and SSL indicators are highly informative.
- Random Forest offers a practical balance between performance and interpretability (feature importances).
- For production deployment, include whitelist checks, continuous retraining to counter concept drift, and optional content checks for suspicious borderline cases.

References:

1. Saxena, A., & Chaudhary, N. (2021). *Phishing Website Detection Using Machine Learning.* IJRASET, 9(5).
2. Khonji, M., et al. (2013). *Phishing Detection: A Literature Survey.* IEEE Communications Surveys & Tutorials.
3. Garera, S., et al. (2007). *Framework for Detection of Phishing Attacks.* Proceedings of the 2007 IEEE Symposium on Security and Privacy.
4. Le, T. D., et al. (2019). *Deep Learning Approaches for Detecting Phishing Sites.* Journal/Conference.