Cyber Security and AI - Task 1

Maya Ben Zeev , 211535513 | Nadav Yitzhak Goldrat , 211895933 | Group 2

Abstract

Phishing websites mimic legitimate pages to deceive users into revealing sensitive data such as credentials or payment information. In this report, we explore the key principles of phishing detection, the practical challenges involved in identifying malicious sites, and the role of machine learning in addressing these challenges. We present a lightweight, browser-based solution implemented as a Chrome extension, which integrates a Random Forest model to detect phishing attempts in real-time. The extension analyzes a webpage's features—derived from the URL structure and dynamic runtime indicators extracted directly from the web page—evaluates them using a trained ensemble model, and displays the results to users via the browser popup interface, providing immediate warnings about potentially malicious pages. Our solution avoids server-side reliance and instead relies on efficient client-side classification, enabling fast, robust protection against sophisticated phishing threats.

1 Introduction

Phishing websites are a common and dangerous cyber threat, often impersonating legitimate services to steal user data. These attacks lead to financial loss, identity theft, and data breaches. Traditional defenses like blacklists and signature filters struggle with detecting new or altered phishing sites, emphasizing the need for smarter, proactive detection.

Machine learning models analyze site characteristics in real time, using static URL-based, content-based, and dynamic behavior-based approaches. Our system combines static and dynamic features for improved accuracy. Static features like URL length and IP presence are quick to extract and predictive [1], while dynamic features—such as DOM changes and script execution patterns—help detect more covert threats [11], [6]. Together, these techniques form a robust detection system.

Running detection on the client side presents challenges. Extraction must be efficient and non-blocking, especially for dynamic features that require observing post-load behavior. We avoided server-side classification to ensure immediate protection and preserve privacy.

We selected Random Forest for its strong phishing detection performance and ability to handle different feature types [4], [5]. We evaluated multiple feature subsets and thresholds, aiming for a true positive rate (TPR) $\geq 97\%$ with minimal false positives. This balance helps detect most phishing attempts while avoiding user disruption [2], [3]. Threshold tuning and ROC-based selection led to a model that performs reliably in practice.

To enable in-browser operation, we trained the model in Python, converted it to JavaScript using m2cgen, and deployed it within a Chrome extension. Feature extraction runs in a content script, and classification occurs locally, providing real-time alerts without sharing user data.

2 Related Work

Extensive research has explored the use of static and dynamic features for phishing detection. Lexical URL properties such as domain length, symbol frequency, and characters like '@' and hyphens provide strong early indicators [1]. These are validated by real-world studies [2], [9]. We also included the is_free_hosting feature, highlighted in recent threat intelligence reports [12], which links free hosting services to phishing risk. Dynamic indicators such as the use of eval, DOM changes, and injected scripts help uncover malicious behaviors that evade static detection [6], [11].

Combining static and dynamic features strengthens detection across phishing strategies [3], [5], [11]. Random Forest models consistently achieve top accuracy in phishing benchmarks and outperform SVMs and neural networks [4], [5]. They are also fast, resistant to overfitting, and suitable for browser deployment [1].

To avoid overfitting and redundancy, we used a two-stage feature selection process. We first removed highly correlated features, then applied wrapper-based selection to evaluate combinations directly with the model [7], [8]. This helped reduce feature complexity while preserving predictive performance.

3 Dataset

We used a dataset consisting of phishing URLs sourced from PhishTank and benign URLs extracted from the Tranco top domains list. When preparing the dataset for model training, we considered two common approaches to class ratio: balanced (1:1 phishing to benign) and

imbalanced (e.g., 1:10). Research has found that while balanced training can improve recall and reduce false negatives, it may increase false positives, which are disruptive in real-world usage. In contrast, an imbalanced dataset more accurately reflects operational conditions and tends to improve precision and generalization [10]. Since we prioritized achieving a true positive rate (TPR ≥ 97%) with minimal false positives (FPR), we adopted the imbalanced ratio with 1:400. This decision reflects the tradeoff between maximizing detection and reducing user disruption, aligning with findings in phishing detection literature.

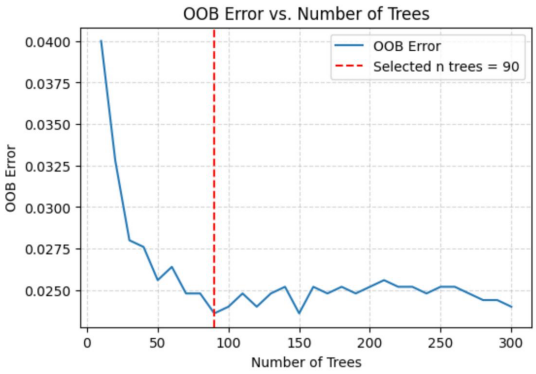
To improve the quality of benign examples, we developed a custom crawler that expanded root domain entries into more realistic URLs with subdomains, query parameters, and path depth. This was necessary because the Tranco data mainly includes root domains, which lack the structural complexity found in phishing URLs. Simulating realistic benign traffic helped the model better distinguish between legitimate and deceptive behavior.

To complete our dataset, we built an automated browser script that mimicked real Chrome behavior, visited each URL, and extracted both static and dynamic features. This ensured that each URL—whether phishing or benign—was analyzed under realistic browsing conditions, capturing runtime behaviors and client-side interactions. The resulting dataset includes 19 extracted features per URL, capturing both static and dynamic indicators.

4 Experimental Setup

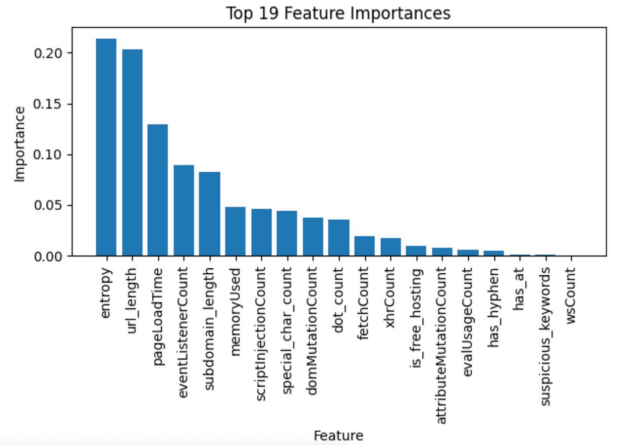
All model training and evaluation were performed in a Jupyter notebook using the scikit-learn library. To select the most effective configuration, we conducted a series of controlled experiments with visual analysis to guide our decisions.

1) Number of Trees (n_estimators): We evaluated Random Forest models with varying tree counts and observed the Out-of-Bag (OOB) error, since it gives a good estimate of model accuracy without needing a separate validation set. This makes it especially useful when working with smaller datasets or during quick testing [11]. We found that the error stabilized at around **90 trees** with OOB= 0.0236, which we adopted as the final model configuration (Figure 1).



2) Feature Importance: RF automatically ranks features by how much they help the model make accurate decisions. FI explains how much each one contributed to predictions - which helped interpret the model's behavior and later select the top n features that were the most meaningful for the prediction. As shown in Figure 3, lexical features like entropy and url_length, appeared most often across samples and had strong predictive power. Some dynamic features like pageLoadTime and eventListenerCount had lower but meaningful importance — not because they were ineffective, but because the behaviors they capture are less common in both phishing and benign sites. Despite that, we saw that these features were still valuable enough to be selected, showing that they provide meaningful signals when present (Figure 2).

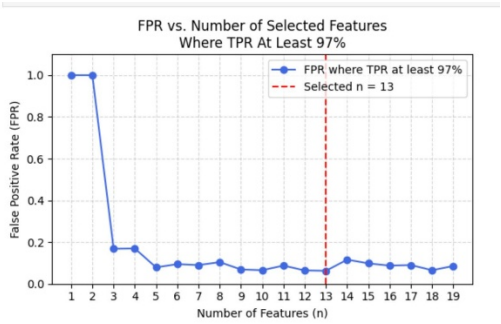
3) Feature Selection: We used a wrapper-based feature selection method that evaluated all subset sizes



from n=1 to N (number of features) features that maintained TPR ≥ 0.97. We then selected n size of subset features (n most important features) that the obtained the lowest FPR, and captured it's detection threshold. This method for feature extraction tests feature combinations directly with the model, rather than scoring features

individually, leading to better overall accuracy and less noise in the model [7]. This process led us to a set of **13 key features**. (Figure 3).

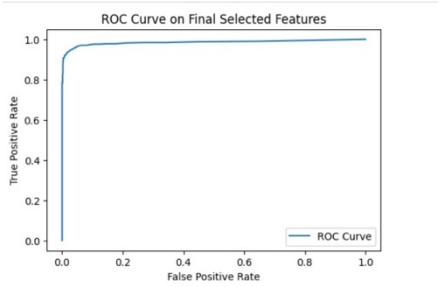
4) Threshold Choosing: In our Jupyter notebook, the optimal threshold determined from the ROC curve was 0.14, which achieved a high true positive rate (TPR) and



low false positive rate (FPR) during validation. However, when deploying the model, we found that a higher threshold of 0.86 (which is 1 – 0.14) yielded significantly better real-world performance, suggesting that the lower threshold may have been a misunderstand of us.

5 Results and Evaluation

The final model, trained on 13 selected features, achieved a True Positive Rate (TPR) of 97% and a False Positive Rate (FPR) of 6%. These results validate the strength of combining static and dynamic features in detecting phishing attempts. Our evaluation metrics were based on 5-fold cross-validation with out-of-fold predictions, ensuring realistic performance measurement.



6 Limitations and Future Work

Despite its effectiveness, our approach has several limitations:

1) Data Quality and Generalization: The model's ability to generalize to new phishing strategies is affected by the limitations of the benign dataset, which lacked the structural complexity often found in real-world phishing URLs. Future work should focus on curating a more diverse and challenging benign dataset, and investigating additional static features that can enhance robustness.

2) Detection Accuracy vs. Usability: While the model achieved a strong TPR, some false positives were observed. Given project constraints, we focused on maximizing detection. Future user testing and possibly adjusting business rules (e.g., prioritizing precision or adaptive thresholds) could better align detection with user expectations.

3) Dynamic Feature Limitations: We aimed to include richer runtime indicators, but browser API restrictions prevented access to lower-level metrics such as detailed JavaScript stack traces, window event handlers, or fine-grained network timing data. These features could further enhance detection depth.

4) Performance Profiling: While our implementation prioritized responsiveness, we did not systematically measure the browser's memory or CPU usage per feature. Adding targeted profiling would help validate performance across diverse environments, especially on lower-end devices.

Future improvements include:

1) Stacking Models: To boost detection power, we may explore combining Random Forest with complementary classifiers (e.g., logistic regression or XGBoost) via model stacking. This technique could capture different decision patterns and reduce variance.

2) Hybrid Detection: Integrating heuristic rules (e.g., suspicious keyword flags or form structure checks) with ML outputs may increase interpretability and robustness, especially for edge cases. However, this would require careful tuning and validation.

3) Privacy and Runtime Efficiency: While no data is sent externally, the collection of dynamic features still presents a theoretical privacy surface. We must ensure that memory or behavioral analysis features do not expose sensitive context. Evaluating the privacy cost of runtime signals is a key direction.

7 References

[1] R. Verma and A. Das, "What's in a URL: Fast Feature Extraction and Malicious URL Detection," Proceedings of the 26th International Conference on World Wide Web Companion, pp. 1093–1101, 2017.

[2] A. Jain and B. B. Gupta, "An assessment of features related to phishing websites using an automated technique," Procedia Computer Science, vol. 62, pp. 507–513, 2015.
[3] K. L. Chiew, K. S. C. Yong, and C. L. Tan, "A Survey of Phishing Attacks: Their Types, Vectors and Technical Approaches," Expert Syst. Appl., vol. 106, pp. 1–20, 2019.
[4] R. M. Mohammad, F. Thabtah, and L. McCluskey, "Predicting Phishing Websites Based on Self-Structuring Neural Network," Neural Comput. Appl., vol. 25, no. 2, pp. 443–458, 2014.
[5] Y. Zhang, J. Hong, and L. F. Cranor, "CANTINA: A Content-Based Approach to Detecting Phishing Websites," in Proc. 16th Int. Conf. World Wide Web, pp. 639–648, 2007.
[6] B. Li et al., "JSgraph: Enabling reconstruction of web attacks via efficient tracking of live in-browser JavaScript executions," in Proc. NDSS, 2018.
[7] S. Rambasnet, "Feature Selection for Improved Phishing Detection," 2021.
[8] A. Muhandisin, "Understanding Wrapper Methods in Machine Learning: A Guide to Feature Selection," Medium, 2022.
[9] Y. Ikram et al., "Phishing Detection Using Feature Selection and Machine Learning: A Study Based on Real-world URLs," arXiv:2011.04412, 2020.
[10] Y. Miyamoto et al., "Feature Analysis for Detecting Phishing Sites Based on HTML DOM," DEIM Forum, 2019.
[11] S. S. Alghamdi et al., "Phishing Website Detection Using Effective Features and Machine Learning," Secur. Priv., vol. 2018, Article ID 7087239, 2018.
[12] PhishLabs, "PhishLabs Releases Q1 Threat Trends & Intelligence Report," 2024.