Phishing Guard

Multi-Layered Phishing Detection Algorithm as a Browser Extension

Orr Sasson(206626103), Guy Meller(208982322)
orr.sasson@post.runi.ac.il, guy.meller@post.runi.ac.il

1. RELATED WORK

Phishing detection has been extensively studied through a variety of approaches, each leveraging different aspects of the attack surface.

URL-Based Detection Techniques:

Early phishing defences primarily used blacklists (e.g., Google Safe Browsing) and heuristic checks like detecting numeric IPs or "@" symbols in URLs. While blacklists yield low false positives, they cannot detect zero-day phishing sites. Heuristic methods, such as SpoofGuard, are faster but more error-prone(Zhang, Hong, & Cranor, 2007). To overcome these issues, researchers applied machine learning to lexical features including URL length, subdomain count, digit and symbol usage, and suspicious keywords. Classifiers like logistic regression, SVMs, and tree-based ensembles have been evaluated, with gradient-boosted decision trees (e.g., XGBoost) achieving superior results(Rathod & Mandwale, 2023). Utami et al. (2025) reported 99.8% accuracy using XGBoost trained on PhishTank and UCI datasets. However, ML models trained on static URL features may still misclassify cleverly crafted phishing links and lack broader context.

Content/DOM-Based Detection Techniques:

To address limitations of URL-only methods, researchers analyze webpage content. CANTINA, for example, uses TF-IDF to compare page content to its domain name, detecting inconsistencies common in phishing pages(Zhang, Hong, & Cranor, 2007). HTML structural analysis looks at form fields, external link ratios, iframes, onMouseOver events, and right-click restrictions. These features are effective for machine learning classification, with studies showing high accuracy when using rich DOM-based feature sets. Some approaches measure visual or structural similarity to known legitimate sites using techniques like DOM embeddings. While effective, such methods can be computationally intensive and vulnerable to dynamic or obfuscated page content.

2. METHOD

Data Collection: The development of our phishing detection models began with compiling a large dataset of phishing and legitimate websites. We aggregated data from multiple sources to ensure diversity. A primary source was the PhishTank repository, from which we obtained thousands of verified phishing URLs (with corresponding phishing webpages). To represent benign behavior, we collected legitimate URLs from the samples Zido gave us and more we add. After cleaning, we merged these sources into a unified dataset. In total, roughly 80,000 URLs were assembled, with an even split between phishing and legitimate examples (for balanced training). After dividing the features, which we will explain later, the data was partitioned into training and testing sets (we used 80% for training, 20% for

testing, ensuring that no domain overlaps leaked between train/test splits). This careful curation aimed to mimic real-world variety: our phishing examples ranged from generic fake login pages to targeted brand spoofs, and benign examples included popular websites as well as random personal or institutional sites.

Feature Extraction: We developed two independent feature sets, one for the URL model and one for the HTML content model, corresponding to the two layers of detection. For the URL analysis model, we implemented a suite of well-known lexical and host-based features, inspired by prior phishing research and public datasets. Key features include: *Have_IP* (whether the URL contains an IP address insted of a domain name), *Have_At* (whether an '@' symbol is in the URL path or query, which can obscure the real target URL), URL length (total number of characters), URL depth (number of "/" in the path, as phish URLs often have long paths), the count of URL redirections (// occurrences), and a binary flag if the domain part contains HTTPS. We also checked for URL shortening services (feature *TinyURL* flag), presence of a hyphen in the domain (*Prefix/Suffix* feature) which is uncommon for legitimate sites, and computed simple statistics like the count of digits and special characters in the URL (phishing URLs might embed random numbers or symbols). Additionally, we maintained a list of suspicious keywords (e.g. "login", "verify", "secure", "account") and checked if the URL contains such terms, which could indicate a deceptive intent. We extracted this URL features with simple string parsing and lookup operations implement in python.

For the HTML content model, we parsed each page's DOM to derive features that capture the presence of forms, links, and scripts that might indicate phishing. We counted the number of <form> elements and specifically noted if any form has an input of type "password" (*has_password_field*), since legitimate pages usually have password fields only on login pages (which should be on the site's own domain). We examined form actions: the ratio of forms whose submission target (action URL) points to a *different* domain (external) versus those pointing internal. A high external form action ratio (*form_action_external_ratio*) is a red flag (e.g., a fake page where the login form sends data to an attacker's server). Similarly, we computed the fraction of hyperlinks on the page that link to external domains vs. internal links (*external_link_ratio*). Phishing pages often include many external links (like images or CSS from the impersonated site, or references to unrelated sites) or sometimes none at all beyond the submit button – both cases differ from typical legitimate sites. We also looked at the fraction of "empty" or dummy links (*empty_link_ratio* – anchors with href="#" or similar), since phishers might copy a template but not populate all links fully, leaving placeholders. We extracted the number of external images and scripts included (*external_image_ratio*, *num_scripts*) as well as the maximum depth of the DOM tree (*dom_max_depth*) as a measure of page complexity. Simpler HTML with shallow depth can indicate a quick phishing kit, though this feature alone is not conclusive. We measured the total text length on the page and counted occurrences of known *suspicious keywords* in the text (such as "confirm", "login", "password", names of banks or popular services, etc.). Finally, we flagged the presence of specific HTML/JS tricks often used in phishing: any <iframe> elements any onMouseOver scripts (often used to hide the real link in the status bar), and any attempt to disable right-click context menu. Each of these became a binary feature. Feature extraction was implemented in Python, using pandas for data manipulation and custom HTML parsing code (built on Python's requests and BeautifulSoup to fetch and parse DOM content).

Deployment in Browser Extension: The ultimate goal was to integrate both models into a Chromium-based browser extension for real-time phishing detection. The extension was developed primarily in JavaScript (with HTML/CSS for the popup UI) and operates entirely on the client side for speed, privacy, and compliance with the task constraints no user data or URLs are sent externally. The

extension consists of a background script that processes URLs and a content script that analyzes the webpage DOM once it loads.

Upon navigation to a new page, the system triggers a two-stage evaluation: (1) the URL is parsed and passed to the URL classification model, and (2) the HTML content is scanned and passed to the content model. Both models were trained using XGBoost and exported to JSON format. We selected XGBoost as our final model due to its consistently high performance across all evaluation metrics, matching or surpassing alternatives like Random Forest and HistGradientBoosting, while offering faster inference times and better model export compatibility for integration in JavaScript. We implemented a lightweight JavaScript inference engine to evaluate these models in the browser. Feature extraction was also re-implemented in JavaScript, mirroring the logic used during training.

To determine whether a site is phishing, we experimented with multiple decision strategies. Initially, we considered a strict OR logic—flagging a page as phishing if either model predicted it. While this was effective in catching threats, it also produced unnecessary false positives. After evaluating performance trade-offs, we adopted a probabilistic ensemble approach, assigning 70% weight to the HTML model and 30% to the URL model, based on their individual accuracies. The final phishing score is a weighted average of both model outputs. If this combined score exceeds 0.71, the extension flags the site as phishing and alerts the user; otherwise, it is considered benign. This decision threshold was selected after empirical tuning to balance recall and precision.

By combining model predictions in this way, the system achieves better overall stability. For example, even if one model is moderately confident, the other must strongly agree to trigger a phishing alert—reducing overreactions to borderline cases. Both models run within milliseconds and consume minimal system resources, ensuring seamless integration into the browsing experience. The result is a client-side, privacy-preserving phishing defense that leverages two complementary models for layered protection.

3. RESULTS

Each model was evaluated separately on a held-out test set. The performance was measured using Accuracy, True Positive Rate (TPR), False Positive Rate (FPR), and weighted F1 Score, as required.

The URL-based model achieved an accuracy of 0.876, a TPR of 0.869, a FPR of 0.102, and a weighted F1 Score of 0.881. The HTML-based model reached an accuracy of 0.988, a TPR of 0.983, a FPR of 0.007, and F1 Score of 0.987. These results demonstrate that both models perform reliably in distinguishing phishing websites from legitimate ones, with the HTML model showing slightly fewer false positives and therefore receiving a higher weight in the final ensemble.

To improve overall robustness, the browser extension uses a weighted average of both model predictions, assigning 70% to the HTML model and 30% to the URL model. A site is flagged as phishing if the combined probability score exceeds a threshold of 0.71. This strategy was selected after testing multiple decision rules and was found to balance detection effectiveness and false positive rate more effectively than a simple OR condition. The results confirm that the combined model provides strong and efficient phishing detection within the constraints of real-time browser-based operation.

4. Limitations and Future Work

While the results are encouraging, our approach has a few notable limitations. Like many machine learning systems, it can occasionally produce false positives flagging legitimate websites as phishing. This typically occurs when benign URLs or pages exhibit unusual traits. For instance, auto-generated URLs with long, random character strings (common in password reset links or cloud storage) may resemble phishing patterns and be misclassified by the URL model. Similarly, benign pages that use iframes or disable right-click (e.g., for copyright protection) may be flagged by the content model. To mitigate this, we expanded our training data by including more legitimate examples with these characteristics, improving the model's ability to distinguish edge cases. However, the system's performance remains dependent on the diversity and quality of its training data. Sophisticated phishing attacks that deviate from known patterns may still bypass detection until retraining occurs.

Another limitation is the static nature of the HTML content model. It may miss attacks that involve dynamic behavior, such as delayed script execution or elements revealed only after user interaction. While the extension attempts to re-evaluate the page upon specific events (e.g., form clicks), this mechanism is not foolproof.

In summary, our multi-layered approach provides strong coverage across common phishing indicators, but like any heuristic-driven system, it is not immune to false positives or highly adaptive threats.

There are several avenues for future work to enhance the system. Initially, we experimented with several decision strategies for combining the two models. Although a simple OR condition (triggering an alert if either model predicts phishing) provided high recall, it also introduced unnecessary false positives. We ultimately adopted a weighted voting approach, assigning 70% weight to the HTML model and 30% to the URL model. A site is flagged as phishing only if the combined probability exceeds a threshold of 0.71. This configuration offered a better balance between TPR and FPR in empirical evaluations.

In future iterations, we may consider replacing this manually defined weighting mechanism with a more adaptive meta-classifier that learns how to combine both models' outputs dynamically. This classifier could also incorporate additional signals (e.g., user interaction, page metadata) and learn to resolve disagreements between the models more intelligently.

Another avenue for enhancement is the integration of dynamic analysis. While our current approach relies solely on static features (from the URL and HTML), many phishing attacks now use JavaScript to load malicious elements only after page load or user interaction. Embedding a lightweight sandbox (e.g., using headless browsing in the background or via a companion cloud service) could allow the system to simulate real user behavior and capture such dynamic threats. However, incorporating dynamic analysis must be balanced against privacy, latency, and performance considerations.

Sources:

1. Zhang, Y. et al. CANTINA: A Content-Based Approach to Detecting Phishing Web Sites. WWW 2007cs.cmu.educs.cmu.edu.
2. Utami, M.R.T. et al. Enhancing Phishing Detection: Integrating XGBoost with Feature Selection Techniques. SSRN Preprint 2025papers.ssrn.compapers.ssrn.com.
3. IJRPR. A Chrome Extension to Detect Phishing Website. 2023ijrpr.comijrpr.com.