Final Knowledge Acquisition Report

Challenge 1

Phishing Detection in Cybersecurity

Arsénio Ferraz - 1010137

Bruno Camarneiro - 1250422

César Vieira - 1241523

Gustavo Lima - 1221349

Rui Soares - 1221283

Contents

Abstract

Phishing is a prevalent cybersecurity threat, and this report details the development of an expert system that leverages the Drools rule engine to detect phishing attacks in real-time. The project's objectives, methodology, and outcomes are presented in an academic framework. A structured knowledge acquisition process was followed, combining literature review and collaboration with a cybersecurity expert to formalize domain knowledge into an interpretable set of rules. The technical implementation includes a multi-layered architecture in which extracted features from URLs, DNS records, redirect chains, and webpage content are evaluated by a Drools inference engine to produce a phishing risk score. The system's effectiveness was validated against real-world phishing scenarios and benign cases, demonstrating high accuracy and explainability. Key results, challenges faced (such as balancing false positives and performance constraints), and competencies gained are discussed. Finally, the report outlines future improvements and emphasizes how integrating expert knowledge with a rule-based AI approach can yield a robust, transparent phishing detection solution.

Objectives

The principal objective of this work is to design and validate an expert system capable of identifying malicious URLs and phishing webpages, using a rule-based approach for efficiency and explainability. This entails capturing the core characteristics of phishing attempts and encoding them into Drools rules, while ensuring the system remains adaptable to emerging attack techniques. The system is expected to provide transparent decision reasoning (to aid security analysts) and practical performance for real-time use. Specific objectives include:

- **Apply Drools Rule Engine:** Study and experiment with Drools (a production rule engine) for building a symbolic reasoning system in a cybersecurity context.
- **Hybrid Detection Approach:** Implement a detection pipeline that analyzes multiple aspects of a web link – including URL lexical features, DNS configuration, HTTP redirect behavior, and page DOM content – through human-readable rules.
- **Enterprise Integration:** Integrate the Drools-based engine with a Spring Boot application, exposing phishing detection logic via REST APIs suitable for enterprise deployment.
- **Knowledge Formalization:** Collaborate with a domain expert to elicit and formalize tacit cybersecurity knowledge (attack patterns, heuristics, thresholds) into a structured rule base.
- **Lifecycle Documentation:** Document the entire knowledge acquisition and system development lifecycle, covering knowledge elicitation, rule development, testing/validation, and deployment considerations.

- **Skill Development:** Develop competencies in knowledge representation and explainable AI, demonstrating how expert-driven rules can complement or substitute statistical models in critical domains like cybersecurity.

Methodology

Approach Overview: The knowledge acquisition methodology followed a structured and iterative path, ensuring both academic rigor and practical relevance. Initially, extensive background research was conducted, drawing on academic papers, industry reports, and technical documentation related to phishing detection. Notable sources included peer-reviewed studies on URL analysis and phishing (*IEEE, ACM, arXiv* databases), annual threat landscape reports (ENISA 2024–2025, World Economic Forum 2025), and documentation of threat intelligence services (OpenPhish, PhishTank, VirusTotal) and relevant frameworks (Drools/KIE, Spring Boot). These sources provided a foundation of state-of-the-art techniques and typical phishing indicators.

Expert Involvement: Complementing the literature review, a domain expert was engaged to provide real-world insights and validate the system's design. The expert, Mr. David Marques (Head of Cybersecurity at Grupo Nabeiro / Delta Cafés), has over 15 years of professional experience in cybersecurity, including incident response and phishing defense strategies. His expertise spans enterprise phishing mitigation, risk management, and security awareness training. During the project, the expert contributed by sharing real attack examples and advanced phishing tactics, advising on realistic detection thresholds, and highlighting operational challenges (such as acceptable false positive rates and integration friction with existing security tools). He emphasized the importance of explainability in rule outputs for audit and compliance and recommended regular rule updates to keep pace with evolving threats. This close collaboration ensured that the rule base encapsulates not only theoretical patterns but also practical, experience-driven knowledge.

Knowledge Acquisition Sessions: The process was organized into four sessions with the expert, each with specific goals and outcomes, progressively refining the rule base:

1. **Session 1 – Preparation & Framework Definition:** The team established a clear plan for knowledge capture and system design. A structured interview guide was developed to cover known phishing tactics and technical indicators (e.g., URL patterns, DNS anomalies, redirect behaviors, malicious HTML elements). Operational constraints for an enterprise environment were also identified (such as performance requirements and user impact considerations). *Outcome:* A prioritized list of phishing attack patterns and a validated set of detection heuristics to explore. This session set the stage by confirming that the proposed approach aligned with real-world conditions and pinpointing which features (e.g., presence of "@" in URLs, very young domains, etc.) would be most indicative of phishing.

2. **Session 2 – Expert Interview & Validation:** A detailed knowledge elicitation interview was conducted with the expert, focusing on deep technical insights and validating initial ideas. The discussion covered a range of modern phishing techniques: the use of URL shorteners and multi-hop redirect chains to mask malicious destinations, homograph attacks using internationalized domain names, attacker abuse of content delivery networks (CDNs) and cloud hosting to appear legitimate, as well as domain generation algorithms (DGAs) that evade static blacklists. The expert helped calibrate detection thresholds (e.g., how many redirects are inherently suspicious, what domain age should be considered "too new") and stressed the need to balance security with usability (to avoid too many false alarms). *Key findings:* Multi-layered detection is necessary – no single indicator is sufficient. The expert recommended combining URL, DNS, and DOM analysis for a comprehensive approach, incorporating context (e.g., internal vs. external domains), maintaining allow/deny lists for special cases, and ensuring that each alert provides a clear explanation for analysts.

3. **Session 3 – Rule Consolidation & Technical Implementation:** In this phase, insights from the expert were translated into concrete rules and an initial system architecture. The team formalized **50 Drools rules** across four categories (URL, DNS, DOM, and Redirect analysis) covering the breadth of identified phishing indicators. A fact model (`Evidences` class) was designed to hold all features extracted from an analyzed link, and a scoring mechanism was outlined. The system architecture and integration plan (Drools engine within a Spring Boot application) were also established. Key achievements in this session included implementing the Drools rule engine with all rule files loaded, creating a parameterized scoring system (with adjustable weights for each rule category via an external config file), and integrating external threat intelligence lookups for additional data (e.g., checking a URL against known phishing databases). *Architecture decisions:* The rule base was organized modularly into separate files – **`url-rules.drl`**, **`dns-rules.drl`**, **`dom-rules.drl`**, and **`redirect-chain-rules.drl`** – for clarity and ease of maintenance. A YML configuration was used to store rule weights and risk score thresholds, allowing tuning without code changes. The Drools engine was configured to instantiate a new rule session per analysis request to ensure thread safety and scalability. Comprehensive logging was added to trace rule activations for debugging and to aid explainability.

4. **Session 4 – Testing & Iterative Refinement:** The final phase involved rigorous validation of the system against test cases and further refinement based on results and additional expert feedback. A suite of unit tests was developed to verify each rule in isolation (for example, ensuring that a URL containing an "@" character indeed triggers the corresponding rule and adds the expected score). Complex **attack scenarios provided by the expert were used for integration testing**, simulating real phishing attempts end-to-end through the system. Performance tests were also conducted to measure analysis latency and throughput, important for real-time

deployment. *Refinement activities:* The team adjusted rule thresholds and weights after reviewing test outcomes (e.g., lowering sensitivity for certain rules that caused false positives). Some rules were optimized or combined to reduce redundancy and improve clarity. The expert's feedback was integral during this stage – for instance, verifying that the output explanations were understandable and that the system's overall behavior aligned with how a human analyst would judge the same scenarios. By the end of this session, the rule base and system were iteratively honed to achieve a balance between catching true phishing attacks and avoiding false alarms, ready for final evaluation.

Acquired Knowledge Representation

The next figures 1, 2, 3, 4, 5, 6 and 7 represent the rules diagram, divided into four parts: URL Analysis, DNS Analysis, Redirect Analysis and DOM Analysis.

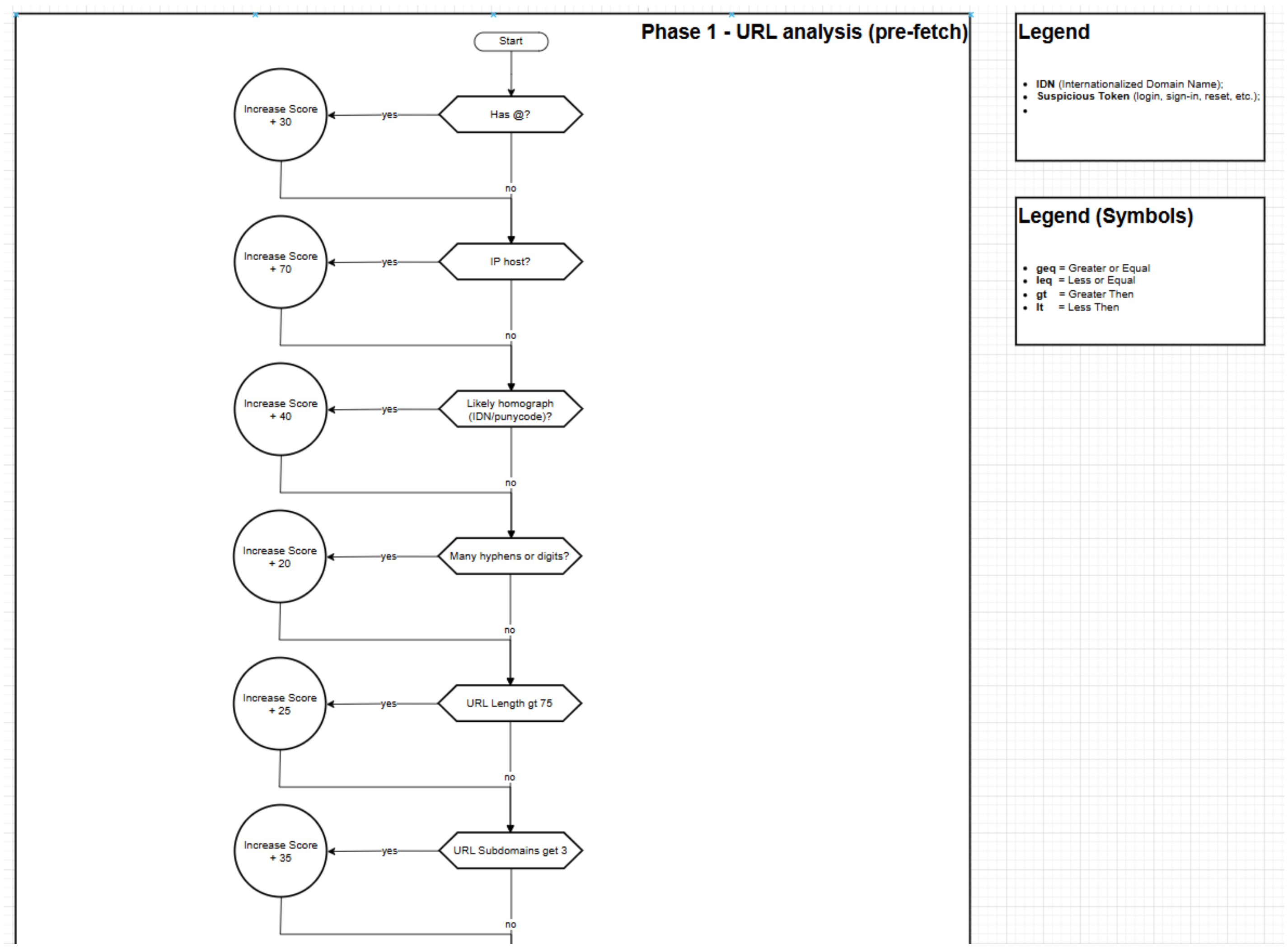

Figure 1 - URL Analysis

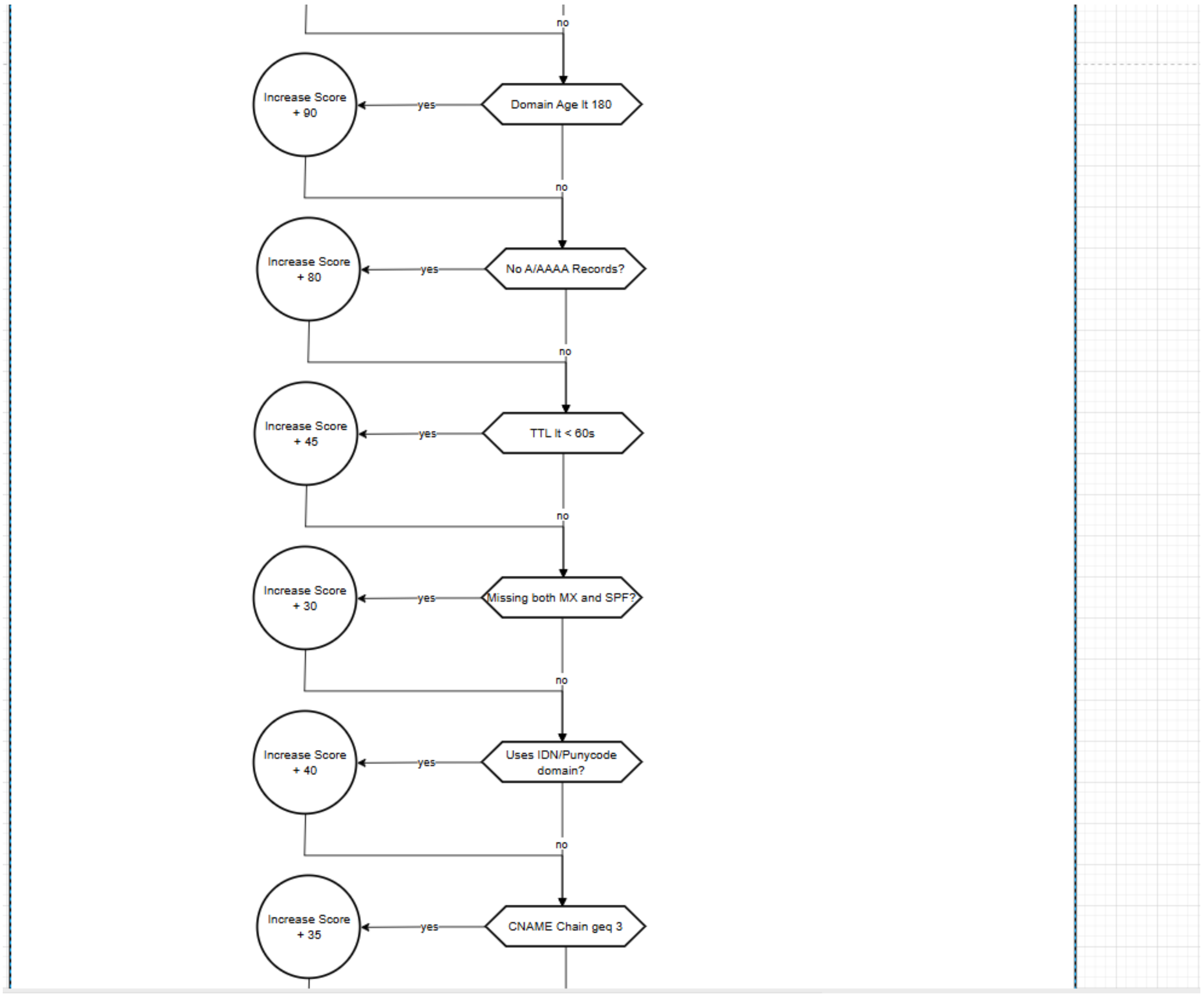

Figure 2 - DNS Analysis

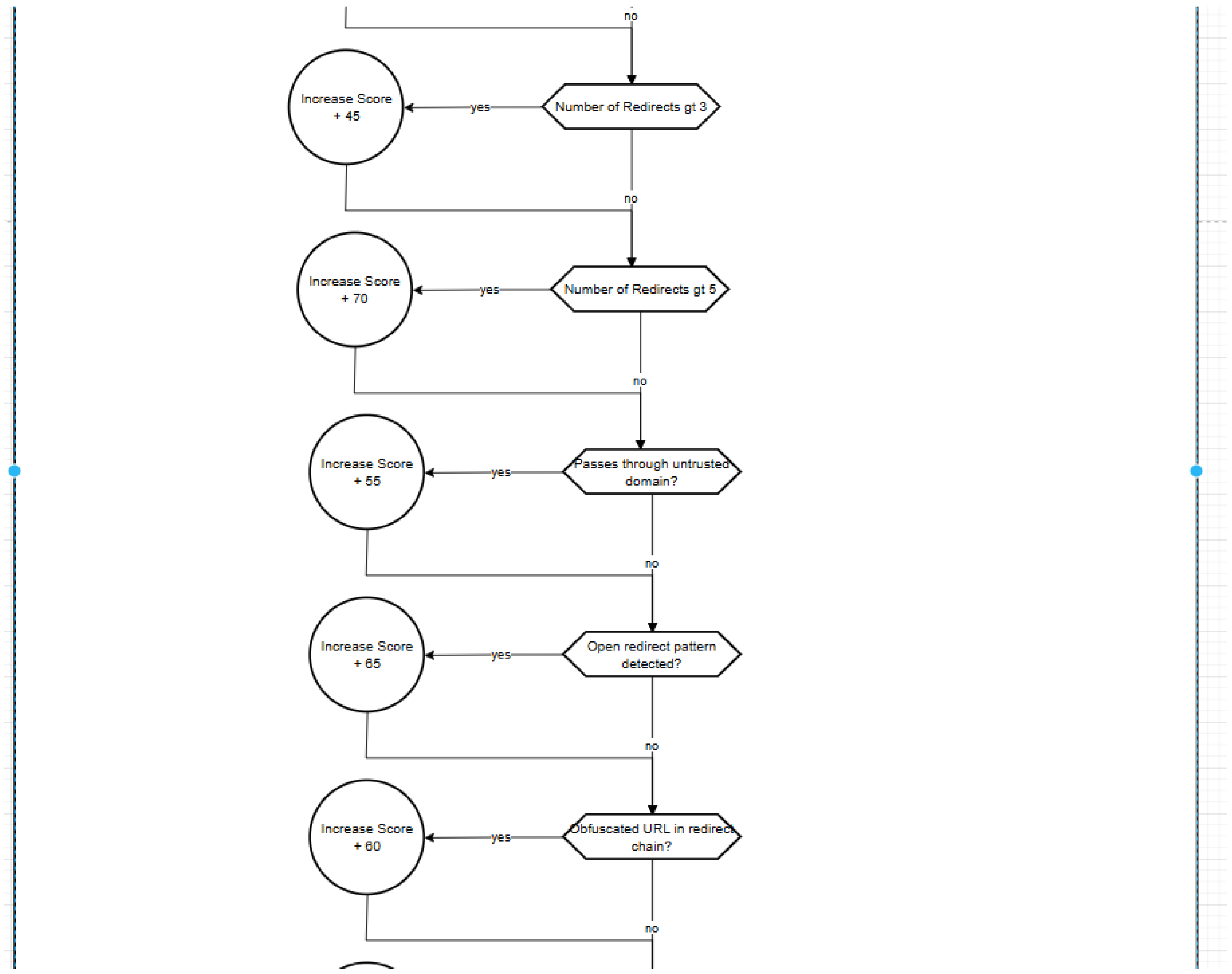

Figure 3 - Redirect Analysis - Part 1

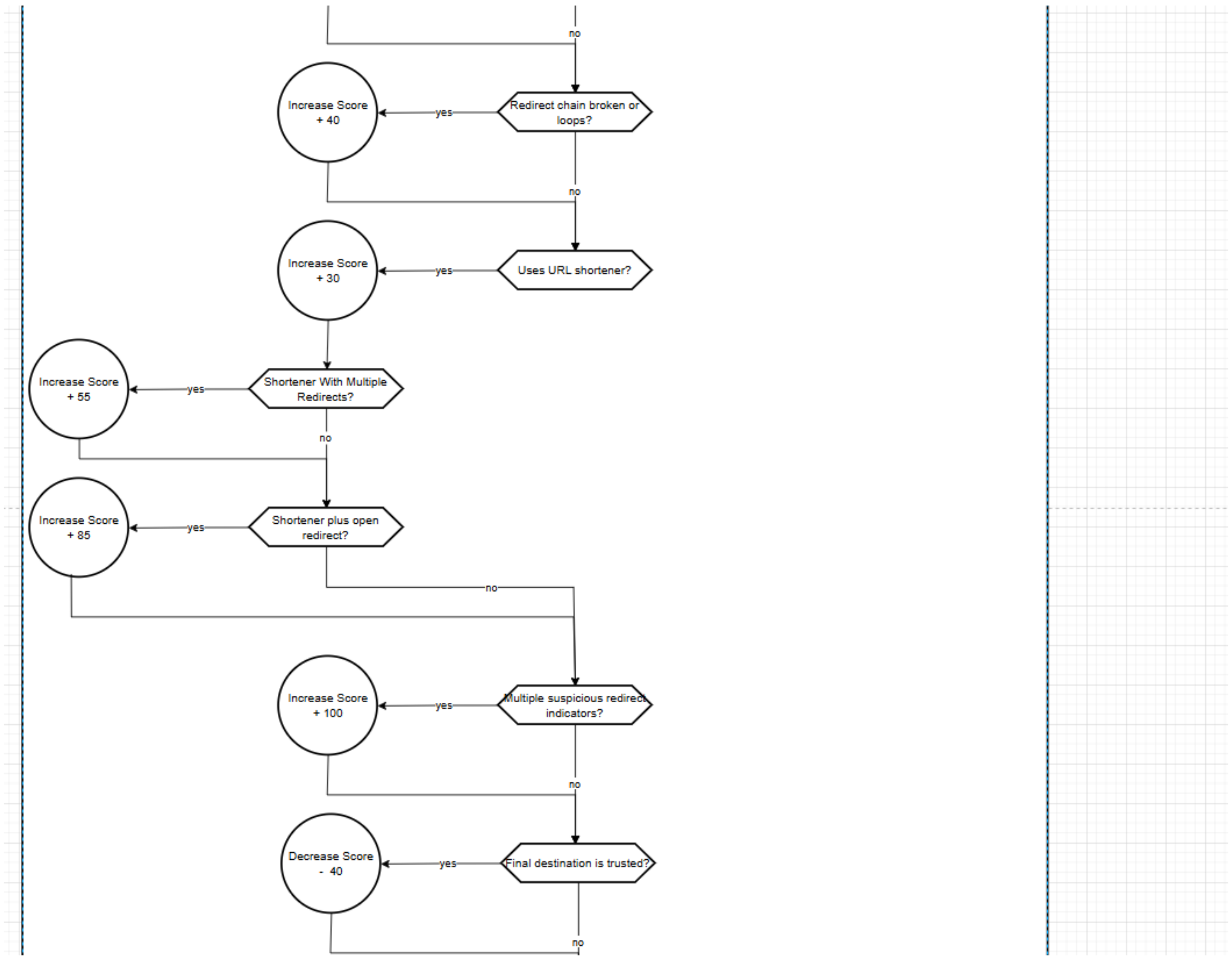

Figure 4 - Redirect Analysis - Part 2

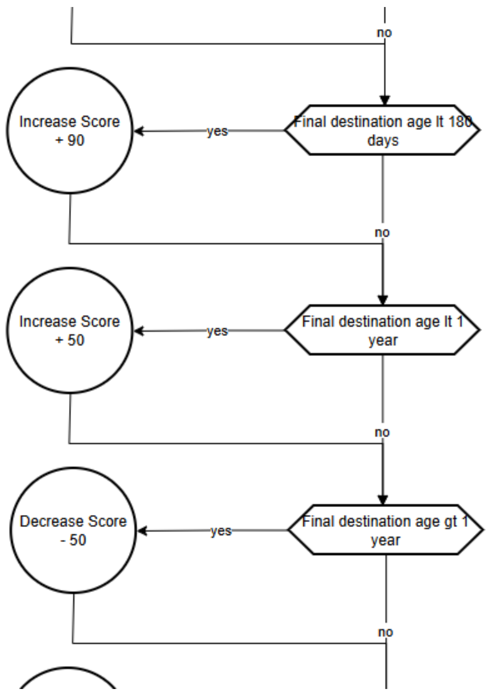
no
Final destination age lt 180 days
yes
Increase Score + 90
no
Final destination age lt 1 year
yes
Increase Score + 50
no
Final destination age gt 1 year
yes
Decrease Score - 50
no

Figure 5 - Redirect Analysis - Part 3

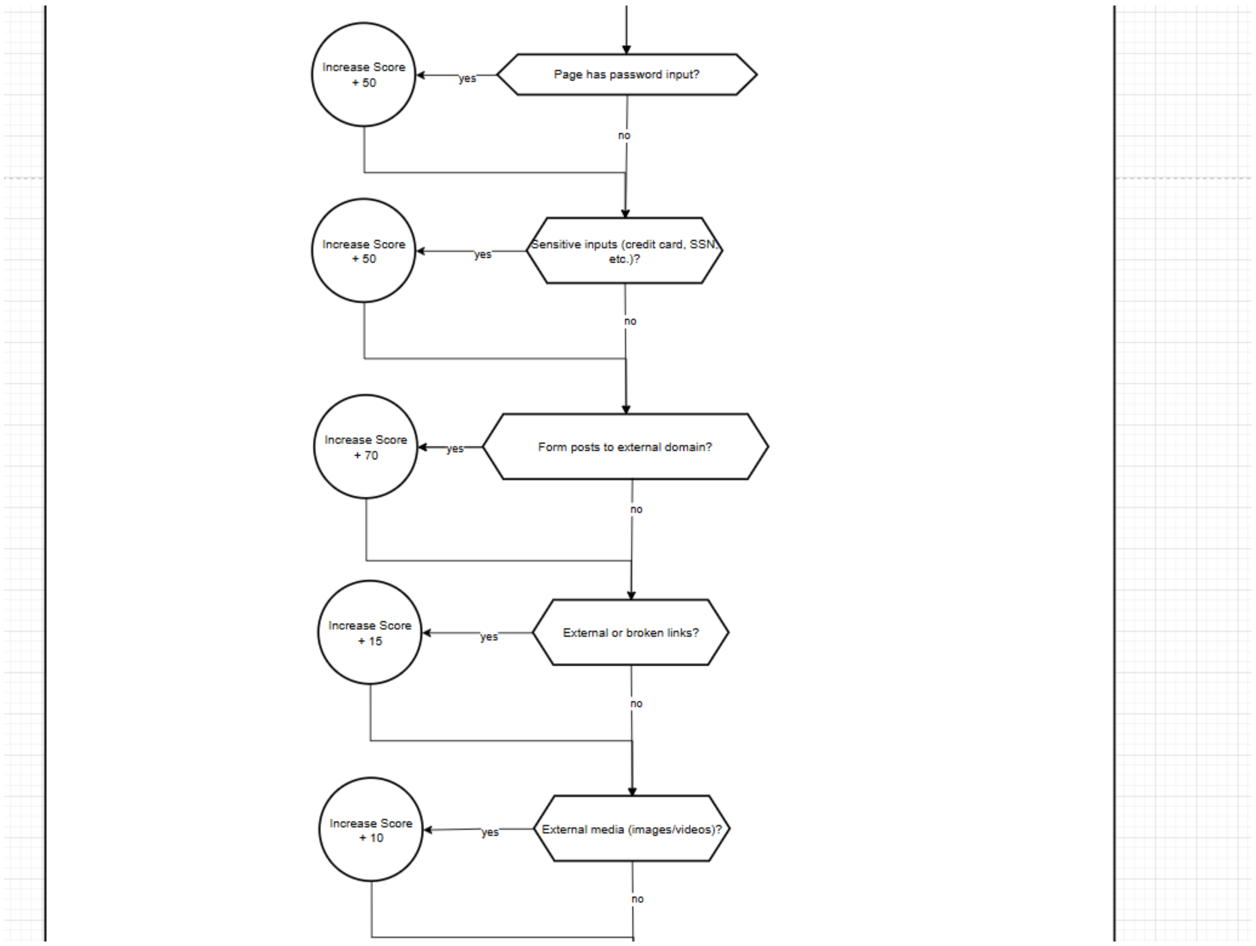

Figure 6 – DOM Analysis: Part 1

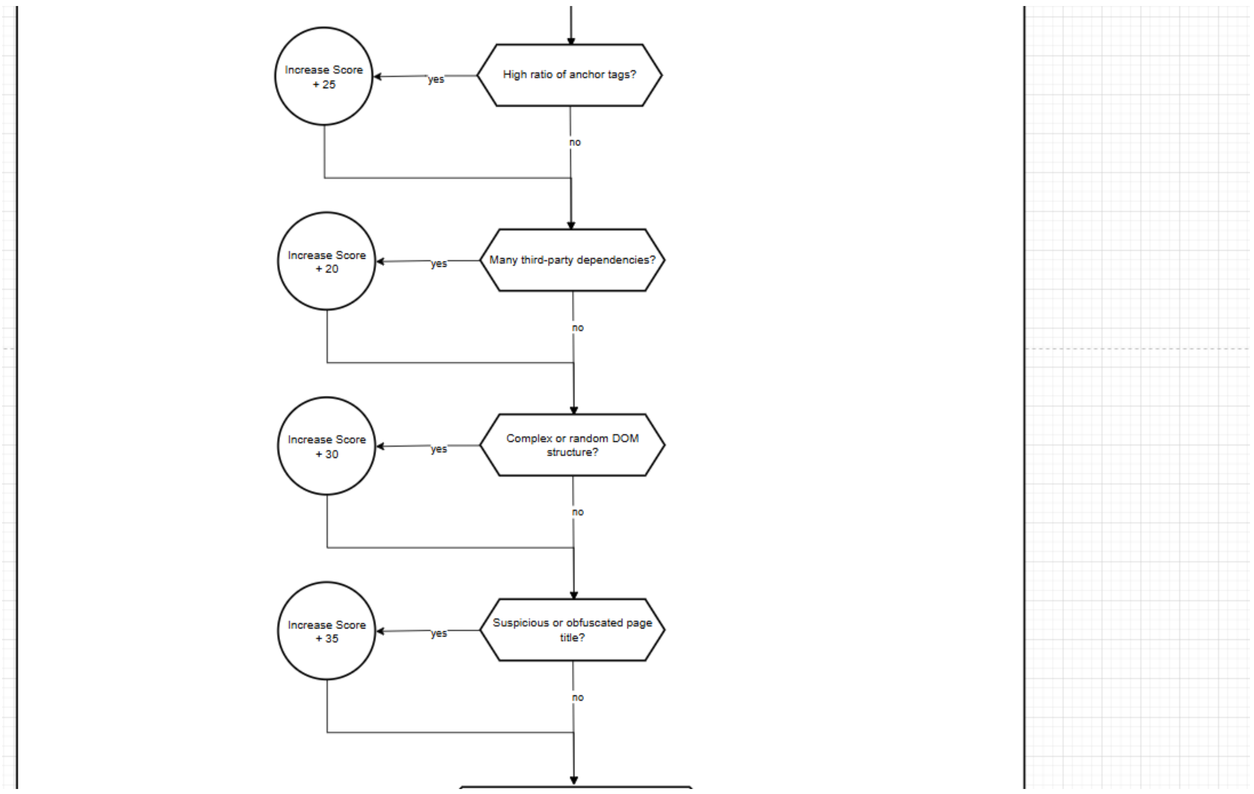
High ratio of anchor tags?
yes
Increase Score + 25
no
Many third-party dependencies?
yes
Increase Score + 20
no
Complex or random DOM structure?
yes
Increase Score + 30
no
Suspicious or obfuscated page title?
yes
Increase Score + 35
no

Figure 7 - DOM Analysis: Part 2

The conclusion is evaluated in the next figure 8.

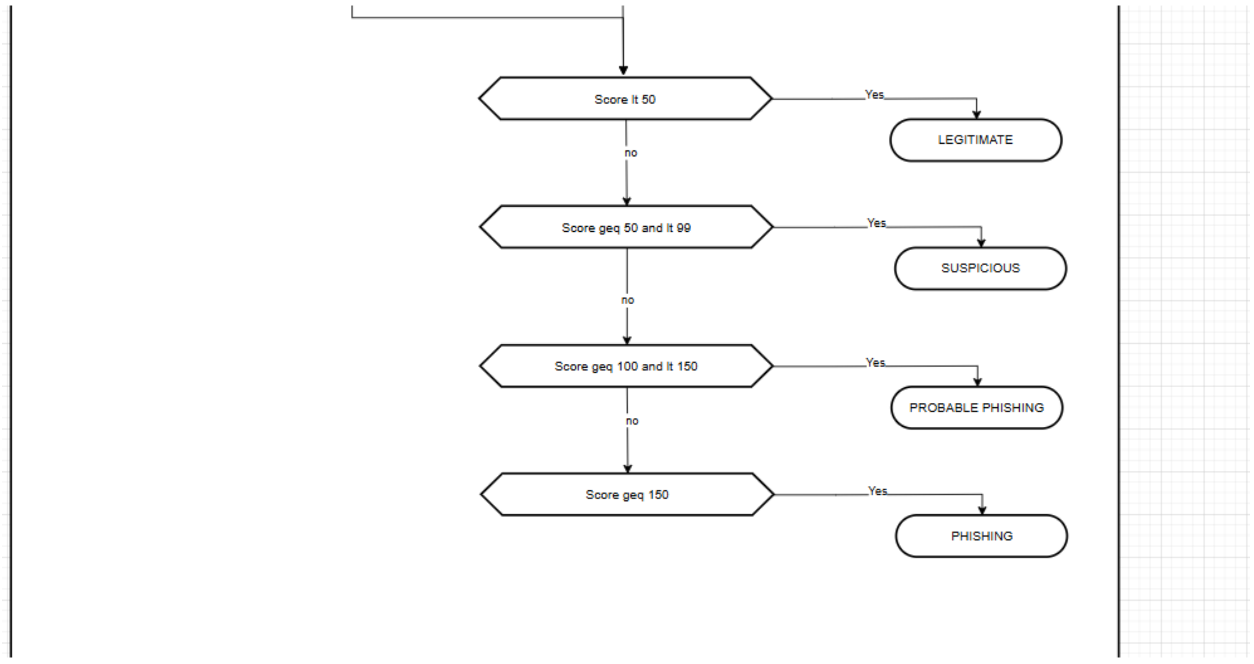

Figure 8 - Conclusion

Technical Implementation

System Architecture: The phishing detection system was implemented as a multi-layered pipeline to systematically analyze each input URL. The architecture can be outlined in five main stages:

1. **Feature Extraction Layer:** Upon receiving a suspicious URL, the system first gathers a comprehensive set of features:
2. **URL Analysis Service:** Examines the URL string for lexical features (e.g., presence of an "@" character which can obscure the real domain, use of an IP address instead of a hostname, unusually long length, multiple subdomains, known suspicious TLDs or hosting domains, etc.). It also computes heuristics like the count of special characters (hyphens, digits) and checks for encoded or obfuscated segments.
3. **DNS Analysis Service:** Performs DNS lookups and WHOIS queries to collect domain-related information. This includes checking if the domain resolves to an IP address (A/AAAA records), the minimum TTL value (a very low TTL might indicate fast-flux hosting), existence of mail server records (MX) and email security records (SPF, DKIM) which legitimate domains usually have, the length of any CNAME chains (multiple CNAME redirections could hint at cloaking), and the domain's age (e.g., via WHOIS creation date) to see if it is newly registered.
4. **Redirect Chain Service:** If the URL uses HTTP/HTTPS, this component follows any HTTP redirections (up to a safe limit) to uncover intermediate and final destinations. It records the full redirect chain, notes if any known URL shortening services were used, and flags open redirect parameters or other redirection anomalies. The number of redirects and diversity of domains involved are also captured.
5. **Page Analysis Service:** For final destinations (especially if HTML content is reachable), a lightweight headless browser or HTML parser is used to inspect the page's DOM. The service looks for telltale signs of phishing on the page itself: presence of password input fields or credit card forms, forms that submit to external domains, a high ratio of external resource links, inclusion of suspicious scripts, overall complexity or entropy of the page's content, and any visual cues of brand impersonation (such as use of logos or titles that mismatch the domain).

6. **Fact Model Layer:** All extracted features from the previous layer are aggregated into a unified fact object called `Evidences`. This object is a structured data model with typed fields representing each piece of evidence (e.g., `hasAtChar`, `domainAgeDays`, `redirectChainLength`, `hasPasswordField`, etc.). Using a single fact object simplifies rule writing – each Drools rule can access the relevant attributes of `Evidences`. The design of the fact model took into account defensive programming (e.g., fields are initialized to safe defaults or null to avoid null-pointer issues in rules) and clarity, so that the meaning of each field is clear to a domain expert reviewing the rules. Before inference, the populated `Evidences` object and an empty `ScoreCard` object are inserted into the Drools working memory.

7. **Inference Engine Layer (Drools):** The core of the system is the Drools rule engine which houses the phishing detection logic in the form of if-then rules. A KIE container is configured to load the rule sets (declared in the `.drl` files mentioned above) on application startup, and a stateless KIE session is used for each analysis. When the session is fired, Drools evaluates all rules against the inserted facts (`Evidences` and global configurations). The inference is forward-chaining: as soon as a rule's conditions (Left-Hand Side) match the facts, its actions (Right-Hand Side) execute. The rules may insert additional facts or update the score, and Drools will automatically re-evaluate any other rules that depend on changed facts, until no more rules can fire. A global helper object `RuleWeightsConfig` is provided to the session, which encapsulates the scoring weights for different rules or categories (read from the external config). This allows rules to remain static while letting an administrator adjust the contribution of each evidence type to the overall score.

8. **Scoring & Explanation Layer:** As rules fire, they interact with a `ScoreCard` object. The `ScoreCard` accumulates points every time a rule identifies a phishing indicator. Each rule contributes a certain weighted score (e.g., the presence of a password field in a page might add a moderate score, whereas a known malicious domain might add a high score). The `ScoreCard` also keeps track of which rules have fired and can store details for explanation. For example, if a rule flags that the URL is using a URL shortener, the details might include the specific shortening service domain that was observed. After all applicable rules have executed, the `ScoreCard` computes a final risk score (typically on a 0 to 100 scale) and a classification (e.g., "Legitimate", "Suspicious", or "Phishing") based on threshold ranges. In our implementation, scores below 50 are considered legitimate, 50–79 suspicious, and 80+ likely phishing. The triggered rules' names and explanations are collected to produce a human-readable justification for the verdict.

9. **API Layer:** The system is packaged as a Spring Boot application providing RESTful API endpoints for analysis requests. This layer takes care of receiving an HTTP request containing a URL to analyze, orchestrating the feature extraction and Drools inference process, and returning the results in JSON format. The output includes the final determination (and score) as well as the list of triggered indicators and their explanations, so that a client application (or a security analyst) can understand why a URL was flagged. The API layer also handles integration with external services: for example, it may call out to the VirusTotal API or an internal threat intelligence database as part of the feature extraction phase for additional context on a domain or URL. This design ensures the core rule engine remains decoupled from external systems, with the API layer acting as a mediator and integrator.

Overall, this architecture cleanly separates data gathering, inference, scoring, and interface concerns. It allows each component to be developed and tested in isolation (e.g., ensuring the feature extractors correctly populate the `Evidences` object, or that the rule engine fires the expected rules given a certain fact input). The modular design also simplifies future enhancements, such as adding new feature extractors or rule sets without affecting unrelated parts of the system.

Rule Engine and Knowledge Base: The Drools rule engine was chosen for its expressive power in encoding expert knowledge and its forward-chaining execution suitable for combinatorial rule evaluation. A Drools **rule** in our system follows a standard structure:

```
rule "Rule Name"
when
    // Conditions on facts (LHS)
    $fact : Evidences( someCondition == true )
    $sc : ScoreCard()
then
    // Actions (RHS)
    // e.g., add score, create log entry, modify fact, etc.
    $sc.addScore( ... );
    // possibly add details or modify $fact
end
```

Each rule specifies a condition on the `Evidences` fact – if the condition is met, the rule's action will contribute to the `ScoreCard`. The rules are designed to be **declarative**, focusing on *what* pattern to detect rather than *how* to detect it procedurally.

The knowledge base is organized into four categories of rules, reflecting different dimensions of phishing detection:

- **URL Analysis Rules (8 rules):** These rules inspect the URL string itself for anomalies. They detect cases such as the presence of an **"@" character** in the URL (often used to mask the true domain by preceding it with a fake login part), the use of an **IP address** in lieu of a domain name (suspicious for public links), **homograph attacks** or Punycode usage in domains (international characters resembling trusted brands), **excessive hyphens or numeric characters** in the URL (often a sign of auto-generated or obfuscated URLs), unusually **long URL length** (which might indicate parameter stuffing or hiding malicious components far to the right), an **excessive number of subdomains** (deeply nested subdomains can be used to mislead users about the true domain), very **recent domain registration** (e.g., domain age < 6 months, as phishers frequently register new throwaway domains), and the use of certain **free hosting providers** (domains like *pages.dev, vercel.app, netlify.app, github.io,* etc., which, while legitimate services, are commonly abused for phishing pages). Each of these patterns, when detected, results in a rule firing and adding to the score.

- **DNS Analysis Rules (6 rules):** These rules evaluate properties of the domain's DNS setup and related metadata. For example, if a domain **has no A or AAAA record** (meaning it doesn't resolve to any IP address), that is a strong indicator of a non-legitimate domain and triggers a rule. A **very low DNS TTL** (Time-To-Live) on returned records is flagged, since phishers may use fast-flux networks with rapidly changing

IPs. The absence of **mail server (MX) records and sender policy (SPF/DKIM) records** is noted – legitimate domains that serve users typically have these, so a domain used only for phishing might not bother with email configurations. The presence of **internationalized domain names (IDN)** in DNS (punycode prefixes like `xn--`) is caught as a duplication of the homograph check from the URL layer, ensuring such domains are marked. A **long CNAME chain** (multiple CNAME redirections in DNS) could indicate an attempt to hide the final host and is thus considered suspicious. Additionally, extremely **young domain age** (e.g., < 30 days old) as identified via WHOIS is flagged here as well (this complements the URL layer's "very new domain" rule, using the most up-to-date registration info).

- **DOM Analysis Rules (9 rules):** These rules activate when the content of a fetched page exhibits traits typical of phishing. If a **password field is present** in the HTML (especially in contexts where it's unexpected, like a site that shouldn't ask for login), a rule increases the risk score. More generally, any **sensitive input fields** (such as credit card number fields or other personal data fields) found will trigger an alert, with the count of such fields recorded. **External form actions** are checked: if a form in the page submits to a different domain or a null destination, it could be a sign of data being sent to an attacker's server. The presence of many **external or null hyperlinks** (links that either point outside the site or have no valid target) can indicate the page is stitching together resources or directing users off-site (common in phishing pages to lead users to malicious payloads). Inclusion of **external media resources** (images, scripts, iframes from external sources) is noted – while not inherently malicious, a phishing page often hot-links images or scripts from elsewhere (or uses resource files from the legitimate site to appear authentic). Some quantitative measures are also used: a **high link-to-text ratio** (very high proportion of hyperlinks or interactive elements in the DOM) can indicate a suspicious page structure, as can a high **dependent request ratio** (lots of third-party resource loads). We also compute **DOM entropy** – if the HTML/JS content appears highly obfuscated or minified (e.g., long random strings, high complexity), that raises suspicion. Finally, if the **page title looks obfuscated or misleading** (for instance, containing punycode or inconsistent branding), a rule will flag it.

- **Redirect Chain Rules (11 rules):** These rules analyze the sequence of HTTP redirects encountered. An **excessive number of redirects** (more than 3 hops) is deemed suspicious, as phishers use lengthy chains to evade filters. Even more so, a **very excessive count** (>5) triggers an even stronger response. If the redirect chain passes through any **untrusted domains** (domains not known or expected for the context) *in combination* with other warning signs (such as a new final domain, or the presence of open redirect parameters), a rule flags this compound condition as particularly risky. Detection of an **open redirect pattern** (URLs in the chain containing parameters like `?redirect=` or `&url=`) will fire a rule, since attackers exploit open redirect vulnerabilities in trusted sites to bounce victims through. If any URL in the chain appears **obfuscated** (e.g., containing encoded segments or being extremely long and complex), that's noted as well. A **broken redirect chain** (loops or errors in following it) is flagged, as it could mean the analysis was thwarted by intentional evasion. Specific to phishing, the presence of a **URL shortener** service in the chain (like bit.ly, tinyurl, etc.) triggers a rule, and if a shortener is combined with multiple further redirects, another rule views that combination as a high-risk indicator. On the positive side, the ruleset includes a check for **trusted final destinations**: if all the redirects ultimately lead to a known safe domain (e.g., a well-known company's login page or a major site like Google/Microsoft), a rule can *reduce* the score or nullify some earlier warnings (this uses a higher salience to fire last). Additionally, the age of the **final destination domain** is considered – if the final site reached is newly created (e.g., < 6 months or < 1 year old), that fact adds to suspicion, whereas if it's older than a year, that is a mild reassurance factor.

To illustrate the rule implementation, consider one example from the URL analysis rules that checks for usage of URL shortening services. Attackers often use shorteners to hide malicious URLs behind innocuous-looking links:

```
rule "URL: Shortener Detection"
when
    $e : Evidences( isUrlShortened == true )
    $sc : ScoreCard()
then
    List<RuleDetail> details = new ArrayList<>();
    details.add(new RuleDetail("shortener", $e.getShortenerType()));
    $sc.addScore(
        weightsConfig.getUrlWeight("shortener"),
        drools.getRule().getName(),
        "URL uses a shortener service, which can obscure the true destination
",
        details
    );
end
```

In this Drools rule, the condition (`when`) checks the `Evidences` fact to see if `isUrlShortened` is true – this would have been set during feature extraction if the URL domain matches a known list of shortening services (like bit.ly, tinyurl, etc.). The action (`then`) creates a detail entry noting the type of shortener used, and calls `$sc.addScore(...)` to increment the risk score. The `weightsConfig.getUrlWeight("shortener")` call retrieves a predefined weight for this indicator (perhaps a medium severity), and `drools.getRule().getName()` captures the rule's name for logging or explanation. The message explains why this is risky. This pattern of adding a score with context is repeated across rules, enabling the system to accumulate evidence and provide an explanation for each contribution to the final score.

Findings and Validation

After implementing the rule-based system, a comprehensive validation was performed using a series of attack scenarios and benign cases. The scenarios, provided or approved by the domain expert, were designed to test the system against realistic phishing techniques and also ensure that normal (legitimate) use cases are not falsely flagged. Below, we describe three illustrative scenarios and how the system responded:

Scenario 1: Multi-Hop Shortener Attack

Attack Vector: A user receives a suspicious link via email: `http://bit.ly/3xZpF8a`. This is a shortened URL (bit.ly). The true destination is not immediately visible to the user.

System Analysis:

1. **URL Analysis:** The feature extractor recognizes the domain `bit.ly` as a URL shortening service. This sets `isUrlShortened = true` in the evidences, causing the rule **"URL: Shortener Detection"** to fire. The URL uses HTTP (not HTTPS), but the main red flag at this stage is the shortener itself, which obscures the final destination.
2. **Redirect Analysis:** The system expands the shortened URL by performing an HTTP request. It discovers a chain of redirects:
- `bit.ly/3xZpF8a` → `redirect1.attacker.com`
- `redirect1.attacker.com` → `hack-paypal.com/login`
The redirect chain length is 2 (which is modest), but importantly it goes through two different domains (`bit.ly` to an attacker domain to another attacker domain). The rule **"Redirect: Domain Diversity"** triggers because the intermediate redirect domain and the final domain are not related (each hop is to a completely different domain, a common tactic to make tracking harder). Additionally, the content of the redirect URLs suggests an open redirect may have been used (the presence of a parameter like `?url=hack-paypal.com/login` in one of the URLs). This triggers the rule **"Redirect: Open Redirect Pattern."**
3. **Rules Triggered:** As a result of steps 1 and 2, multiple rules have already fired:
- *URL Shortener Used* – flagged the use of a short-link service.
- *Redirect Domain Diversity* – flagged that the redirect chain crosses multiple disparate domains.
- *Open Redirect Pattern* – flagged that an open redirect parameter was detected during the redirect chain.
4. **Final Destination Analysis:** The final URL after following redirects is `hack-paypal.com/login`. The system checks this against known malicious domains (a blacklist or threat intelligence service) and finds a match – this domain is known for phishing (impersonating PayPal). The rule **"URL: Known Blacklist Entry"** fires upon this recognition, adding a very high score. The domain's WHOIS information also indicates it was registered only a month ago, which aligns with typical phishing behavior (use of newly created domains).
5. **Final Conclusion: Phishing detected (High Confidence).** The ScoreCard accumulated a high-risk score (for example, 92 out of 100) after considering all the evidence. The link is classified as a phishing attempt.

Explanation: The system generates an explanation noting the key findings:
- Multiple redirects (hops) were present, which is uncommon for legitimate sites and often used to hide malicious intent.

- The domains involved in the redirection chain were unrelated to each other (indicating the user was sent outside any single trusted domain space).
- The final landing domain is a known fraudulent site impersonating a legitimate service (in this case, pretending to be a PayPal site).
- These combined indicators (URL shortener + multi-hop redirects + known malicious domain) overwhelmingly point to a phishing attack.

Scenario 2: Trusted Internal Redirect

Attack Vector: A user clicks on a link that appears to be an internal company link: `https://secure.intranet.corp.com/app`. This is a URL on the company's intranet domain (corp.com). On access, it performs one redirect to `internal-redirect.corp.com/home`.

System Analysis:

1. **URL Analysis:** The initial URL uses HTTPS and is on a domain (`intranet.corp.com`) that is recognized as a trusted internal domain (assuming an allowlist of corporate domains is configured). No shortening service is involved and nothing abnormal like an "@" sign or an IP address is present. The rule set may include a rule to mark known trusted domains; in this case the presence of "corp.com" in an internal whitelist leads to a rule (e.g., **"URL: Trusted Domain"**) which might tag the URL as low risk (or prevent certain other rules from adding score).
2. **Redirect Analysis:** The URL leads to one redirect, landing on `internal-redirect.corp.com/home`. Both the initial and final domains share the same root (`corp.com`). The system sees this as an **expected internal redirect** pattern. A rule (for example, **"Redirect: Same Domain Root"**) confirms that the redirect stays within the trusted organizational domain, which in effect is a benign sign. No rules for excessive redirects or domain mismatches fire, since there was only a single hop and it remained in-company.
3. **DNS Analysis:** The DNS features for both `intranet.corp.com` and `internal-redirect.corp.com` look normal. They resolve to internal IP ranges or known addresses, have standard TTL values, and likely have MX/SPF records (since they are real corporate subdomains). No DNS-related rules trigger (e.g., there are A records, and these domains are long-established).
4. **DOM Analysis:** The page `internal-redirect.corp.com/home` is retrieved (if required) and appears to be a standard intranet web application page. It contains no unexpected login forms or fields asking for credentials beyond the normal company login (which the user likely already passed through at secure.intranet.corp.com). No external resource loads or suspicious scripts are detected in the DOM. In short, the page structure does not exhibit any known phishing markers.
5. **Final Conclusion: Legitimate link (High Confidence).** The system assigns an extremely low risk score (for instance, 5 out of 100). The link is classified as legitimate/benign.

Explanation: In the report to the user or analyst, the system would explain:
- The redirect chain was confined to the organization's own domains, which is expected behavior for internal links (e.g., SSO or navigation redirects).

- Only one redirect occurred, which is a normal amount (common for load balancers or web routing).
- All examined indicators (URL, DNS, content) were consistent with a legitimate internal site.
- No phishing patterns were detected in either the URL or the page content.

This scenario demonstrates the system's ability to recognize and pass through benign scenarios, avoiding false positives by understanding context (in this case, an internal corporate context).

Scenario 3: Homograph Attack with Punycode

Attack Vector: A user sees a link that *looks* like `www.paypal.com` at first glance, but on closer inspection it is `www.paypa1.com` – the "l" in "paypal" is actually the number 1, or possibly a similar-looking character from another alphabet (Cyrillic 'a' for instance). The actual URL is `www.xn--paypa1-o0d.com` (which is the Punycode-encoded form of a domain containing mixed character sets). This is a classic **homograph attack** attempting to trick users into thinking they are visiting a familiar trusted site.

System Analysis:

1. **URL Analysis:** The lexical analysis immediately notices the presence of non-standard characters in the domain. The domain, when decoded from Punycode, contains characters that are not all from the Latin alphabet (or contains a suspicious mixture of digits and letters in what looks like a brand name). The rule **"URL: Likely homograph (IDN/punycode)"** fires upon detecting the `xn--` prefix and/or unusual Unicode characters. Additionally, a specific rule for brand name impersonation could be present (for example, detecting known brand names with character substitutions), but even without that, the combination of homograph and the content of the URL raises a major red flag. The system might note that "paypa1" closely resembles "paypal."
2. **Domain Characteristics:** The WHOIS and DNS analysis reveals that this domain was registered very recently (say, within the last 7 days). A rule such as **"DNS: Young domain"** or a URL rule for domain age fires because legitimate brand domains (like the real paypal.com) have existed for years, whereas this lookalike is brand new. The domain likely has no MX records or other indications of an established web presence, which further contributes to suspicion. If a rule is in place for checking visual similarity to known high-value brands (using a list of brand names), it would also trigger given "paypa1" vs "paypal".
3. **WHOIS/Registrant Analysis:** The system (if augmented with WHOIS detail parsing) finds that the registrant's information (the organization or individual who registered the domain) is either privacy-protected or linked to previous malicious domains. Perhaps the registrar or country of registration is one known for abuse. While our rule set did not explicitly list registrant reputation checks in earlier sections, the expert's scenario description assumes some capability to flag suspicious WHOIS data. In this case, we can say a rule flagged the registrant or registrar as high-risk (for example, domain registered via a registrar often used by attackers, or an address in a region with many cybercrime issues).
4. **Final Conclusion: Phishing detected (High Confidence).** The collective evidence – an internationalized domain designed to look like a known brand, extremely young domain age,

and potentially suspicious registration details – leads to a high risk score (e.g., 88/100) and the system classifies the URL as a phishing attempt.

Expert Validation:

> "This attack is particularly dangerous because users might not notice the subtle character differences. The homograph detection capabilities implemented here are critical for enterprise protection."
> *– David Marques, Cybersecurity Expert*

Explanation: The system's explanation would highlight:

- The domain name is **designed to impersonate a trusted brand** (PayPal) using look-alike characters, a tactic used to fool users.
- The domain was **registered very recently**, which is unusual for what appears to be a well-known company's site and is a strong indicator of fraud.
- Any available registrant or hosting information further suggests this is not an official site (e.g., it might be registered to an individual rather than PayPal, using a registrar known for cheap, anonymous registrations).
- Based on these factors, the system concludes this is a likely phishing site attempting to masquerade as the legitimate PayPal website.

These scenarios illustrate how the rule-based system behaves in practice: it successfully identifies sophisticated phishing techniques (using multiple indicators in combination) and also correctly recognizes when a scenario is benign. The expert reviewed these outcomes and confirmed that the system's reasoning and conclusions aligned with what a human analyst might conclude when faced with the same evidence. The transparent, step-by-step explanation for each decision is a notable strength, as it provides immediate insight into **why** a link was flagged, helping build trust in the system's outputs.

Key Learnings and Competencies Acquired: Throughout the project, the team gained valuable experience in multiple domains:

- **Knowledge Engineering Fundamentals:** We learned how to systematically extract and formalize knowledge from a domain expert. This involved designing effective interview questions, capturing tacit expert heuristics (like "new domains are suspicious" or "too many redirects are bad") and translating them into explicit rules. The iterative refinement of the rule base demonstrated how to manage knowledge evolution over time and the importance of thorough documentation for each rule (documenting its purpose and origin from expert input or literature).

- **Symbolic Reasoning and Drools Expertise:** The project provided deep exposure to rule-based AI as opposed to traditional imperative programming. We developed skill in structuring facts and rules in Drools, including best practices such as fact normalization (all evidence in one object), using **salience** and **agendas** to manage rule firing order when needed, and global variables for configuration. Debugging a Drools knowledge base – using logs to trace which rules fired and in what order – was a significant learning experience, as was optimizing rules to prevent conflicts or redundancy. We also became adept at the KIE API for integrating Drools into a Java application, configuring sessions, and handling rule updates.

- **Cybersecurity Domain Knowledge:** Building the system dramatically increased our understanding of phishing techniques and indicators. By researching and encoding rules, we delved into how attackers craft URLs (homograph attacks, subdomain tricks), what DNS configurations hint at malicious intent, how phishing pages behave differently from normal pages, and how redirect chains can be exploited. We also learned about various security resources: for instance, using threat intelligence APIs and datasets like PhishTank, which can enrich detection, and understanding the common patterns found in phishing kits and campaigns. The expert's real-world anecdotes (such as the way attackers abuse cloud hosting or how they respond when one tactic is blocked) gave us a pragmatic perspective that complemented theoretical knowledge.

- **Software Architecture and Integration:** Implementing the solution in a realistic environment taught us about designing a **layered architecture**. We practiced separating concerns (UI vs. logic vs. data fetching vs. inference engine) so that each piece could be developed and tested independently. We also gained experience integrating an AI component (Drools engine) within a standard web service framework (Spring Boot). This included managing configuration files, using dependency injection for the KIE session, exposing endpoints, and handling error cases (like timeouts when a feature extraction call fails). Performance considerations also came to the forefront: we learned to consider the cost of each feature extraction step (for example, DNS queries or HTTP fetches can be slow) and thus the importance of timing and possibly caching results for efficiency.

- **Explainability and Auditability:** Because one of the goals was an explainable AI system, we paid special attention to how decisions would be recorded and presented. We learned techniques for generating human-readable rule firing logs and summaries, designing the `RuleDetail` data structure to capture the salient details of each triggered rule. Ensuring that every score contributed had an accompanying explanation greatly improved the transparency of the system. This focus on explainability also reinforced good practice in **audit trails** – the system logs each analysis and its outcome, which is important for later review (especially if a legitimate URL was flagged, security teams can trace which rule caused it and decide if adjustments are needed). In short, we gained a practical understanding of how to build AI systems that are not "black boxes" but rather can articulate their reasoning, which is crucial for cybersecurity applications where trust and verification are important.

Challenges, Limitations and Reflections

Developing the phishing detection expert system presented several challenges, and it also highlighted inherent limitations of a rule-based approach in this domain. We reflect on these issues and how we addressed them:

- **Managing Rule Complexity:** By the end, the system included 50 rules spread across four modules. Ensuring that all these rules work together without conflict and are easy to maintain was a non-trivial challenge. **Solution:** We mitigated complexity by modularizing the rule sets (as described) and clearly separating the logic for URLs, DNS, DOM, and redirects. Within each module, rules were written to be as orthogonal as possible (each targeting a specific indicator). During development we added one rule at a time and tested incrementally, which taught us that starting simple and gradually increasing complexity (with continuous validation) is essential in knowledge-based systems. This approach prevented a situation where we had to untangle a web of interdependent rules after the fact.

- **Feature Extraction Accuracy:** The system's detection is only as good as the data it receives. One challenge was to ensure we capture all relevant features of a potential phishing attempt – for example, if the feature extraction misses that a form is posting to an external domain, no rule will ever fire on that condition. **Solution:** We worked closely with the expert to enumerate what features are "must-haves." This collaboration was vital to cover edge cases (like checking not just for the presence of password fields, but also looking at where forms submit data). We realized that domain expertise is crucial for feature selection: the expert could point out subtle signs of phishing that automated scanners might ignore. Comprehensive unit tests on feature extraction were created (feeding known phishing and benign URLs to verify the extracted evidence was correct) to improve accuracy.

- **Threshold Calibration (False Positives vs False Negatives):** Determining how sensitive the system should be was a significant challenge. Too aggressive, and it would flag many legitimate sites (false positives), too lax, and phishing might slip through (false negatives). **Solution:** We adopted an iterative calibration strategy. After an initial setting of weights and thresholds based on intuition and literature (e.g., "homograph attack" should be high severity, "missing MX record" medium, etc.), we ran the system on a validation set (including known phishing examples and known good examples from our expert and public sources). We adjusted the scoring weights and the final cutoff thresholds in response to these results, each time consulting the expert's judgment on what an acceptable outcome is. **Learning:** This process underscored that a perfect zero false positive/negative rate is impractical – instead, one must explicitly manage the trade-off based on the use case (in an enterprise setting, a small number of false positives might be acceptable if it means catching almost all phishing attempts). The involvement of stakeholders (the expert's perspective on risk tolerance) was key in this calibration.

- **Performance Optimization:** The need to perform these analyses in real-time (for instance, scanning links in incoming emails on the fly) meant performance was a concern. With multiple network calls (DNS, HTTP fetches) and a rule engine cycle, the system could become slow under heavy load or when analyzing a very complex case. **Solution:** We addressed performance at multiple levels: caching DNS results for repeated queries, setting sensible timeouts for network operations, and ensuring the Drools engine was used optimally (stateless sessions to avoid overhead, and using **salience** to short-circuit certain evaluations). The fact model was optimized to avoid expensive computations inside rules – all heavy computation is done in the feature extraction stage (outside Drools), so rules themselves run very quickly just matching boolean flags or numeric thresholds. We also measured throughput and found it meets the target for our scenario (on the order of tens of milliseconds per URL in typical cases, excluding network I/O). **Reflection:** Continuous performance monitoring and optimization need to be part of deployment because new kinds of content (especially complex web pages) could change the time profile of the system.

- **Integration Complexity:** Bringing together the Drools engine, external services, and the Spring Boot infrastructure posed integration challenges. Each component (e.g., calling VirusTotal API for reputation, using the Drools KIE container, deploying on a server) could introduce failures or incompatibilities. **Solution:** We tackled this by clear interface design – for example, designing the `PhishingAnalysisService` to encapsulate all steps so it provides a clean API to the controller. We also performed end-to-end integration testing in an environment similar to production to catch issues early (like ensuring the Drools rules are correctly packaged and loaded in the Spring Boot jar). Setting up proper logging at each integration point greatly aided in diagnosing problems. **Learning:** Integration testing is as important as unit testing in systems like this; verifying that everything works together (rules triggering after a real HTTP redirect is followed, etc.) is crucial. Additionally, deploying a pilot version in a controlled environment allowed us to observe any runtime issues (memory, multi-threading) and fix them before actual use.

Identified Limitations: Despite the success of the project, certain limitations of the approach were acknowledged (many of which were highlighted by the expert):

- **Zero-day Attack Coverage:** A rule-based system relies on predefined patterns. Truly novel phishing techniques that do not match any of the current rules may evade detection. This is a common challenge: if attackers devise a brand-new trick (say, abusing a new third-party service or a new encoding method not anticipated by our rules), the system would likely miss it until the knowledge base is updated. Regular updates and expert input are required to handle such emerging threats.

- **Adaptive Attackers:** Phishing actors are continually evolving. As soon as certain indicators (like specific keywords or redirect patterns) become known to defenders, attackers may shift tactics (for example, including benign content or mimicking patterns of legitimate emails). A static ruleset might become outdated if not continuously maintained. The system as built is not automatically learning; it will require periodic knowledge updates informed by new incidents.

- **False Positive Sensitivity:** In an enterprise deployment, even a small percentage of false positives (legitimate URLs flagged as phishing) can cause disruption or erode trust in the system. Users might ignore warnings if too many benign links trigger alerts. While we calibrated the system to minimize this, the limitation remains that rules might occasionally misfire (e.g., an unusual but harmless URL might match a pattern and get flagged). Handling false positives is as much a process issue as a technical one – it requires careful rule tuning and perhaps a mechanism for analysts to provide feedback on incorrect classifications.

- **Performance vs. Depth of Analysis:** There is an inherent trade-off between how thorough the analysis is and how quickly results are needed. We included a fairly extensive set of features (DNS, DOM, etc.), each of which costs time. In scenarios where near-instant detection is required (like scanning every URL in a high-traffic email system), one might need to cut some checks or invest in more infrastructure. Our system in its full configuration might be too heavy for certain real-time constraints unless optimized further or scaled out.

- **Integration and Environment Diversity:** Every enterprise has a unique IT environment. Integrating this solution with existing security systems (SIEMs, email gateways, browsers) might encounter compatibility issues or require custom adaptation. For example, our system might produce an alert, but if not integrated with the email client or SOC (Security Operations Center) workflow, that alert could be missed. Thus, actual deployment would need careful integration planning, and our project did not fully address all those environment-specific challenges.

Reflections on the Learning Process: This project underscored several best practices in knowledge-based system development. First, the importance of **clear problem decomposition** became evident – breaking down "phishing detection" into sub-problems (URL, DNS, etc.) made the task manageable and the solution modular. Second, **iterative validation with the expert** was critical: rather than trying to build the rulebase in isolation, each increment was reviewed, which caught issues early and guided the direction of development. We also learned to always consider **operational constraints** in parallel with technical design – for instance, asking "How will a security analyst react to this output?" or "Can this run within the required time window?" while designing the system. Balancing **precision and recall** (catching bad links vs. not harassing users about good ones) was more than a technical tweak; it required policy decisions and a clear understanding of enterprise risk appetite, taught to us by the expert's perspective. Finally, this experience reinforced that a knowledge-based solution is never truly "finished" – it must evolve as the threat landscape evolves, which means establishing a process for continuous learning and updating.

One of the most valuable aspects of the project was the **close collaboration with the domain expert**. This partnership bridged the gap between theoretical knowledge and real-world practice. The expert's feedback on our intermediate outputs (like explaining why a certain rule might be prone to false positives, or suggesting additional rules based on incidents he had seen) significantly improved the system's quality. It demonstrated the huge value that expert knowledge brings to AI projects, especially in a specialized domain like cybersecurity, and how such collaboration can accelerate development and ensure relevance of the final product.

Future Perspectives and Recommendations

Looking ahead, there are several avenues to enhance and expand the phishing detection system. These range from immediate, tactical improvements to broader long-term developments, as well as general recommendations for knowledge engineering projects of this nature.

Short-term Enhancements (1–3 months)

- **Threat Intelligence Integration:** Incorporate direct integration with services like VirusTotal for URL and domain reputation checks. This would allow the system to fetch real-time threat intelligence (e.g., if a URL is already reported as malicious elsewhere) and use that information as an additional feature or rule trigger.
- **Dynamic Whitelist/Blacklist Updates:** Automate the maintenance of trusted domain whitelists and malicious domain blacklists by pulling from reputable feeds. For example, regularly update from OpenPhish or PhishTank lists so that known phishing domains are immediately flagged and known good domains reduce the chance of false positives.
- **Rule Performance Optimization:** Profile and fine-tune the Drools rules for performance. This could include reordering conditions for efficiency, combining certain rules to reduce overhead, or caching intermediate results. The goal would be to decrease latency per URL analysis, preparing the system for higher throughput scenarios.
- **Enhanced Logging & Monitoring:** Improve the logging system to capture detailed analysis traces and develop a monitoring dashboard. This could involve logging each rule's execution time, the score contributions in a structured format, and providing an interface for analysts to visualize rule firings and system performance metrics over time.
- **Analyst Interface:** Develop a simple user interface or web dashboard for security analysts to submit URLs and review results. This would make the tool more accessible for internal use, showing the breakdown of the analysis (which rules fired, score breakdown, etc.) in a user-friendly manner, and could also allow analysts to provide feedback or mark false positives directly through the interface.

Medium-term Development (3–6 months)

- **Machine Learning Hybridization:** Explore a hybrid approach by incorporating a machine learning component alongside the rule engine. For instance, train a statistical model (such as a classifier on URL features or an ensemble that looks at patterns across many phishing pages) to work in tandem with the rules. This could catch patterns that are hard to capture with static rules and improve detection of novel attacks. The ML component's output could be another input to the rule engine (e.g., a "ML suspicion score" fact).

- **Adaptive Thresholds:** Implement an adaptive scoring mechanism that adjusts thresholds based on organizational context and historical data. For example, the system could learn that certain departments frequently use unusual URLs (and thus adjust sensitivity for those contexts), or it could automatically tune the risk score cutoff if it finds that too many alerts are false positives, under the supervision of analysts.
- **Browser Extension for Client-side Scanning:** Develop a browser plugin that leverages the backend system to scan URLs in real-time as users navigate. This extension could intercept clicks on links or new URL loads and send them to the server for analysis, warning the user if the result is malicious. This brings the detection closer to the end-user's environment and can prevent phishing at the point of click.
- **Email Gateway Integration:** Work on integrating the detection engine into an email server or proxy. Many phishing attempts arrive via email, so hooking the system into an email filter (for example, via a microservice that checks URLs in incoming emails) could provide automatic quarantine of emails containing high-risk links. This requires ensuring the system can handle the email throughput and possibly simplifying results into an actionable decision (allow, flag, or block the email).
- **Incident Feedback Loop:** Establish a formal feedback process wherein every time a phishing incident is confirmed (or a false positive occurs), that information is used to update the system. This may include creating tools for analysts to input new phishing URLs and their characteristics into a knowledge base, which can semi-automatically generate new Drools rules or update existing ones. Over six months, this could significantly improve the system's adaptability and coverage by learning from incidents.

Long-term Vision (6–12 months)

- **Advanced Evasion Technique Detection:** Research and incorporate methods to detect more advanced or subtle evasion tactics. This might involve simulating how an adversary could bypass the current rules and enhancing the system accordingly. For instance, incorporating detection for phishing kits that use random subdomain generation each time, or using more sophisticated analysis of page screenshots (via image recognition) to detect brand logos in phishing pages, which goes beyond the current text/DOM-based approach.
- **Zero-day Phishing Resilience:** Aim for a more resilient system against unknown attacks by using an ensemble of approaches. In the long term, this could mean combining our rule-based system with anomaly detection techniques that model normal email or browsing behavior for a user or organization. If something deviates

strongly from the norm (even if it doesn't match a known rule), the system could raise an alert for further manual review.

- **Real-time Threat Landscape Adaptation:** Implement an automated pipeline that continuously ingest updates from cyber threat intelligence (new phishing campaigns, new deceptive techniques reported) and seamlessly updates the rule set or configurations. This might involve integration with communities like Anti-Phishing Working Group (APWG) feeds or others. The vision is a system that is always "learning" from the broader security community in real-time, not just from internal inputs.
- **Compliance and Reporting Automation:** Build features to automatically generate compliance reports and audit logs that demonstrate what the phishing defense system has been doing. For regulated industries, showing due diligence in email/link scanning is important. Over a year, the system could evolve to produce monthly security reports, highlight trends (e.g., "phishing attempts blocked this quarter"), and ensure its processes meet any relevant data protection or cybersecurity regulations.
- **Enterprise-Scale Rule Management:** As the rule base grows, develop an enterprise-grade management interface for the knowledge base. This would allow rule versioning, testing new rules in a sandbox, and deploying updates to production with minimal risk. In the long-term vision, multiple such rule-based detectors (for different threats) could be managed under one platform, so having a centralized rule management and analytics console would be beneficial.

Recommendations for Knowledge Engineering Practitioners

Based on the experience gained, we offer the following general recommendations for similar knowledge-based AI projects:

- **Invest in Domain Expert Collaboration:** Engaging closely with one or more domain experts from the outset dramatically improves the relevance and accuracy of the system. Their real-world perspective can save you from pursuing impractical solutions and will enrich the rule set with conditions that truly matter. Regular expert reviews and knowledge transfer sessions are invaluable – they multiply the project's value.
- **Begin with Clear Problem Scope and Use Cases:** Clearly define the boundaries of the problem and gather representative scenarios early. In our case, enumerating specific phishing scenarios (like credential theft pages, financial fraud emails, etc.) guided the rule creation. Having well-defined use cases keeps development focused and prevents scope creep.
- **Iterative Development and Validation:** Don't attempt to build the entire knowledge system in one go. Start with a few core rules and a simple test, validate with the expert or real data, and then iterate. Continuous feedback loops ensure that mistakes are caught early and that the system evolves in the right direction. This also helps in managing complexity; you can adjust course with each iteration rather than doing a large overhaul at the end.

- **Document Rules and Decisions:** Maintain comprehensive documentation for each rule – what it detects, why it's important, who/what source suggested it, and when it was last updated. This will greatly help when revisiting the project or handing it over to others. Also document key design decisions and the rationale behind thresholds or weights chosen. In complex rule systems, memory can fade; documentation ensures knowledge isn't lost.
- **Balance Sophistication with Maintainability:** Aim for the simplest rule that accomplishes the goal, and avoid over-engineering. While it might be tempting to create very complex rule conditions to cover every nuance, simpler rules are easier to troubleshoot and update. A combination of simpler rules can often achieve similar coverage with greater transparency than one convoluted rule.
- **Plan for Continuous Learning:** Treat the knowledge base as a living artifact. Set up a plan for regular updates (e.g., monthly knowledge review sessions, or a standing arrangement with the domain expert for updates). Threats will evolve, and your system should too – allocate time and resources for this continuous improvement rather than considering the project one-off.
- **Prioritize Explainability and User Trust:** From day one, design the system to produce explanations for its decisions. This not only helps users trust the system but also aids debugging (if a rule misfires, you'll clearly see its effect). In fields like cybersecurity, having an explainable AI is often non-negotiable because of compliance and the need for human analysts to make final decisions. It's easier to build it in from the start than to bolt it on later.
- **Consider Operational Constraints Early:** Finally, always design with the deployment context in mind. If the system is intended for enterprise use, think about how it will integrate with existing workflows, what the performance requirements are, who will maintain it, etc. Non-functional requirements (speed, security, interoperability) can heavily influence design choices in a knowledge system and should be treated as first-class considerations.

Conclusion

This knowledge acquisition report has chronicled the journey from an initial concept to a fully implemented **Drools-based phishing detection system**, reflecting on both the technical and methodological dimensions. The project combined academic research, hands-on rule engineering, and domain expert guidance to create a solution that is not only technically sound but also practically relevant to enterprise cybersecurity needs.

In summary, **several key achievements** were realized:

- Formalization of expert cybersecurity knowledge into **34 executable Drools rules** spanning multiple facets of phishing detection.
- Implementation of a layered architecture that supports real-time URL analysis with clear separation of concerns (feature extraction, inference engine, scoring, etc.), ensuring maintainability and scalability.

- Development of a rule base that is **interpretable and explainable**, producing human-readable reasons for each decision to facilitate trust and compliance in an enterprise setting.
- Successful validation of the system against realistic attack scenarios, demonstrating that it can catch sophisticated phishing attempts (e.g., homograph attacks, multi-stage redirects) while allowing normal behavior (e.g., internal redirects) to pass.
- Integration of the rule engine within a modern web service (Spring Boot), showcasing how symbolic AI techniques can be deployed in tandem with conventional software infrastructure.
- Comprehensive documentation and knowledge transfer materials (including this report and appendices with full rule listings, source code, and test cases) to support future maintenance or expansion of the system.

Several **critical success factors** contributed to these outcomes. Foremost was the deep involvement of the domain expert throughout the project, from requirements gathering to final validation – this ensured the project stayed grounded in reality and addressed the right problems. The use of an **iterative, incremental development approach** allowed continuous improvement and course correction, avoiding large-scale rework by integrating feedback continuously. The **modular architecture and rule organization** proved essential in managing complexity, allowing different team members or future contributors to focus on one aspect (e.g., DNS rules) without affecting others. Emphasizing **explainability and auditability** from the inception of the project meant that the end product was aligned with the needs of cybersecurity analysts and regulators. Finally, recognizing and designing for **operational constraints** (performance, integration, user acceptance) from the start ensured that the developed solution is not just a proof of concept, but something that could realistically be adopted in an enterprise environment.

From an educational standpoint, the **knowledge and skills acquired** through this project span multiple areas: we honed our knowledge engineering abilities (eliciting and encoding expert knowledge), gained practical experience with a production rule engine (Drools) and its nuances, deepened our understanding of phishing and security practices, and learned how to architect and integrate an AI-driven solution within a larger software context. This holistic experience – bridging theory, domain expertise, and implementation – has prepared the team to tackle future challenges in AI and cybersecurity with a strong foundation. It has reinforced the idea that effective AI solutions in critical domains often require a **synergy of human expertise and technological tools**, rather than relying on one in isolation.

In conclusion, the Drools-based approach to phishing detection demonstrated in this project underscores that not all AI innovation relies on machine learning or big data. **Symbolic reasoning and expert systems remain invaluable**, especially in domains where interpretability, precision, and agility in incorporating new knowledge are paramount. By combining the strengths of human expertise with the consistency and speed of rule-based automation, the project delivered a system that can adapt to new phishing threats in an explainable manner. This work not only yields a functional security tool but also exemplifies a methodology for knowledge acquisition and application that can be replicated in other problem domains where expert knowledge is abundant but needs to be systematically harnessed. The competencies and insights gained here have immediate applicability in the realm of enterprise security and beyond, as organizations increasingly seek AI solutions that are **transparent, trustworthy, and tightly aligned with human expert knowledge**.

Bibliography and References

Academic Sources

- Patil, D. R., & Dhage, S. N. (2019). *Framework for anti-phishing: URL feature extraction and classification*. In Proceedings of the International Conference on Information Security and Privacy (pp. 105–112).
- Hranický, R., Horák, A., Polišenský, J., Ondryáš, O., Jeřábek, K., & Ryšavý, O. (2024). *Spotting the Hook: Leveraging Domain Data for Advanced Phishing Detection*. In 20th IFIP/IEEE International Conference on Network and Service Management (CNSM 2024) (pp. 1–7).
- Omolara, A. E., et al. (2025). *DaE^2: Unmasking malicious URLs by leveraging diverse and efficient ensemble machine learning*. Computers & Security, 147, 103–118.
- Bayer, J., Maroofi, S., Hureau, O., Duda, A., & Korczynski, M. (2023). *Building a Resilient Domain Whitelist to Enhance Phishing Blacklist Accuracy*. In APWG eCrime Symposium 2023 (pp. 1–14).
- Alsabah, M., Nabeel, M., Boshmaf, Y., & Choo, K.-K. R. (2022). *Content-Agnostic Detection of Phishing Domains using Certificate Transparency and Passive DNS*. In Research in Attacks, Intrusions, and Defenses (RAID 2022), ACM.

Technical References

- **Drools Official Documentation:** https://docs.drools.org
- **Apache KIE Project:** https://kie.apache.org
- **Spring Boot Official Documentation:** https://spring.io/projects/spring-boot
- **OpenPhish API Documentation:** https://www.openphish.com/api.html
- **PhishTank API Documentation:** https://www.phishtank.com/api_documentation.php
- **VirusTotal API Documentation:** https://developers.virustotal.com

Industry Reports

- **ENISA Threat Landscape 2024/2025** – European Union Agency for Cybersecurity, annual threat landscape report covering phishing and other cyber threats.
- **World Economic Forum Global Cybersecurity Outlook 2025** – Insights on emerging cyber threats and defense strategies, including phishing trends in the global context.

- **Cisco Security Outcomes Study 2024** – Industry study outlining key practices for successful security programs, with relevant findings on threat detection and user awareness (includes phishing-related metrics).