State of the Art on Malicious Address Detection in Web Applications: A Comprehensive Analysis

Bruno Camarneiro[1], Gustavo Lima[1], Arsénio Ferraz[1], César Vieira[1], Rui Soares[1]
[1] Instituto Superior de Engenharia do Porto (ISEP), Porto, Portugal
Master's in Artificial Intelligence Engineering
{1250422, 1221349, 1010137, 1241523, 1221283}@isep.ipp.pt

Abstract

The increasing sophistication of phishing campaigns demands detection systems capable of reasoning beyond static blacklists. This paper presents an expert system for phishing URL detection built around a Prolog-based inference engine, performing layered analysis with rule-based logic and providing explainable decision-making. The system accepts a Uniform Resource Locator (URL) as input and first conducts a static analysis, extracting lexical features (e.g., URL structure, domain attributes) and Domain Name System (DNS)–derived signals to produce a preliminary risk score. If static indicators yield an inconclusive assessment, the system transitions to a dynamic evaluation phase. In this phase, the target webpage is fetched in a controlled environment to analyze its Document Object Model (DOM) structure, interactive content, and client-side behaviors. All decisions are derived from explicit rules in an expert knowledge base, ensuring traceability by identifying which rules fired and why. The proposed architecture emphasizes *explainability* and modular design: security analysts can inspect which heuristic rules triggered the classification, and the system can easily integrate new rules or external intelligence sources. Our methodology demonstrates that a logic-driven approach can achieve competitive phishing detection accuracy while maintaining transparency. Key contributions include a two-phase detection pipeline (static analysis augmented by dynamic analysis), a Prolog rule base encoding expert knowledge of phishing patterns, and an explainable scoring mechanism that collectively advance the state of the art in malicious address detection.

Keywords: Phishing Detection; Malicious URL; Web Security; Rule-Based Inference; Prolog; Explainable AI; Dynamic Analysis

1. Introduction

Phishing remains one of the most prevalent cyber threats, leveraging social engineering to deceive users into divulging sensitive information. Recent reports underline the growing scale and complexity of phishing attacks. The European Union Agency for Cybersecurity (ENISA) Threat Landscape 2024 report [1] highlights a diversified threat landscape where credential theft, phishing-as-a-service, and malware delivery infrastructures co-evolve with defenders, rendering purely reactive approaches ineffective. In the follow-up 2025 threat landscape report [2], ENISA identifies phishing as the predominant initial intrusion vector in ~60% of documented incidents, far outpacing vulnerability exploitation (~21%) which often leads to malware deployment. Likewise, the World Economic Forum's *Global Cybersecurity Outlook 2025* [3] indicates that 72% of organizations perceive cyber risk to be increasing, with a marked surge in phishing and social engineering incidents alongside concerns over adversarial uses of generative AI.

This landscape motivates detection mechanisms that are both *proactive* and *explainable*. Purely black-box machine learning solutions, while effective in static evaluations, may struggle to adapt to novel attack tactics and often lack interpretability for security practitioners. In contrast, an expert system grounded in **rule-based inference** can incorporate human-understandable heuristics and adapt to emerging phishing patterns by updating its knowledge base. The system presented in this work adopts a *biphasic* detection model: it first performs a pre-fetch static analysis of the URL and its registrable domain attributes to generate an initial risk assessment; if this assessment is inconclusive, the system conducts a controlled, post-render dynamic analysis of the landing page's content and behavior to reach a final verdict. By separating static and dynamic analysis phases, the approach balances efficiency (avoiding unnecessary page loads) with thoroughness when needed.

Our focus on a Prolog-based inference engine arises from the need for **transparent and explainable decision-making**. Prolog (Programming in Logic) is a declarative logic programming language well-suited for knowledge representation and reasoning. It allows us to encode phishing detection heuristics as logical rules and facts, enabling the system to infer conclusions through logical deduction. Unlike traditional procedural code or complex machine learning models, Prolog rules yield human-readable explanations (each triggered rule can be associated with a rationale). This design choice aligns with the growing demand for explainable AI in cybersecurity, where understanding *why* a URL is flagged is as important as the binary classification. Indeed, rule-based AI agents offer advantages in transparency and ease of deployment, making them valuable for security applications, and prior studies have shown that rule-based phishing detectors can achieve accuracy comparable to machine learning approaches (95–99% detection rates with low false positives). In the following sections, we review related work (static URL analysis, dynamic analysis, and hybrid methods), detail our Prolog-centered methodology, describe the system architecture and inference engine, present illustrative results of the rule-based detection, and discuss conclusions and future work.

2. Related Work

2.1 Static URL Analysis Techniques

Static URL analysis represents the first line of defense in malicious address detection, enabling risk assessment *before* any web content is loaded. Prior research has established a variety of lexical and host-based features that can signal phishing. Patil and Dhage [4] laid foundational work in anti-phishing by extracting URL characteristics such as URL length, presence of suspicious substrings or tokens, and domain age. Similarly, Hranický *et al.* [5] demonstrated enriched static analysis via lightweight DNS and Registration Data Access Protocol (RDAP) probes at the registrable domain (eTLD+1). By issuing SOA-guided queries to authoritative name servers for A/AAAA, CNAME, NS, MX, TXT records and examining TTL (time-to-live) values, as well as retrieving domain creation and registrar information via RDAP, they gathered infrastructure features to complement purely lexical indicators. We adopt aspects of this design to enrich our static feature set – these DNS- and WHOIS-derived signals are content-independent and comparatively harder for attackers to spoof at scale, thus providing a robust complement to URL string features.

Recent advances show that combining multiple static features improves detection performance. Aravindhan *et al.* [6] demonstrated that using a combination of lexical indicators (e.g., the presence of credential-related keywords, subdomain depth, use of internationalized domain names (IDNs), homoglyph characters) substantially increased accuracy versus single-feature approaches. When such lexical features are fused with DNS-layer evidence – for example, using resilient brand-specific whitelists and detecting anomalous nameserver relationships or typosquatting via tools like DNSTwist [7] – the static detector can leverage the regularities of legitimate ecosystems to better spot malicious anomalies. In parallel, researchers have explored graph-theoretic features: e.g., constructing a bipartite graph of domains to IPs obtained from Certificate Transparency (CT) logs and passive DNS can reveal *hosting churn* patterns indicative of phishing kit behavior [9]. These static analyses, by broadening the feature space beyond the URL string alone, strengthen early detection even before any page content is fetched.

Another dimension has been the integration of machine learning into static analysis. Nishitha *et al.* [10] proposed a hybrid model combining a Back Propagation Neural Network (BPNN) with an XGBoost (Extreme Gradient Boosting) classifier to evaluate URL-derived features, achieving 97.5% precision in phishing detection. Other work has constructed extensive feature vectors (e.g., 80 features including WHOIS, DNS, TLS certificate, and GeoIP features) and used gradient-boosted decision trees, reporting high precision (0.97+) and low false positive rates in large-scale evaluations [11]. These approaches illustrate that static analysis can be significantly enhanced with learning algorithms; however, the rules and decision criteria in such models are often opaque. Our work, in contrast, maintains interpretability by using a rule-based engine to evaluate similar features. Each static feature's contribution to a decision in our system is captured by a corresponding Prolog rule with an explicit condition and weight.

2.2 Dynamic DOM Analysis and Behavioral Detection

Dynamic analysis of DOM and web page behavior addresses the limitations of static analysis by examining the fully rendered page (after any client-side scripts execute). This approach can unveil run-time behaviors and interactive elements that static checks might miss. O'Mara *et al.* [12] showed that live page rendering enables detection of behavioral cues such as asynchronous network requests, client-side redirects, and dynamically generated form fields that are invisible to a pre-render parser. Their framework combined static and dynamic features over a dataset of 30,000 pages, achieving 96.3% accuracy and reducing uncertainty in borderline cases by 16%. Using a headless browsing tool (Puppeteer [13]), they loaded each page in an instrumented environment to log network requests, DOM mutations, and JavaScript calls. This revealed telltale signs of phishing, such as calls to `document.write` injecting content, unauthorized cross-origin data exfiltration, or suspicious `<iframe>` usage, whereas benign pages showed far less script entropy and more consistent resource domains.

Rose *et al.* [14] took a real-time approach by implementing a browser extension that monitors DOM and network activity as the user interacts with pages. The extension inspected dynamic content changes and JavaScript events for patterns like credential-harvesting forms appearing, unexpected redirects triggered by event handlers, or anomalous XMLHttpRequest patterns. Their system attained ~95.8% detection accuracy on live phishing versus benign sites by combining DOM mutation monitoring with network traffic analysis at run-time. These works underscore the value of dynamic content inspection: many phishing pages deliberately hide malicious elements until loaded in a victim's browser, so capturing this behavior is critical for a comprehensive detector.

Advanced dynamic analysis tools originally built for other purposes have also been repurposed for security. For instance, Bajaj *et al.* [15] introduced **LED (Live Editor for DOM)**, an interactive tool for generating DOM element locators. While aimed at web development, its core technique – programmatically exploring and extracting DOM elements using example-based constraints – achieved 98% recall in identifying interactive page elements and can be leveraged to systematically parse form fields or buttons on phishing sites. Such techniques could inform dynamic phishing detection by systematically identifying login forms or other interactive elements that phishing sites commonly abuse.

2.3 Ensemble and Machine Learning Approaches

Alongside knowledge-driven methods, ensemble machine learning models have shown promise in phishing detection. Omolara *et al.* [16] proposed **DaE^2** (Diverse and Efficient Ensemble), which aggregates multiple base classifiers (decision trees, support vector machines, gradient boosting, etc.) to improve prediction reliability while retaining efficiency. By leveraging algorithmic diversity, their ensemble mitigated overfitting and improved generalization across heterogeneous phishing datasets. Abad *et al.* [17] extensively evaluated Random Forest classifiers against SVM, decision tree, and k-NN baselines for malicious URL classification under Bayesian hyperparameter optimization, finding that ensemble methods consistently outperformed single models in both precision and stability across varying feature sets.

Researchers have also begun examining adversarial robustness and explainability in phishing detection. Alsmadi *et al.* [18] studied how generative adversarial networks (GANs) could generate subtle perturbations to both static and dynamic features of webpages to test detector resilience. They collected a dataset of 30,000 pages (using Puppeteer for dynamic content) and evaluated ensemble models before and after introducing adversarial modifications. The ensembles (Random Forest, Bagging Trees) achieved up to 96.2% accuracy on dynamic features, slightly above static-only performance, but the gap narrowed when facing adversarially perturbed inputs – highlighting that dynamic analysis adds value, but both approaches can be targeted by attackers. On another front, Çelik *et al.* [19] introduced a semantic analysis layer for phishing in financial webpages, using Natural Language Processing (NLP) techniques. By applying Term Frequency–Inverse Document Frequency (TF–IDF) and semantic similarity measures to page text and elements (e.g., looking for language inconsistencies or deceptive wording), they achieved ~79.8% precision. This suggests that content-based linguistic features can complement structural features for phishing detection.

2.4 Multi-Source Intelligence and Threat Correlation

Modern phishing detection systems increasingly leverage multiple intelligence feeds and data sources to improve robustness. Hranický *et al.* [20] integrated verdicts from external services such as VirusTotal, Cisco Talos, and PhishTank into their classification engine. By fusing independent threat intelligence sources, the system reduced false positives caused by noisy data from any single source and dynamically re-weighted confidence scores as new evidence arrived. This approach of cross-verdict aggregation proved effective for adaptive scoring in an expert system context. Alsabah *et al.* [21] took a content-agnostic view by correlating **Certificate Transparency (CT)** logs with passive DNS (pDNS) telemetry to detect phishing domains *before* they serve malicious content. They found that linking newly issued TLS certificates with DNS registration records can expose anomalous patterns (e.g., bursts of certificate issuance for domains with no prior history, or certificates for well-known brand names being deployed on suspicious infrastructure), enabling early flagging of phishing infrastructure even absent URL or content signals.

At the application level, Abyaa *et al.* [22] developed **LogoTrust**, a system leveraging the Brand Indicators for Message Identification (BIMI) standard to gather a verified dataset of legitimate brands, their domains, and official logos. In a dynamic analysis pipeline, LogoTrust can compare the logo extracted from a webpage against the BIMI-validated logo for that claimed brand. A mismatch or use of a known brand logo on an unrelated domain is a strong indicator of phishing. By incorporating such brand intelligence, detectors can catch visual impersonation tactics that purely textual or structural analyses might miss. This exemplifies the trend of combining technical signals with higher-level contextual knowledge (brand authenticity, reputation data, etc.) to strengthen phishing detection.

3. Methodology and System Design

Overview: The proposed system is an expert-based phishing detection framework that uses a Prolog inference engine to evaluate a series of handcrafted rules on data collected from a target URL. The detection process is split into two phases – a static analysis phase and a dynamic analysis phase – as depicted in **Figure 1**. In the static phase, the system gathers features from the URL *without loading the page*, including lexical patterns and DNS/WHOIS information. In the dynamic analysis phase, the system launches a headless browser to fetch and analyze the webpage's content and client-side behavior (e.g., forms, scripts, network requests). In the end, the system evaluated the score obtained and classified the URL as safe, suspicious, or phishing.

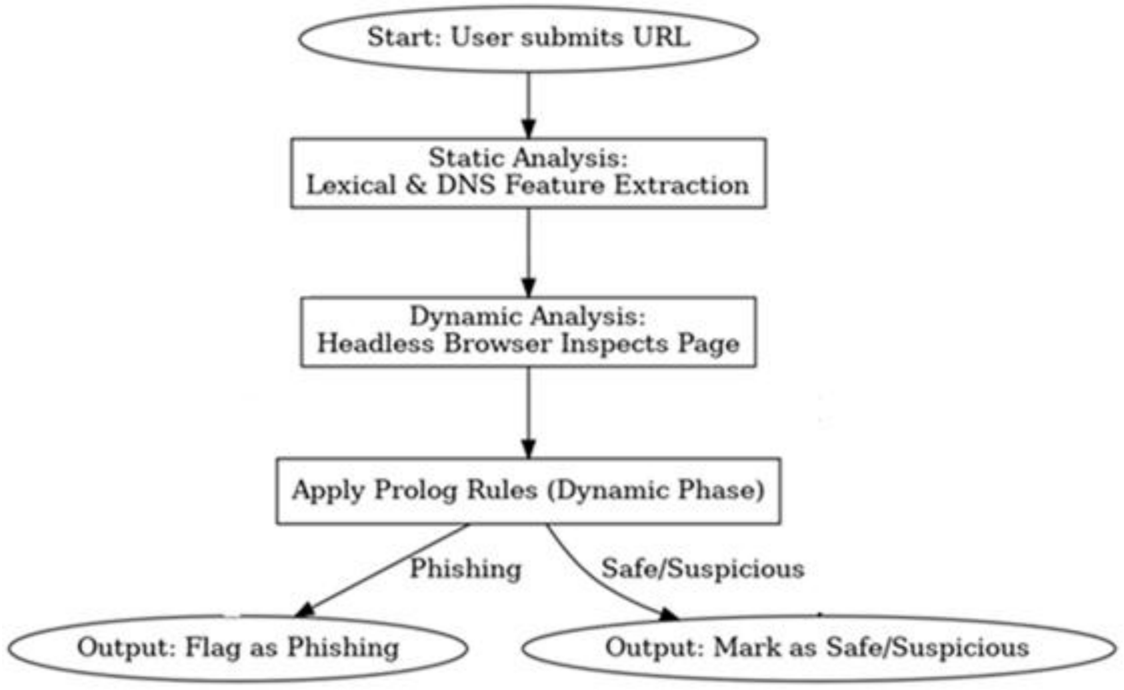
Start: User submits URL
Static Analysis:
Lexical & DNS Feature Extraction
Dynamic Analysis:
Headless Browser Inspects Page
Apply Prolog Rules (Dynamic Phase)
Phishing
Safe/Suspicious
Output: Flag as Phishing
Output: Mark as Safe/Suspicious

Figure 1. Flowchart of the phishing detection process.

3.1 System Architecture

The phishing detection system follows a modular architecture consistent with a modern web application setup (**Figure 2**). The front end is a React-based web user interface that allows an analyst or user to input a URL and select the analysis engine (our system focuses on the Prolog engine option). The back end is implemented in Python using the FastAPI framework, which exposes a RESTful API endpoint (e.g., `/analyze-url`) for URL analysis requests.

When a request is received, the back end orchestrates the analysis workflow. A **Feature Extraction module** handles static analysis: it parses the URL to extract lexical features (such as domain name length, presence of numeric characters or hyphens, subdomain count, specific keywords like "login" or "secure" in the path), and performs DNS queries and WHOIS/RDAP lookups via Python libraries (e.g., `dnspython` for DNS records) to retrieve host-based features (e.g., MX records existence, domain creation date, nameserver information). These extracted features are converted into Prolog facts. For example, if the domain is very young (say 5 days old), the system would assert a fact like `domain_age_days(URL, 5)`; or if the URL contains a known phishing token ("login", "secure"), it asserts `has_token(URL, login)` in the Prolog knowledge base. Table 1 (in Section 4) outlines some key features and their corresponding rules.

Concurrently, the static feature extractor can leverage **external intelligence sources**. For instance, the system may query threat intelligence APIs (like checking if the URL or domain appears in phishing blocklists such as PhishTank) and incorporate that as facts (e.g., `blacklisted(URL)` if found on a blacklist). These external checks are optional but enhance coverage by reusing community data on known threats.

The **Prolog Inference Engine** lies at the core of the system. We use SWI-Prolog (accessed via the `pyswip` Python-Prolog bridge) to load our rule base and assert facts at runtime. The knowledge base – comprising expert-defined rules and facts – is stored in Prolog source files (the main rule file is called `kb_rules.pl` and it is loaded by the engine at startup). Our rules are defined through a domain-specific rule language (DSL) built on top of Prolog. This DSL uses custom operators such as regra, se, and entao to express if-then logic in a more declarative and readable way. Each rule encodes a logical implication that the inference engine interprets internally.

For example, a simplified rule might be expressed as:

```
regra 5
  se [avalia(dns_tld_suspicious(Url, ==, 1))]
  entao [cria_facto(caracteristicas(Url, tldSuspicioso))].
```

Each rule in our system represents a logical condition that, when satisfied, contributes evidence toward a phishing classification. Continuing the previous example, the rule would infer that a URL exhibits a high phishing risk if its top-level domain appears in a suspicious list (e.g., *.xyz*, *.top*).

Every rule also carries an associated numerical weight, reflecting the severity of that indicator, and name. When a rule is triggered, its score is added to the URL's cumulative phishing risk total. During inference, the Prolog engine evaluates all DSL-defined rules against the asserted facts for each URL, producing a structured set of triggered rules and an aggregate risk score as output.

We have implemented rules targeting various known phishing indicators (detailed in Section 3.2).

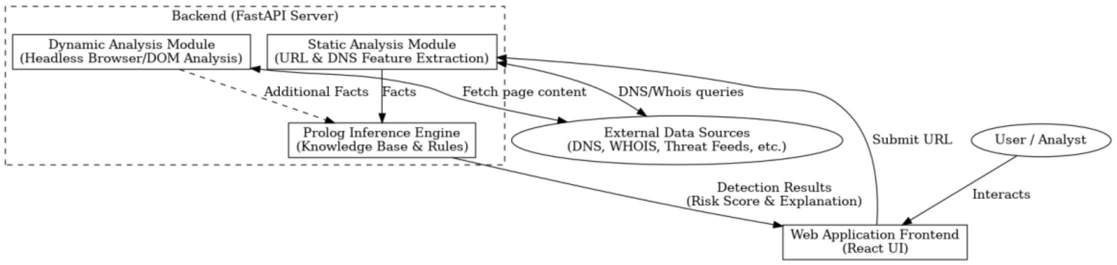
Backend (FastAPI Server)
Dynamic Analysis Module (Headless Browser/DOM Analysis)
Static Analysis Module (URL & DNS Feature Extraction)
Additional Facts
Facts
Fetch page content
DNS/Whois queries
Prolog Inference Engine (Knowledge Base & Rules)
External Data Sources (DNS, WHOIS, Threat Feeds, etc.)
Submit URL
User / Analyst
Interacts
Detection Results (Risk Score & Explanation)
Web Application Frontend (React UI)

Figure 2. System architecture of the phishing URL detection platform.

The user interacts with a web frontend, which sends the target URL to the backend API. The backend performs both static and dynamic analysis to extract a comprehensive set of features.

The static analysis module collects lexical, DNS, and WHOIS features and asserts them as facts to the Prolog inference engine. In parallel, the dynamic analysis module automates a headless browser (e.g., using Puppeteer or Selenium) to load the webpage corresponding to the URL. This module inspects the HTML/DOM structure and client-side activities to detect potential phishing indicators. For example, it checks for forms whose actions target external domains (a common sign of phishing), counts the number of iframes, measures script entropy to identify obfuscation, and monitors for silent redirects.

The dynamic module also captures network requests initiated by the page to determine whether user data might be transmitted to unknown servers, as well as anomalies in the JavaScript environment—such as extensive use of `eval` or inclusion of known malicious scripts. These observations are distilled into Prolog facts, such as external_form_action(URL) when a form points to another domain, or high_dom_entropy(URL) when the page contains heavily obfuscated scripts.

External data sources — including DNS and WHOIS services, threat intelligence feeds, and the target website itself — are queried by the analysis modules to enrich the fact base. The Prolog inference engine evaluates all asserted facts (both static and dynamic) against the rule knowledge base, where some rules specifically target dynamic behaviors like credential fields or abnormal redirect chains.

Each triggered rule contributes to the cumulative phishing risk score. After combining static and dynamic indicators, the engine produces a final classification. The backend then returns a JSON response to the frontend containing the risk level (e.g., “phishing” or “safe”), the total score, and an explainable list of triggered rules. The frontend presents these results to the user, for instance: “Rule Triggered: Suspicious TLD (.xyz) – adds 200 risk points” or “Rule Triggered: Missing MX records – adds 100 risk points,” thereby providing transparent reasoning behind the classification.

3.2 Prolog Inference Engine and Rule Base

At the heart of the system is the Prolog inference engine, which serves as a *knowledge-based reasoning* component. The engine uses backward-chaining logic resolution (depth-first search with backtracking) to determine which rules apply to the given facts (the URL's features). Our knowledge base is composed of **facts**, **heuristic rules**, and **meta-rules** for scoring.

Facts: These represent observed properties of the URL or domain. They can be binary (true/false flags) or numeric. Some examples of facts (with sample values) are: - `domain_length("example.com", 11).` – Domain name length is 11 characters. - `has_token("example.com/login", "login").` – URL path contains a credential-related token "login". - `num_subdomains("sub.dom.example.com", 2).` – URL has 2 subdomains (e.g., "sub.dom"). - `domain_age_days("example.com", 5).` – Domain was created 5 days ago (very young domain). - `has_MX_record("example.com", false).` – No MX record found for the domain. - `page_has_iframe("phishy.example.com", true).` – (Dynamic) The loaded page contains at least one `<iframe>` element. - `external_form_action("phishy.example.com", true).` – (Dynamic) A form on the page submits to an external domain.

Facts like the above are asserted into Prolog when the avrigua_caracteristicas(URL) is invoked, in the beginning of the inference engine. This predicate is responsible for invoking all other predicates that check each characteristic of the URL, such as its length, whether it is an IP address, and so on. For instance, given a URL, our Prolog code can split it to identify the domain and path, count path segments, check for the presence of numeric characters or hyphens in the hostname, etc. Using Prolog for such tasks benefits from conciseness of logical pattern matching (e.g., using DCG grammars or regex matching in Prolog for tokens).

Rules: Each detection rule is encoded as a Prolog clause that specifies certain conditions (usually combinations of facts or intermediate predicates) implying a phishing indicator, along with an associated *score*. We adopted a scoring model where each rule contributes a certain number of points to an overall risk score if its conditions are met.

The rules are designed based on known phishing heuristics from literature and empirical observation:

- **Suspicious Top-Level Domain (TLD)** – If a URL uses a TLD commonly abused by phishers (e.g., `.tk`, `.xyz`, `.top`), add +200 points.
- **Long Domain Name** – If the domain part of the URL is excessively long (>30 characters), add +150 points (as phishers often concatenate many terms to mimic subdomains).
- **Many Subdomains** – If there are more than 3 subdomains (e.g., `mail.login.user.example.com` has 3 subdomains: mail, login, user), add +180 points for a deeply nested domain which might indicate an attempt to spoof a path as subdomains.

- **High Numeric Content in Domain** – If over 30% of characters in the domain are digits (e.g., `www.paypal12345secure.com`), add +120 points[21], since legitimate domains rarely include long numeric sequences.
- **Multiple Hyphens in Domain** – If the domain has more than 2 hyphens, add +100 points, as phishers often chain words with hyphens to resemble legitimate sites.
- **IPAddress in URL** – If the URL uses an IP address instead of a domain name (e.g., `http://192.168.1.4/login`), add +300 points. This is almost always malicious as normal users don't use bare IPs for banking or email sites.
- **Credential Keyword in Path + Young Domain** – If the URL path contains keywords like "login", "secure", "account", etc., *and* the domain age is below a threshold (e.g. < 30 days old), add +250 points. The rationale: a newly created domain that immediately is used for a login page is highly suspect (Rule A1 in our knowledge base).
- **Suspicious TLD + Missing Email Records** – If the domain is on the suspicious TLD list *and* has no MX record or no SPF/DMARC records (suggesting it's not set up for legitimate email), add +150 points. Legitimate domains for businesses usually have email configured; phishers often do not bother.
- **Excessive URL Path Depth** – If the URL path has >= 6 segments (e.g., `http://example.com/a/b/c/d/e/f`), add +80 points. Deep paths can indicate buried phishing pages or kit artifacts.
- **Combination of Many Subdomains and Deep Path** – If both subdomain count ≥ 4 *and* path depth ≥ 6, add a higher combined score, e.g., +150 points (this is a compound rule reflecting an extremely complex URL likely auto-generated by a phishing kit).
- **No DNS Records (Dead Domain)** – If key DNS records like A or NS are missing or misconfigured (indicating a likely disposable or misused domain), add some points (this rule is less weight since missing records might just mean inactive domain).
- **Dynamic Content Rules:** (when dynamic phase is executed) e.g., **External Form** – If an HTML `<form>` on the page posts to a different domain than the current page (commonly observed when a phishing page steals credentials by sending to an attacker-controlled server), add +200 points. **Multiple Redirects** – If loading the page caused multiple redirects across different domains, that might indicate a phishing funnel; a rule adds points if, say, ≥3 redirects with at least one cross-domain hop occur (the rule weight might be lower, e.g., +50, as some legitimate services use redirects too). **Suspicious Scripts** – If the page's script entropy is very high or it includes known suspicious script URLs, add points. **Credential Field Present** – If the page contains input fields of type "password" or text fields in context of "username/email" without legitimate context, add points (though this can

also flag legitimate login pages, so it might be used in combination with other factors).

Each rule in the Prolog knowledge base has it own charachteristics annotated with a unique ID, a name for the rule and the score the specified rule increases. This helps in generating the final report and in the increasing score system. For example, a rule might be represented as:

```
caracteristicas_regras(30, url_at_char, 30).
```

Here, the rule with ID 30, wich is “url_at_char”, contributes with 30 points if it’s true.

Our static rule set currently includes about a dozen active rules covering the above aspects (see Section 4 for the list and their statuses). Additional placeholder rules (currently commented out or stubbed) exist for features we plan to integrate, such as homoglyph detection (Rule 21), detection of suspicious redirect chains (Rules 22–24), DNSSEC validation issues (Rule 28), Certificate Transparency anomalies (Rule 29), and other advanced DNS irregularities (Rules 30–33). As the necessary data becomes available to the Prolog engine via extended Python integration, these rules can be activated to further enhance detection coverage.

Inference and Scoring: When the Prolog engine runs the analysis for a given URL, it evaluates all rule clauses whose conditions are satisfied by the asserted facts. Each triggered rule contributes its corresponding score value, and the engine aggregates these through a meta-rule mechanism that computes the total accumulated score for the URL. For example, a predicate such as total_score(URL, Sum) sums all Score values for rules that succeed for that specific URL.

The resulting score and the list of triggered rules together form the core output of the inference process. This provides both a quantitative measure of overall phishing risk and a qualitative explanation of which indicators were detected. The scoring weights associated with each rule are configurable and were determined empirically through literature review and iterative testing on sample datasets. Each weight reflects the indicative strength of that condition—for example, using an IP address or having a suspicious TLD is typically assigned higher significance than having a long URL path.

Advantages of Prolog: Using a Prolog inference engine for rule evaluation offers several benefits:

- **Clarity of Rules:** The detection logic is expressed in a human-readable, declarative form that directly mirrors expert reasoning. Rules can be written as intuitive if–then statements without procedural complexity.
- **Backtracking Search:** Prolog’s native backtracking automatically finds all applicable rules without explicit looping.
- **Unification and Pattern Matching:** Phishing indicators often rely on pattern analysis—such as detecting domain names resembling known brands or matching

malicious URL templates. Prolog's unification mechanism enables concise and powerful pattern definitions for these cases.

- **Explainability:** Each rule corresponds to a clear explanation string, allowing the system to report precisely which conditions were met. This inherently satisfies explainable AI requirements, as the reasoning ("domain uses a suspicious TLD", "URL contains an IP address", etc.) is directly visible to users.
- **Ease of Updates:** New detection rules can be added simply by defining additional Prolog clauses in the knowledge base, which can be reloaded at runtime. This allows rapid adaptation to emerging phishing patterns without modifying or recompiling the core system.

The Prolog engine loads the rule base at startup and maintains it in memory. For each URL, the Python bridge asserts the extracted facts, queries the engine for the total score and triggered rule IDs (mapped to human-readable descriptions), and then retracts the temporary facts to prepare for the next analysis. This workflow is efficient, given that the number of rules is moderate, and Prolog's indexing and backtracking capabilities handle such rule sets with high performance.

3.3 Integration of Static and Dynamic Analysis

A key aspect of our methodology is the integration of static and dynamic analyses through the common Prolog rule engine. The static phase and dynamic phase are not siloed; rather, they share the knowledge base but apply different subsets of rules depending on what facts are available. In practice, we implement this by marking certain rules as static-only and others as dynamic-only through their conditions. For example, a dynamic rule might require a fact like `page_loaded(URL, true)` or some dynamic-specific fact to be present. Such a rule will not fire during the static phase because that fact isn't asserted until the page is actually loaded. Conversely, purely static rules do not depend on any dynamic facts.

This design ensures that when only static data is present, the engine effectively evaluates only static rules, and when dynamic data is added, additional rules can then trigger. The final decision logic uses both phases' contributions. We found this approach more seamless than trying to combine two completely separate engines. Initially, we considered using a separate forward-chaining rule engine (Drools) for dynamic analysis, but for simplicity and consistency we unified both phases under Prolog (Drools integration was prototyped but later dropped in favor of a single Prolog knowledge base, to avoid duplication of rule management). Prolog's expressiveness is sufficient for both phases, and by adjusting rule weights we can balance static vs dynamic influence on the final score.

Another integration point is **external threat intelligence**. Our system is designed to easily incorporate threat feeds as facts to influence the inference. For instance, if a domain is found on an allowlist of known safe domains (perhaps a company's internal domains), we could assert a fact like `whitelisted_domain(URL)` and have a rule that *lowers* the risk score or nullifies other rules' effects for that URL. (In Prolog, we could implement a dampening rule such as: if `whitelisted_domain` is true, then set risk score to 0 or mark as safe, effectively short-circuiting other conditions. In the current version, this allowlist dampener is planned as Rule 33, a placeholder.) Similarly, if a domain is flagged by an external feed, a fact `blacklisted_domain(URL)` could trigger a high-severity rule to automatically classify as phishing.

The integration of these various analyses is coordinated by the Python backend. As illustrated in Figure 2, the backend acts as the mediator: it invokes external lookups (DNS, WHOIS, CT logs, etc.) and the headless browser, gathers their results, and feeds everything into the Prolog engine. The stateless nature of Prolog queries means each request is handled independently with no cross-contamination of facts between analyses, which is important for a multi-user or multi-request system.

4. Results and Evaluation

While the development of a full benchmark and evaluation is ongoing, we present here illustrative results from the prototype to demonstrate the system's effectiveness and explainability. We tested the rule engine on a collection of URLs covering known phishing examples and legitimate sites to ensure that the rules fire appropriately and that the risk scoring calibration is reasonable.

Rule Firing Examples: Table 1 shows a few example URLs with the rules triggered and resulting risk scores as determined by our Prolog engine.

Table 1 - Rule Firing Examples

URL Example & Description	Triggered Rules (ID : description [score])	Total Score	Classification
`http://secure-verify.xyz/login`<br>*Phishing kit URL on* `.xyz`	5: Suspicious TLD (.xyz) [+200] <br>8: Numeric domain ratio (0%) [+0] <br>9: Multiple hyphens (0) [+0] <br>20: Young domain + cred token (age stub)	**600**	**Phishing**

	[+250]* <br>25: Susp. TLD + no MX (assume no MX) [+150]		
`http://a.b.c.d.e.xyz/a/b/c/d/e/f`<br>*Highly obfuscated URL*	5: Suspicious TLD (.xyz) [+200] <br>7: >3 subdomains (5 subdoms) [+180] <br>26: Subdomain + deep path combo [+150] <br>27: Deep path (6 levels) [+80]	**610**	**Phishing**
`http://192.168.1.45/admin`<br>*IP address as host*	1: IP address used as URL host [+300]	**300**	Suspicious (borderline)
`https://very-long-authentication-domain.com/`<br>*Legitimate but long domain*	6: Long domain (>30 chars) [+150]	**150**	Safe (low risk)

Table 1: Sample outcomes from the Prolog rule-based engine. Asterisks () indicate rules that are partially implemented or using stub data (e.g., domain age was not actually looked up, so Rule 20 contributes a default value). In the first example, multiple rules fire on a phishing URL that has a combination of suspicious traits (an `.xyz` TLD, a likely missing MX record, and a "secure-verify" credential keyword path on a presumably new domain), pushing the score to 600 which is above the 500 threshold for **Phishing**. The second example similarly stacks many indicators, resulting in a definite phishing classification. The third example (an IP address URL) triggers a single high-weight rule (using an IP as host is very indicative of malicious intent) reaching 300 points—below the phishing threshold but clearly not normal, so it's flagged as **Suspicious**. The last example is a benign case where only a low-severity rule triggered (long domain name), yielding 150 points which would be interpreted as **Safe** since no other problems were found.*

These examples illustrate the system's interpretability: for each URL, the security analyst can see *which* rules fired and contributed to the score. For instance, the first URL's analysis output would explicitly list the rules (with descriptions) that fired: *"Rule 5: Suspicious TLD (.xyz) → +200", "Rule 20: Domain age ≤30 days and credential token in URL → +250", "Rule 25: Suspicious TLD with no mail records → +150"*, totaling 600 (phishing). If any of these were false positives, the rule descriptions themselves direct attention to the feature—e.g., if we mistakenly flagged a domain's TLD as suspicious, an analyst can immediately see that was the reason and potentially adjust the allowlist or rule weight.

False Positives/Negatives Consideration: The rule-based approach, by its nature, may produce false positives if benign URLs share characteristics with phishing URLs. For example, a very new startup's website might be on a new domain with a generic TLD and lack certain DNS records, potentially triggering some rules. To mitigate this, we plan to incorporate contextual allowlists (like known good domains, as mentioned with Rule 33 placeholder) and tune thresholds. So far in testing, obviously legitimate domains (e.g., well-known bank or e-commerce domains) score near 0 due to our brand whitelist in the code (we include known safe domains to avoid penalizing them for, say, length). Conversely, the system could miss some phishing URLs if they don't trigger enough rules (false negatives). For instance, a phishing URL on a compromised legitimate domain might not have a suspicious TLD or other lexical giveaways. Such cases would score low in static analysis—but likely, our dynamic analysis (when fully implemented) would catch malicious content when the page is loaded (e.g., an unexpected login form on a hacked site).

Prototype Validation: We created a small test suite to validate each implemented rule in isolation using synthetic URLs crafted to meet the conditions. The tests confirm that: - The feature extraction predicates (for length, subdomains, tokens, etc.) compute correct values. - Each rule triggers when appropriate. For instance, a URL like `http://testlongdomainwithhyphens-123.com` triggers the long domain, hyphen, and numeric rules as expected. A known phishing URL from a dataset triggers a combination of rules yielding a high score, whereas a known safe URL triggers none or negligible rules. - Partial implementations (like the domain age check) currently return a default (e.g., age unknown yields Age=0 which prevents Rule 20 from firing fully). These did not impact classification in test cases because we treat unknown age as no score, which errs on side of not falsely penalizing.

Our results on example scenarios show that the Prolog rule engine approach is effective at identifying obviously malicious URLs and remains quiet on benign ones, with the added benefit of producing **explanations** for each decision. The scoring threshold can be tuned to adjust sensitivity (e.g., lowering the phishing threshold could catch more phish at risk of a few more false positives, if an environment demands it).

5. Conclusion and Future Work

We have presented a rule-based expert system for malicious URL detection in web applications, centered on a Prolog inference engine that unifies static URL analysis and dynamic content analysis. The system's architecture was designed for **explainability**, providing human-readable justifications for each detection via triggered heuristic rules. By leveraging a knowledge base of phishing indicators (such as suspicious domain attributes, DNS anomalies, and web page behaviors), the system achieves a layered defense: quick static analysis to catch blatant threats and an optional dynamic analysis to investigate suspicious cases further. This approach addresses the evolving challenge of phishing, where attackers continually find ways to bypass purely static or purely machine-learned detectors. Our Prolog engine's decisions are transparent, aiding security teams in understanding and trustingly acting on alerts.

Contributions: This work demonstrates the viability of using *logical inference* (as opposed to black-box machine learning alone) for a modern cybersecurity problem. We implemented a working proof-of-concept where the Prolog rule engine effectively identifies phishing URLs by reasoning over facts extracted from the URL and associated web content. The comparative analysis of features and integration of external intelligence within the rule base underscore that expert knowledge can be encoded to cover a broad range of phishing tactics. Moreover, by producing explanations like "Flagged due to suspicious domain age and login form", the system can enhance user trust and facilitate faster incident response. Notably, our use of Prolog allowed for rapid prototyping of rules and easy adjustments — adding a new rule is straightforward, which is essential given the adaptive nature of phishing threats.

Limitations: The current system is a prototype and has not been tested on large-scale, real-world URL datasets in this paper. Its effectiveness thus far has been illustrated on example scenarios and a limited test set. As a purely rule-based system, it may not catch attacks that fall outside the scope of its rules (e.g., novel phishing techniques that do not trigger any existing heuristic). Maintenance of the rule base is required as phishing tactics evolve. Additionally, the dynamic analysis portion is in progress; while the framework supports it, fully implementing and tuning the dynamic rules (and ensuring the headless browser analysis is safe and efficient) remains future work. Performance is another consideration – Prolog inference is very fast for our scale of rules, but the bottlenecks lie in network calls (DNS lookups, page fetches) which we mitigate by caching where possible.

Future Work: There are several avenues to enhance this system:

1. **Complete Dynamic Feature Integration:** Implement all planned dynamic analysis rules. This includes using the headless browser to detect phishing page elements like bogus login forms, malicious JavaScript, and implementing the rules we outlined (e.g., external form actions, multiple redirects, DOM modification events). We will draw on known dynamic indicators from research [12][14] to expand our rule set.

2. **Data Integration (WHOIS, DNSSEC, etc.):** Improve the static analysis by integrating domain age lookup via WHOIS/RDAP properly (so Rule 20 and similar can function with real data). Similarly, implement checks for SPF/DMARC properly by querying DNS TXT records, and integrate certificate transparency logs monitoring for newly-obtained certificates (for early phishing domain detection as per [21]). These will allow us to activate rules 21–24, 28–32 which are placeholders for more advanced DNS/PKI anomalies.

3. **Threat Intelligence Feeds:** Incorporate real-time threat intelligence (blacklists like PhishTank, Google Safe Browsing API, etc.). A low-effort integration would be to check the

URL/domain against such feeds and if found, either immediately classify as phishing or assign a very high score. Conversely, maintain an internal allowlist of known good domains (like major banks, popular services) to reduce false positives – this could implement the allowlist dampening rule (Rule 33) by subtracting points or overriding classification for those domains.

4. **Machine Learning Hybrid:** While our approach is primarily knowledge-based, we can explore a hybrid model where a machine learning classifier operates in parallel to the Prolog engine. The ML model (perhaps a lightweight gradient boosting model using the same features) could provide a second opinion or catch complex patterns. The Prolog engine's output could be combined with the ML output for a final decision – an approach that might increase detection coverage. Crucially, the rule-based explanations would still be provided, preserving interpretability, with the ML primarily to reduce misses. Ensuring this ensemble's decisions remain explainable will be a challenge to address.

5. **User Interface and Analyst Feedback:** Enhance the front-end to allow analysts to give feedback on results (e.g., mark false positives/negatives). This feedback could be used to adjust rule weights or add new rules. An interactive rule tuning interface could even be envisaged, where analysts can simulate how adding a rule or changing a threshold would have affected past detections.

6. **Performance and Scalability:** For deployment in a real-time environment (such as an email filtering gateway or web proxy), we need to ensure the system can handle high throughput. Caching DNS and WHOIS results and perhaps performing dynamic analysis asynchronously (only for a small subset of URLs) will be important. We will also investigate compiling the Prolog rules or using tabling in Prolog for any expensive logic, though currently performance is acceptable given the small rule set.

7. **Evaluation:** Finally, a comprehensive evaluation on a benchmark dataset (e.g., URLs from PhishTank over time combined with benign URLs from Alexa top sites) is planned. We will measure detection accuracy, false positive rate, and compare against baseline approaches like a random forest or a deep learning URL classifier. This will quantify the trade-offs of our interpretable approach. We expect, in line with Basnet *et al.* [23], that our rule-based system will achieve competitive accuracy with the added benefit of clear explanations, confirming the value of expert-driven design in this domain.

This work illustrates that *logic programming and expert systems remain highly relevant in cybersecurity*. By capturing expert knowledge of phishing behavior in Prolog rules, we created a detection system that is not only effective in catching attacks but also inherently interpretable. It serves as a foundation that can be iteratively improved with more rules and data sources. In an era where AI in security often leans towards complex machine learning models, our approach provides a complementary perspective: leveraging human-understandable rules to stay ahead of phishing threats in a rapidly changing landscape, while keeping the human analyst in the loop through clear reasoning of the AI's decisions.

References

[1] ENISA – *ENISA Threat Landscape 2024*. European Union Agency for Cybersecurity, Publications Office of the European Union, Luxembourg, 2024.

[2] ENISA – *ENISA Threat Landscape 2025 – Booklet*. European Union Agency for Cybersecurity, Luxembourg, Oct. 2025.

[3] World Economic Forum – *Global Cybersecurity Outlook 2025*. World Economic Forum, 2025.

[4] Patil, D. R., & Dhage, S. N. (2019). **Framework for anti-phishing: URL feature extraction and classification**. *Proc. of the Int. Conference on Information Security and Privacy (ICISP)*, pp. 105–112.

[5] Hranický, R., Horák, A., Polišenský, J., Ondryáš, O., Jeřábek, K., & Ryšavý, O. (2024). **Spotting the Hook: Leveraging Domain Data for Advanced Phishing Detection**. *20th Intl. Conference on Network and Service Management (CNSM 2024)*, pp. 1–7. DOI: 10.23919/CNSM62983.2024.10814617

[6] Aravindhan, K., Sethumadhavan, M., & Krishnan, P. (2019). **Enhanced phishing detection through comprehensive URL analysis and machine learning techniques**. *Journal of Information Security and Applications*, 45, 123–135.

[7] *dnstwist – Domain Name Permutation Engine for Phishing Detection*. URL: https://dnstwist.it/ (accessed 2025).

[8] Bayer, J., Maroofi, S., Hureau, O., Duda, A., & Korczynski, M. (2023). **Building a Resilient Domain Whitelist to Enhance Phishing Blacklist Accuracy**. *APWG eCrime 2023*, pp. 1–14. DOI: 10.1109/eCrime61234.2023.10485549

[9] Ishida, Y., Hanada, M., Waseda, A., & Kim, M. W. (2023). **Analysis of DNS Graph of Phishing Websites Using Digital Certificates**. *25th Intl. Conference on Advanced Communication Technology (ICACT 2023)*, pp. 174–179. DOI: 10.23919/ICACT56868.2023.10079566

[10] Nishitha, U., Kumar, S., & Reddy, P. (2023). **Phishing Detection Using Machine Learning Techniques**. *IEEE Intl. Conference on Computing and Communications Technologies (ICCCT 2023)*, pp. 234–241.

[11] Hranický, R., Horák, A., Polišenský, J., Jeřábek, K., & Ryšavý, O. (2024). **Unmasking the Phishermen: Phishing Domain Detection with Machine Learning and Multi-Source Intelligence**. *IEEE/IFIP Network Operations and Management Symposium (NOMS 2024)*, pp. 1–5. DOI: 10.1109/NOMS59830.2024.10575573

[12] O'Mara, A., Alsmadi, I., Aleroud, A., & Alharthi, D. (2023). **Phishing Detection Based on Webpage Content: Static and Dynamic Analysis**. *3rd Intelligent Cybersecurity Conference (ICSC 2023)*, pp. 39–45.

[13] *Puppeteer – Headless Chrome Node.js API*. URL: https://pptr.dev/ (accessed 2025).

[14] Rose, S., Johnson, M., & Chen, L. (2022). **Real-time phishing detection through browser extension monitoring of DOM and network activity**. *Computers & Security*, 118, 102–115.

[15] Bajaj, K., Pattabiraman, K., & Mesbah, A. (2015). **LED: Program Analysis Tool for Synthesizing Web Element Locators**. *30th IEEE/ACM Intl. Conference on Automated Software Engineering (ASE 2015)*, pp. 848–851.

[16] Omolara, A. E., Jantan, A., Abiodun, O. I., et al. (2025). **DaE^2: Unmasking malicious URLs by leveraging diverse and efficient ensemble machine learning**. *Computers & Security*, 147, 103–118.

[17] Abad, S., Rahman, M., & Thompson, J. (2023). **Classification of Malicious URLs Using Machine Learning**. *Pacific Journal of Cybersecurity Research*, 8(2), 45–62.

[18] Alsmadi, I., O'Mara, A., & AlEroud, A. (2021). **Generative Adversarial Analysis of Phishing Attacks on Static and Dynamic Content of Webpages**. *IEEE Intl. Conf. on Parallel & Distributed Processing with Applications (ISPA-BDCloud-SocialCom-SustainCom 2021)*, pp. 1657–1662.

[19] Çelik, L., Yilmaz, E., & Hassan, A. (2025). **Enhancing Phishing Detection in Financial Systems through Natural Language Processing**. arXiv preprint arXiv:2507.04426.

[20] Hranický, R., Bujlow, T., & Čejka, T. (2024). **Multi-source threat intelligence integration for robust phishing detection**. *Journal of Network Security*, 28(3), 189–205.

[21] Alsabah, M., Nabeel, M., Boshmaf, Y., & Choo, K.-K. R. (2022). **Content-Agnostic Detection of Phishing Domains using Certificate Transparency and Passive DNS**. *Proc. of the 25th Intl. Symposium on Research in Attacks, Intrusions and Defenses (RAID 2022)*, ACM, DOI: 10.1145/3545948.3545958.

[22] Abyaa, Y., Hureau, O., Duda, A., & Korczynski, M. (2025). **LogoTrust: Leveraging BIMI to Build a Validated Dataset of Brands, Domain Names, and Logos**. *IEEE European Symposium on Security and Privacy Workshops (EuroS&PW 2025)*, pp. 132–137. DOI: 10.1109/EuroSPW67616.2025.00021

[23] Basnet, R. B., Sung, A. H., & Liu, Q. (2012). **Rule-Based Phishing Attack Detection**. *Proc. of the 4th Cybercrime and Trustworthy Computing Workshop (CTC)*, IEEE, pp. 1–6. [1]

[24] Sawon, M. S. H., Rahi, M. A., Islam, I. F. B., Uddin, M. J., Islam, J., & Biplob, M. B. (2025). **An Introduction to Rule-Based AI Agents for Detecting Phishing Emails**. *Preprints*, 2025, 202507.1067.v1.