International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 13 Issue: 11 | Nov 2026 www.irjet.net p-ISSN: 2395-0072

ScamShield: An Explainable and Privacy-Conscious Framework for Browser-Based Scam Detection

P Aishwarya[1], Pranjal Kumar Jha[2], Pranav P[3], Ron Varghese Cherian[4], P Namrata[5], A. Vishwanath[6]

[12345] *Student, Department of Computer Science and Engineering, Hyderabad Institute of Technology and Management (HITAM), Hyderabad, India.*
[6] *Assistant Professor, Department of Computer Science and Engineering, Hyderabad Institute of Technology and Management (HITAM), Hyderabad, India*

Abstract- *Phishing and online scams persist because a deceptive destination can imitate a trusted service convincingly when a user decides to click. Automated detectors assign a risk label, but a label alone gives little basis for judging whether a warning is well founded. ScamShield is a browser-based prototype coupling multi-signal URL assessment with an explicit account of the evidence behind each verdict. A Chrome Manifest V3 extension submits URLs to a local FastAPI service combining structural and lexical URL analysis, brand-impersonation detection using character-substitution patterns and Levenshtein distance, RDAP-derived domain-age information, shortener and redirect indicators, and an experimental logistic-regression classifier over thirteen URL features. A weighted risk engine maps evidence to Low Risk, Caution, Suspicious, High Risk or Confirmed Threat and returns reasons, reporting evidence strength separately from severity. Community reports are stored in SQLite through SQLAlchemy, distinct from verified threat indicators, and a local heuristic fallback provides a limited URL assessment when the backend is unreachable. The extension analyses URLs only; it does not read page content, form fields, passwords, cookies or browsing history. Evaluation comprised six deterministic URL vectors, documented manual browser checks and a 56-test automated API suite that exposed and confirmed correction of two input-handling defects in the reporting endpoint. These checks show consistent workflow behaviour but do not establish detection accuracy against an independent benchmark; the classifier and local threat database remain experimental and local.*

Key Words: Browser Security, Phishing Detection, Explainable Artificial Intelligence, Chrome Extension, FastAPI, URL Analysis, Machine Learning.

1 INTRODUCTION

Online banking, shopping, employment and public administration now depend on web services, and the messages that surround those services have become a routine channel for fraud. A scam campaign rarely needs to compromise a browser; it needs only to persuade a user to open a destination that imitates a familiar brand, requests credentials, or promises a payment. Phishing therefore exploits the gap between the apparent identity of a destination and its actual ownership — a gap that is often partly visible in the address itself. A brand name in an unrelated domain, a character substituted for a visually similar one, a shortened link, or a chain of redirect-like parameters can each indicate that a destination deserves scrutiny.

Research and practice offer several families of countermeasures: blocklists and threat-intelligence lookups, rule-based URL heuristics, learned classifiers, page-content analysis and domain reputation [1]–[9]. Each addresses part of the problem and each leaves a residue: blocklists cannot describe an unreported destination, heuristics are transparent but evadable, learned models depend on data that a small prototype rarely possesses, and content inspection requires fetching hostile pages. The output is also consumed by a person under time pressure. A warning that states only that a destination is “unsafe” gives the user no way to connect the warning to what is visible on screen, and a stream of unexplained warnings encourages dismissal [10]–[12].

An explanation must nevertheless remain proportionate to the evidence. A long URL, an unfamiliar top-level domain or a brand-like string is not proof of fraud, and the absence of a threat record is not proof of safety. ScamShield is organised around this distinction: risk severity expresses how much concern the evidence warrants, while evidence confidence expresses how much evidence was available. The two are reported separately, and neither is presented as a calibrated probability.

The prototype consists of a Chrome Manifest V3 extension, a Python FastAPI service and a SQLite database accessed through SQLAlchemy. The extension forwards destination URLs to the backend, which runs a set of detector modules,

consults a local threat-indicator table for exact matches, optionally enriches the assessment with Registration Data Access Protocol (RDAP) registration data, and fuses the results through a weighted risk engine. The engine returns one of five categories Low Risk, Caution, Suspicious, High Risk or Confirmed Threat with the reasons that contributed. The extension presents the result in its popup and, for elevated categories, in a warning interstitial that offers a return path and a discouraged override. Community reports are stored separately from verified indicators, and a reduced heuristic fallback preserves limited function when the backend is unreachable.

The main contributions of this work are:

- an integrated browser-side pipeline combining structural, lexical, brand-impersonation, domain-intelligence and redirect-related evidence rather than a single rule;
- a risk engine whose strongest category, Confirmed Threat, is reserved for an exact match against a local indicator and is never produced by score alone;
- an explicit separation of risk severity from evidence confidence;
- a privacy-conscious boundary in which only destination URLs, and base domains for domain-age lookups, leave the browser;
- a community reporting path whose records remain unverified and structurally distinct from indicators; and
- a conservative evaluation of six deterministic vectors, documented manual checks and a 56-test API suite that exposed two input-handling defects, both corrected.

Section 2 reviews related work and states the research gap. Sections 3 to 5 describe the methodology, modules and implementation. Section 6 reports results, Section 7 discusses their interpretation, and Section 8 concludes.

2 RELATED WORK

URL-based detection has been studied from several directions, and the approaches differ not only in accuracy but in what they can observe, what they disclose and what they can explain.

2.1 Blocklists and Threat Intelligence

Blocklist-based protection compares a URL, host or resource against known-bad entries; Google Safe Browsing exposes maintained lists and lookup mechanisms of this kind [9]. Its strength is evidential, because a match associates the destination with previously observed activity. Its weakness is temporal: Sheng et al. found that phishing blacklist entries were typically added after a substantial delay and that coverage varied between lists, so a new campaign is frequently unlisted [8]. A non-match therefore means only that a destination is not currently known to be bad. Remote lookups also disclose the queried URL, or a derived representation of it, to the provider.

2.2 Heuristic and Lexical URL Analysis

Rule-based detectors examine the address string: length, dot and subdomain counts, digit and special-character composition, token structure, IP literals and suggestive words. Garera et al. combined such features with page-level signals [4], and Canali et al. built a fast pre-filter from URL- and page-derived features so that most pages need not be rendered [5]. Heuristics are inexpensive, deterministic and explainable, which matters when a warning must be justified. They are also brittle: thresholds are hand-set, unusual benign URLs produce false positives, and attackers adapt. A triggered rule describes which property was observed; it does not establish intent.

2.3 Machine-Learning Classifiers

Supervised classifiers learn a decision boundary from labelled URLs. Ma et al. formulated suspicious-URL identification as a large-scale online learning problem with continuous adaptation as labels arrive [1]; Le et al. proposed URLNet, which learns character- and word-level representations directly from the string [2]; and Sahingoz et al. compared seven classical algorithms over lexical and natural-language features [3]. Reported performance is a property of a dataset and a protocol, however. Class balance, label provenance, temporal drift and the separation of training from evaluation data determine whether a figure transfers to deployment, and a probability-like output is neither automatically calibrated nor self-explanatory.

2.4 Content-Based Analysis

Content-based systems examine the rendered page: form actions, links, script behaviour, branding and visual similarity to a protected site. Cantina combined page text, DOM structure and search-engine rank to judge whether a page is a legitimate instance of the brand it displays [6], and Whittaker et al. classified phishing pages at scale for Safe Browsing [7]. Such methods reach evidence that the URL does not contain, but pages must be fetched or rendered, cloaked content may behave differently for a detector than for a user, and interacting with hostile content enlarges the attack surface. ScamShield deliberately avoids this class of analysis.

2.5 Domain Intelligence and Typosquatting

Registration data adds context that the string cannot provide. RDAP provides a structured, HTTP-based replacement for WHOIS with machine-readable responses for registration events [13], from which an approximate domain age can be derived. Age is suggestive but not probative: many malicious domains are new and many new domains are legitimate. Typosquatting detection compares a candidate with protected brand names. Edit distance, defined by Levenshtein as the minimum number of insertions, deletions and substitutions needed to transform one string into another [14], is simple and explainable, while substitution rules capture visual confusions such as “1” for “l”. Neither can establish ownership, because internationalised domains, legitimate regional subdomains and unrelated short names all produce small distances.

2.6 Explainability and Warning Usability

Explainable artificial intelligence supplies techniques for exposing the basis of a prediction; LIME, for example, approximates a model locally with an interpretable surrogate [15]. For a rule-based pipeline, explanations can instead be generated from the signals that entered the decision, which avoids the fidelity gap between a surrogate and the model it approximates. Usable-security research shows why presentation matters. Sunshine et al. found that certificate warnings were widely ignored and that redesign improved comprehension without eliminating unsafe choices [10]; Felt et al. argued for simpler, more consistent connection indicators [11]; and Egelman et al. showed that active phishing warnings reduced unsafe behaviour relative to passive indicators [12]. A detector that cannot explain itself is unlikely to change behaviour.

2.7 Limitations of Existing Approaches and Research Gap

Three limitations recur. Coverage: no single source observes every deceptive destination, and each fails characteristically blocklists temporally, heuristics under evasion, classifiers under distribution shift, content analysis under cloaking. Interpretability: systems optimised for accuracy often report a score without a faithful account of the evidence, while transparent systems rarely integrate heterogeneous signals. Privacy and continuity: remote lookups disclose destinations and create a network dependency, whereas purely local detection lacks current intelligence.

ScamShield addresses an integration and reporting gap rather than a new detection principle. It combines transparent signal families, keeps an exact-match threat path separate from weighted scoring, and attaches to each verdict the reasons that produced it, while stating the strength of the evidence separately from the severity of the outcome. The contribution is the design, implementation and candid evaluation of that combination, not a claim of superior accuracy.

3 METHODOLOGY

3.1 Proposed System

ScamShield assesses a destination before the user commits to it, using information available in the URL and in domain registration metadata. It is not a browser-wide security boundary and does not claim to be one. Four principles govern the design. Evidence complementarity keeps the assessment dependent on structurally different observations, so that a single evasion does not defeat the pipeline. Asymmetric error handling records an unavailable signal as unknown and never as benign. Explicit uncertainty keeps severity and confidence separate. Data minimisation confines analysis to the address, so page content, credentials and history are never required.

3.2 System Architecture

Figure 1 shows the three boundaries. The browser side contains the Manifest V3 service worker, the popup, the warning interstitial and the fallback rules. The backend contains the FastAPI routes, detector modules, experimental classifier,

weighted engine and SQLite access. The external side contains the RDAP service. The extension sends URL-related input and receives a compact decision with reasons; neither page content nor credentials cross any boundary.

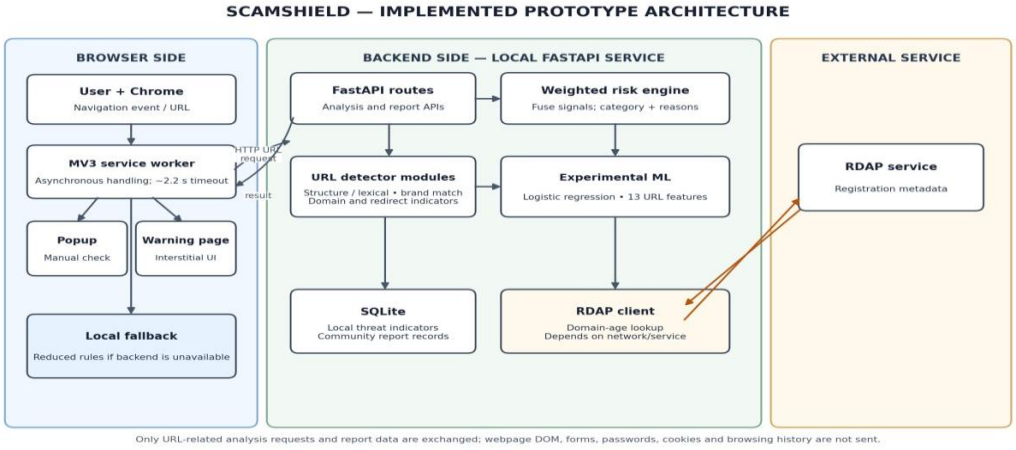


Fig -1: Implemented architecture of ScamShield, showing the browser, backend and external-service boundaries.

3.3 System Workflow and Data Flow

A navigation event supplies a URL to the extension's asynchronous handler, which issues an HTTP analysis request and waits for up to approximately 2.2 seconds. The backend routes the request through the pipeline, collects the available detector and database signals, applies the risk engine, and returns a category, reasons and a confidence label. The extension updates its presentation and, for elevated categories, opens the interstitial. If the request fails or the timeout expires, the extension applies its reduced local rules and labels the result as a fallback.

The handler is event-driven, so the timeout is a decision point rather than a gate. Browser scheduling, service-worker lifecycle, network conditions, backend processing and lookup latency all affect whether a verdict arrives before the destination begins to load. The prototype therefore claims that a verdict is produced and displayed in time, not that navigation is guaranteed to be blocked.

3.4 URL Analysis and Detection Methodology

Each request is normalised before analysis: a missing scheme is inferred, whitespace is removed, the host is lower-cased and parsed, and the URL is retained for exact comparison. Normalisation prepares the string for parsing; it does not resolve or visit the destination. A property that cannot be established is marked unavailable. Table 1 summarises the signal families.

Structural checks treat the URL as a series of observable decisions: whether the host is an IP address, the address length, subdomain depth, hyphens, at-signs and percent-encoding, credential-related terms such as "login" or "verify", unusual ports, and the irregularity of the character distribution. Each property is recorded separately so that it can be named in the explanation, and combined in the engine so that no single property decides the outcome.

Table -1: Signal families used by the detection pipeline.

Module	Evidence contributed	Principal limitation
URL structure and lexical analysis	IP-literal host, length, subdomain depth, hyphens, at-sign, percent-encoding, credential-related words, unusual port, character entropy, suspicious top-level domains	The same properties occur in benign URLs; thresholds are heuristic
Brand impersonation	Brand-like tokens in unrelated domains, character substitution, Levenshtein similarity to brand names	Similarity does not establish ownership or intent

Module	Evidence contributed	Principal limitation
Domain intelligence	Trusted-domain exemption; RDAP registration date, from which age is estimated	Depends on network and registry support; missing data is not a safe verdict
Redirect indicators	Known shorteners; redirect-like parameters and patterns	Chains are not followed, so the final destination is unknown
Local threat matching	Exact match of a domain or URL against the local indicator table	Coverage limited to stored records; not a live feed
Experimental classifier	Logistic-regression output over thirteen URL features	Thirty training URLs, no held-out split

Brand impersonation compares host and path text with protected brand names using substitution rules and Levenshtein distance. A small distance in an unrelated registrable domain is meaningful evidence of lookalike construction; a small distance inside a legitimate corporate subdomain is not. The module therefore compares registrable-domain text and reports the matched brand and the transformation.

Domain intelligence contributes two kinds of context. A trusted-domain list can exempt recognised domains from heuristic scoring, reducing warnings on familiar services. An RDAP lookup can return registration events from which age is estimated; a base domain is sent to the registry and the response is cached. Both are policy aids rather than proofs: a trusted domain can be compromised or impersonated through a confusing subdomain, an old domain can be malicious, and a new one can be legitimate. When RDAP is unavailable the age signal is absent and the engine proceeds without it.

Redirect checks identify known shorteners and parameters that resemble redirection instructions. They do not dereference the URL, so the final landing page and intermediate hops remain unknown. A shortener conceals a destination and justifies scrutiny, but it is not evidence of fraud, and the behaviour reported in Section 6 reflects that restraint.

Finally, an experimental logistic-regression classifier converts the URL into thirteen structural and lexical features and returns a probability-like score under its model assumptions. Its role is modest: it is one weighted signal, it is not consulted when the model cannot be loaded, and its output is never shown to the user as a probability of maliciousness.

3.5 Risk Classification and Explainability

The engine applies a fixed order of operations: normalise the URL, check the local threat-indicator table for an exact match, apply any trusted-domain exemption, invoke the available modules, and combine their contributions in a weighted sum

$$S = \Sigma\, w_i \cdot s_i \quad (i = 1, \ldots, n)$$

where w_i is the weight assigned to the i-th signal and s_i is that signal's contribution to the aggregate score.

The weights and band boundaries are configuration constants rather than learned parameters, and no claim is made that they are optimal. The score is mapped to one of the first four categories in Table 2, and the signals that fired are rendered as reasons. An exact match bypasses this path and returns the fifth category.

Table -2: Risk categories and their interpretation.

Category	Interpretation	Basis in the pipeline
Low Risk	Few or no concerning signals; not a guarantee of safety	Weighted result in the lowest band, often with a trusted-domain exemption
Caution	Limited or ambiguous indicators warrant attention	One weak indicator, such as a shortener
Suspicious	Multiple or stronger indicators deserve scrutiny	Several independent indicators combined

Category	Interpretation	Basis in the pipeline
High Risk	A high level of concern without an exact match	Strong signal combination and/or high classifier output
Confirmed Threat	The URL exactly matches a local threat indicator	Exact domain or URL match; never produced by score alone

Confirmed Threat has a different epistemic status from the other four. It is not a stronger High Risk; it states that this exact indicator is recorded as a verified threat. The distinction separates "the evidence resembles known fraud" from "this destination is a known record", and no score can promote a URL into it.

Severity and confidence are reported separately. Severity is the assigned category; confidence is a heuristic description of how much evidence was available and how coherent it was strong when independent signals agree, limited when one weak signal fired, and verified when the verdict rests on a database record. The labels are not calibrated probabilities. A severe verdict can rest on thin evidence, and a low-severity verdict does not establish safety. Figure 2 shows the decision path and the separation.

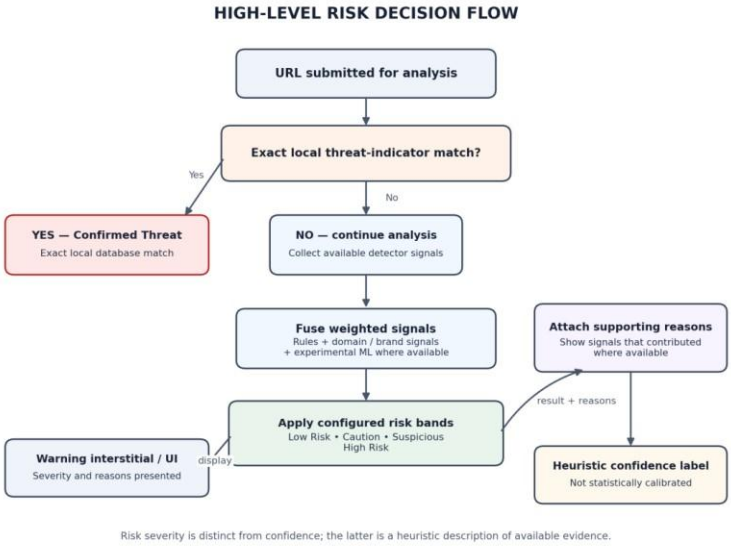


Fig -2: Risk decision flow, including the separate exact-match path and the heuristic confidence label.

Reasons are generated from the signals that entered the decision, so the explanation describes the pipeline rather than supplying a post-hoc narrative. Each reason names an observation and its bearing on the verdict — a brand imitation through character substitution, a top-level domain associated with higher-risk registration, or credential-related terms in the path. A reason states why the system responded as it did; it does not prove that the observation is correct.

3.6 Database Design

Persistence uses two SQLAlchemy models with deliberately different meanings. ThreatIndicator stores the indicator value, its type, threat category and name, verification status, source, description and creation time, and is the only source of exact-match recognition. UserReport stores the submitted URL, the extracted domain, a category, optional comments, a status, a report count and timestamps. Keeping them apart is a design decision rather than a schema detail: it prevents an unreviewed report from acquiring the evidential status of a verified indicator through an incidental code path. A repeat report for the same domain increments a count instead of creating an unrelated duplicate, and no operation promotes a report into the threat table. The database is a local file, initialised at start-up and seeded with demonstration indicators, so its coverage is exactly the set of records it contains.

4 SYSTEM MODULES

4.1 Chrome Extension

The extension follows the Manifest V3 model [16]. A background service worker registers a navigation listener, performs the backend request and owns the fallback; a popup provides a compact interface for the active destination and a manual check; and a warning page renders the interstitial. It uses vanilla JavaScript, HTML and CSS with no build step, declares only the permissions its workflow requires, and does not inject content scripts to read page structure, forms or input fields.

4.2 URL Detection

Detection is decomposed into single-responsibility modules for structure and lexical analysis, brand impersonation, domain intelligence, redirects and the classifier. Each returns a uniform result: a signal identifier, whether it fired, its contribution when applicable, and a human-readable reason. That contract lets the engine treat modules interchangeably and lets the interface explain a verdict without knowing which module produced which observation. A module that cannot complete returns unavailable rather than negative.

4.3 Risk Engine

The engine is the only component that decides a category. It normalises the input, performs the exact-match check, applies trusted-domain handling, invokes the modules and fuses their outputs. Its behaviour is asymmetric at the extremes: an exact match short-circuits to Confirmed Threat, a trusted domain can suppress heuristic scoring, and everything else is a weighted combination followed by band mapping. Because the engine owns both the score and the explanation, the two cannot drift apart as modules are added.

4.4 Warning Interface

The interstitial is the main explanatory surface. It presents the category, the flagged destination, a section titled “Why ScamShield intervened” listing the reasons, the confidence label, and two choices: a recommended return to safety and an explicit, discouraged continuation. The wording of each category is distinct, so a confirmed record is not described in the same terms as a heuristic suspicion. The popup presents the same information in compressed form. A temporary in-memory allowlist lets a user proceed within the session; it is not durable storage and is not presented as a trust decision.

4.5 Community Reporting

The reporting module connects the interface to a submission endpoint: the user selects a category, may add a comment and submits the destination, and the backend stores the report and confirms receipt. Retrieval is available through a companion endpoint. The module captures observations that automated signals miss, but it is deliberately unambitious: reports are not verified, not deduplicated beyond a per-domain count, not moderated and never promoted automatically.

4.6 Local Heuristic Fallback

The fallback exists so that the extension degrades rather than fails. When the backend request cannot be completed, a small set of browser-side rules returns a reduced verdict without access to the threat table, domain-age enrichment or model scoring. It includes a hard-coded demonstration domain that makes the degraded path visible during testing, and its result is labelled as a fallback so that no user is led to believe a full analysis was performed.

5 IMPLEMENTATION

5.1 Frontend and Chrome Extension

The browser component is organised as a service worker, a popup pair of HTML and JavaScript files, and a warning pair of the same kind. The service worker holds the navigation listener, the endpoint configuration and the timeout, and coordinates the transition from analysis to presentation. The popup scans the active tab or a manually entered link and renders the category, reasons and confidence. The warning page reads the flagged URL from its query string, renders the category-specific layout and binds the two choices. Styling is self-contained, and the reason list remains readable when several signals fire at once.

5.2 FastAPI Backend

The backend is a FastAPI application launched through a small entry script. Pydantic v2 schemas validate request and response bodies at the boundary, so malformed input is rejected before it reaches the engine. Six routes constitute the public surface (Table 3), and the generated documentation is available locally while the service runs. The route layer is thin: it validates, delegates to the engine or the database, and returns a typed response.

Table -3: Implemented API endpoints.

Endpoint	Method	Purpose
/api/health	GET	Service status, name and version
/api/analyze	POST	Analyse a URL; return category, reasons and confidence
/api/reports	POST	Store a community report
/api/reports	GET	Retrieve stored community reports
/api/threats	GET	Retrieve locally stored threat indicators
/api/stats	GET	Counts, engine status and supported signal groups

5.3 SQLite Database

SQLAlchemy 2 maps the models to a local SQLite file, and initialisation creates the schema and loads demonstration indicators from a JSON seed file. SQLite suits the prototype because it requires no server, keeps the demonstration self-contained and makes the data set inspectable. It also fixes the limits: there is no replication, no concurrent moderation and no feed synchronisation, so the database is a local artefact rather than a service with independent coverage.

5.4 Detection Pipeline

For each request the engine builds a working representation of the URL, runs the modules that can operate on it, and records each outcome with its reason. Exact matching is attempted first, because a database record is the strongest evidence the prototype can hold. Domain intelligence runs next, since a trusted-domain decision changes how later signals should be read. Structural, brand and redirect checks follow, and the classifier is consulted last, with its output treated as one contribution. A module that raises or times out yields an unavailable signal, and the response records that the analysis was partial rather than silently omitting it.

5.5 Experimental Machine-Learning Component

The classifier is a scikit-learn pipeline that standardises thirteen URL features and fits a logistic regression with C = 1.0. It attempts to load a trained model at start-up and, if loading fails, substitutes a hand-coded heuristic score rather than treating the absence of a model as a benign result, which keeps the endpoint functional while making the degradation observable. The training set contains thirty hard-coded URLs, sixteen labelled legitimate and fourteen labelled phishing, and there is no held-out split. Its probabilities therefore reflect a fitted boundary over thirty examples, not an estimated real-world likelihood, and no accuracy, precision, recall or F1 figure is reported because no protocol capable of supporting one has been executed. The component demonstrates how a learned signal can be integrated without displacing the transparent rules.

5.6 Security and Privacy Considerations

The privacy posture follows from the analysis boundary. Only the destination URL is sent to the backend, and only a base domain is sent to RDAP for an age lookup. The extension does not read the DOM, form values, passwords, cookies, local storage or browsing history, and it keeps no navigation log. These flows are narrow but not zero: a destination URL can itself be sensitive, and an operator of the backend sees every analysed address unless the service is run locally, as it is here.

The prototype also carries known gaps that must be closed before public deployment. The reporting endpoints have no authentication and no rate limiting, the CORS policy is permissive, and communication is plain HTTP on the local host.

Input validation is present at the boundary and was strengthened during testing, but it is not a substitute for access control. These limitations delineate the work required to move from a demonstration to a service.

6 RESULTS

6.1 Testing Environment

Three sources of evidence were used and are reported separately because they differ in method and in what they can establish. A deterministic audit harness exercised six URL vectors with the database connection disabled and SQLAlchemy mocked, so that engine behaviour could be observed without persistence. Documented manual checks in a Windows and Chrome environment covered starting the backend, opening the health endpoint and the generated documentation, loading and pinning the extension, scanning destinations, opening the interstitial, and submitting and retrieving a report. An automated pytest suite ran against the FastAPI application through its test client, with each test on a freshly created temporary SQLite database seeded by the project's own initialisation routine and with the RDAP lookup replaced by a stub that always fails.

The suite ran on Linux with Python 3.13.18, pytest 9.2.0, FastAPI 0.162.0, Starlette 1.10.0, SQLAlchemy 2.1.8, Pydantic 2.14.2 and scikit-learn 1.9.1 a different environment from the Windows machine used for browser checks. Because RDAP was stubbed to fail, the domain-age signal was not exercised, and because the suite runs in-process it does not test the extension, navigation timing or the fallback. The sources are therefore not combined into a single pass rate.

6.2 Functional Testing

Table 4 reports the six deterministic vectors. The trusted-domain vectors returned Low Risk with a score of −0.60 and a strong evidence label; the shortener returned Caution with a score of 0.20 and limited evidence; the brand-keyword domain returned Suspicious; and both the character-substitution domain and the raw-IP host with a credential path returned High Risk. Scores and model outputs are reproduced as observed for these inputs and are not calibrated probabilities.

Table -4: Functional verification matrix for the six deterministic vectors.

ID	Input URL	Observed category	Contributing signals	Score / model output
T1	https://en.wikipedia.org/wiki/Phishing	LOW_RISK	Trusted-domain exemption	Score −0.60; strong evidence
T2	https://www.paypal.com/signin	LOW_RISK	Trusted-domain exemption	Score −0.60; strong evidence
T3	https://bit.ly/sample-link	CAUTION	Shortener indicator	Score 0.20; limited evidence
T4	http://paypal-account-support.com/login	SUSPICIOUS	Brand keyword; multiple indicators	Model output 0.44; score 0.70
T5	http://paypa1-security-login.xyz/account/signin	HIGH_RISK	Suspicious TLD, credential keywords, entropy, brand substitution	Model output 0.96; score 2.35
T6	http://192.168.1.150/bank/login.php	HIGH_RISK	IP-address host, credential keywords	Model output 0.76; score 1.20

The harness had no active database connection, so the Confirmed Threat path was not executed there. It was exercised instead by the automated suite against a seeded temporary database: a seeded domain and a seeded exact-URL indicator both returned Confirmed Threat with the blocking flag set, and deleting the row removed the verdict. The category is therefore produced by the stored record rather than a hard-coded rule, although the record itself is a local demonstration entry rather than a live intelligence feed.

6.3 Automated API Testing

The suite contains 56 tests in eight groups, summarised in Table 5. Its first execution produced 53 passes and 3 failures, and the failures were genuine defects rather than incorrect expectations. The reporting endpoint accepted a blank or whitespace-only URL with HTTP 200 and stored an empty domain, whereas the analysis endpoint rejected the same input with HTTP 400; and a repeat report for an already-reported domain with a null comment appended the literal text "[Update]: None". Both were corrected in the route implementation: a blank URL now returns HTTP 400, and a comment is appended only when one is supplied. The tests were not altered, and the corrected build reported 56 passed with no failures or skips.

Table -5: Automated API test summary by behaviour group.

Test group	Behaviour checked	Tests
Health and routing	Status, service name and version; unsupported methods and unknown routes rejected	3
Analysis: normal and low risk	Trusted domains return LOW_RISK; shortener returns CAUTION; scheme-less input accepted; whitespace stripped; full schema returned	6
Analysis: suspicious and high risk	Brand-keyword, character-substitution and raw-IP vectors return the expected categories; high risk never becomes CONFIRMED_THREAT	4
Analysis: seeded threat database	Seeded domain and exact-URL indicators return CONFIRMED_THREAT with the blocking flag; deleting the row removes the verdict	3
Analysis: invalid input	Blank URLs return 400; schema violations 422; unsupported methods 405; eight malformed URLs return well-formed responses and write nothing	18
Community reports	Submission, retrieval, domain extraction, duplicate counting, comment handling, no promotion, limit parameter and validation errors	15
Threats and statistics	Listing and limits; counts, engine status, signal groups and updates after new reports	5
Documented limitations	CORS accepts an arbitrary origin; the report endpoint accepts unauthenticated submissions	2
Total	**Run result: 56 passed, 0 failed, 0 skipped**	**56**

Two tests confirm documented limitations rather than removing them: the CORS configuration accepts an arbitrary origin, and the reporting endpoint accepts unauthenticated submissions. Eight malformed or unusual strings — including an unterminated IPv6 literal, free text, an empty host, a space in the host, a non-HTTP scheme, an over-long label, embedded credentials and a javascript: URI each returned a well-formed response rather than a server error. These results establish consistency of API behaviour, not detection accuracy.

6.4 Test Results and Observations

The captured interface states are consistent with the underlying responses. The popup on a trusted destination reports a Low Risk verdict, a strong evidence label, the reason that the destination belongs to a recognised organisation, and the available actions. Figure 3 shows the interstitial for the character-substitution vector, where three independent reasons are listed: brand imitation through character substitution, a top-level domain associated with higher-risk registration, and credential-related terms in the path. Both surfaces draw on the same response, so the reasons in the warning are exactly those the engine returned.

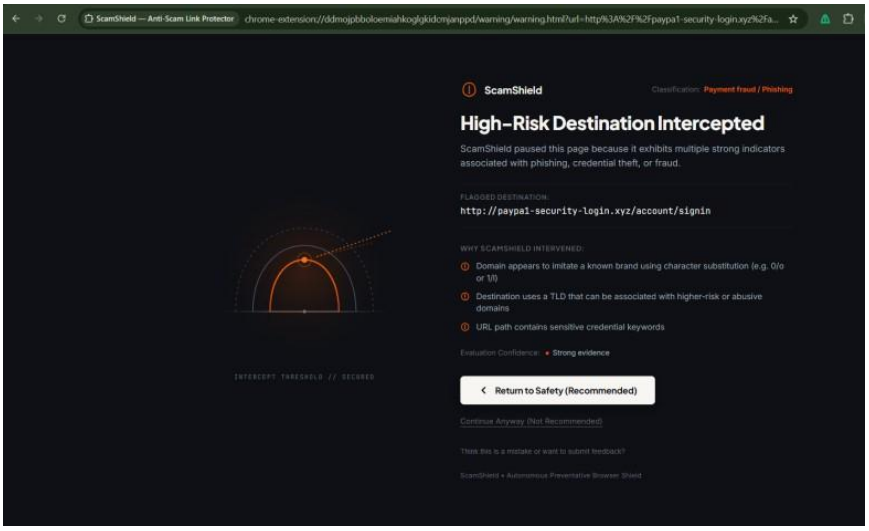


Fig -3: Warning interstitial for a high-risk destination, listing the three contributing reasons and the confidence label.

The statistics and retrieval endpoints reflected the database at capture time: four threat indicators, five stored reports, an active engine and six declared signal groups. Each threat record carried an indicator value, type, threat category, name, verification status, source label and description; the source labels are stored text from the seeded data and do not indicate that the prototype queries an external feed. The five reports each carried the status "Reported" and a count of one, including a development report for a browser-internal page filed under a false-warning category. No report appeared in the threat table, which is the intended separation.

Three observations follow. First, the trusted-domain exemption dominates the low end of the range: both trusted vectors scored −0.60 and neither produced an interstitial, which is the desired behaviour and also the point of greatest risk if the list is wrong. Second, the elevated verdicts arose from combinations rather than single features — T5 combined four families, and its classifier output alone would not have produced the same confidence. Third, the single shortener indicator produced Caution rather than a higher category, which is the proportionate response for a destination that is merely opaque.

7 DISCUSSION

The results support a narrow claim. For the vectors tested, the pipeline produced the intended categories, the explanation listed the signals that were actually consulted, and the API returned consistent responses across normal, malformed and adversarial input. They do not support a claim about detection accuracy. Six deterministic vectors are a demonstration of behaviour, not a sample; the automated suite validates endpoint semantics against the project's own expectations, not verdicts against adjudicated ground truth; and the classifier was neither trained nor evaluated in a way that could support a performance figure.

The most consequential decision is the separation of Confirmed Threat from the weighted bands. A single score invites a single interpretation, and users reasonably read a high score as a statement of fact. Reserving the strongest label for an exact match keeps three claims apart: that a record exists, that the evidence resembles known fraud, and that the evidence is weak but present. The cost is coverage, because an unlisted malicious destination can never receive the strongest label; the benefit is that the label means something specific and can be audited against the database.

Separating severity from confidence addresses a related failure. A system that reports only severity implicitly claims uniform certainty, and users learn to discount the label when it is wrong. Reporting confidence as a separate, explicitly heuristic quantity makes the evidential basis visible: a High Risk verdict supported by four independent signals reads differently from one supported by a shortener, even though both receive the same category. The implementation does not calibrate these labels, and the interface says so.

The trusted-domain exemption deserves scrutiny because it is the component most likely to produce a confident false negative. An exemption is a policy statement about a registrable domain, not a verification of the specific URL, and it can be defeated by a confusing subdomain or a compromised host. In the tested vectors it produced the correct outcome, but it

trades coverage for usability and depends on list maintenance and correct host parsing. The prototype does not implement public-suffix-aware registrable-domain extraction, which is a correctness gap for domains such as co.uk and for encoded or internationalised forms.

The privacy posture is coherent but not absolute. Confining analysis to the URL avoids the largest intrusion associated with content-based detection and removes the need to execute hostile code, but it does not make the system local: every analysed destination reaches the backend, and every age lookup reaches a registry. A deployment serving many users would accumulate a log of visited destinations unless retention is deliberately constrained. Running the service locally reduces the exposure, but the design should not be described as anonymous.

Community reporting is presented as a data path rather than a capability. Its value depends on volume, moderation and provenance, none of which the prototype supplies. Keeping reports out of the threat table is nonetheless the right default, because an unmoderated report is an assertion by an unidentified party, and promoting such assertions automatically would let anyone who can reach the endpoint produce the strongest label in the system — an endpoint that has neither authentication nor rate limiting. Finally, the asynchronous navigation model bounds what a verdict can achieve: a warning delivered after a page has begun to load still helps a user who has not yet acted, but the prototype cannot claim to prevent the destination from loading.

8 CONCLUSION AND FUTURE WORK

ScamShield demonstrates that a small, transparent pipeline can produce browser warnings that state their own basis. A Chrome Manifest V3 extension submits destination URLs to a FastAPI service, which combines structural and lexical analysis, brand-impersonation checks, RDAP-derived age, redirect indicators and an experimental classifier through a weighted risk engine. The engine returns one of five categories with the reasons that produced it, reports evidence confidence separately from severity, keeps community reports distinct from verified indicators, and falls back to reduced local rules when the backend is unavailable. The extension analyses addresses only, and the privacy boundary is stated in terms of what it does and does not inspect.

Evaluation comprised six deterministic vectors, documented manual browser checks and a 56-test automated API suite. The suite exposed two input-handling defects in the reporting endpoint, both corrected before the final run, and confirmed that Confirmed Threat is driven by a stored record rather than a rule. The same evidence delimits the work: the classifier was trained on thirty URLs without a held-out split, the threat database is local and seeded, redirect chains are not followed, navigation handling is asynchronous, and the fallback and the extension itself are not covered by automated tests. No accuracy, precision, recall or F1 figure is claimed.

The following directions would strengthen the work:

- benchmark evaluation against independently sourced phishing and benign URL datasets, with documented provenance and separate training, validation and test partitions;
- standard performance measurement under a repeatable protocol, reporting precision, recall, F1, false-positive and false-negative rates, class-wise results and latency against stated baselines;
- threat-intelligence provenance, recording source, observation time and verification time for each indicator, with explicit expiry policies;
- a review workflow with abuse controls, duplicate handling and moderation before any report becomes an indicator;
- API hardening through access control, rate limiting, restrictive CORS and a documented retention policy;
- public-suffix-aware URL canonicalisation covering encoded characters, internationalised names, ports and trailing dots;
- systematic testing of navigation timing and of the fallback across failure modes, so that any stronger blocking claim rests on evidence; and
- local QR-code decoding routed through the existing pipeline, followed by structured usability testing of the category, reason and confidence presentation.

These steps would move the prototype from a demonstration of an explainable design to a measured one. The present contribution is the design itself and the evidence that its components operate as intended: a browser warning that names its evidence, distinguishes severity from certainty, and declines to claim more than its data can support.

ACKNOWLEDGMENT

The authors express their sincere gratitude to the management of Hyderabad Institute of Technology and Management for the facilities and academic environment required for this project. We are grateful to the Department of Computer Science and Engineering, the Program Head, the PBL-I Co-ordinator and the faculty members for their guidance and feedback, and we thank our guide, Mr A. Vishwanath, and our families, friends and peers for their support during the development and documentation of ScamShield.

REFERENCES

[1] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, "Identifying Suspicious URLs: An Application of Large-Scale Online Learning," in Proc. 26th Int. Conf. Machine Learning (ICML), 2009, pp. 681–688. doi:

10.1145/1553374.1553462.

[2] H. Le, Q. Pham, D. Sahoo, and S. C. H. Hoi, "URLNet: Learning a URL Representation with Deep Learning for Malicious URL Detection," arXiv:1802.03162, 2018.

[3] O. K. Sahingoz, E. Buber, O. Demir, and B. Diri, "Machine Learning Based Phishing Detection from URLs," Expert Systems with Applications, vol. 117, pp. 345–357, 2019. doi: 10.1016/j.eswa.2018.09.029.

[4] S. Garera, N. Provos, M. Chew, and A. D. Rubin, "A Framework for Detection and Measurement of Phishing Attacks," in Proc. ACM Workshop on Recurring Malcode (WORM), 2007, pp. 1–8. doi:

10.1145/1314389.1314391.

[5] D. Canali, M. Cova, G. Vigna, and C. Kruegel, "Prophiler: A Fast Filter for the Large-Scale Detection of Malicious Web Pages," in Proc. 20th Int. Conf. World Wide Web (WWW), 2011, pp. 197–206. doi: 10.1145/1963405.1963436.

[6] Y. Zhang, J. I. Hong, and L. F. Cranor, "Cantina: A Content-Based Approach to Detecting Phishing Web Sites," in Proc. 16th Int. Conf. World Wide Web (WWW), 2007, pp. 639–648. doi: 10.1145/1242572.1242659.

[7] C. Whittaker, B. Ryner, and M. Nazif, "Large-Scale Automatic Classification of Phishing Pages," in Proc. Network and Distributed System Security Symposium (NDSS), 2010.

[8] S. Sheng, B. Wardman, G. Warner, L. F. Cranor, J. Hong, and C. Zhang, "An Empirical Analysis of Phishing Blacklists," in Proc. 6th Conf. Email and Anti-Spam (CEAS), 2009.

[9] Google for Developers, "Safe Browsing API." [Online]. Available: https://developers.google.com/safe-browsing/reference/rest

[10] J. Sunshine, S. Egelman, H. Almuhimedi, N. Atri, and L. F. Cranor, "Crying Wolf: An Empirical Study of SSL Warning Effectiveness," in Proc. 18th USENIX Security Symposium, 2009, pp. 399–416.

[11] A. P. Felt, R. W. Reeder, A. Ainslie, H. Harris, M. Walker, C. Thompson, M. E. Acer, E. Morant, and S. Consolvo, "Rethinking Connection Security Indicators," in Proc. 12th Symposium on Usable Privacy and Security (SOUPS), 2016, pp. 1–14.

[12] S. Egelman, L. F. Cranor, and J. Hong, "You've Been Warned: An Empirical Study of the Effectiveness of Web Browser Phishing Warnings," in Proc. SIGCHI Conf. Human Factors in Computing Systems (CHI), 2008, pp. 1065–1074. doi: 10.1145/1357054.1357219.

[13] A. Newton, "JSON Responses for the Registration Data Access Protocol (RDAP)," RFC 9083, IETF, Jun. 2021.

doi: 10.17487/RFC9083.

[14] V. I. Levenshtein, "Binary Codes Capable of Correcting Deletions, Insertions, and Reversals," Soviet Physics Doklady, vol. 10, no. 8, pp. 707–710, 1966.

[15] M. T. Ribeiro, S. Singh, and C. Guestrin, "Why Should I Trust You?: Explaining the Predictions of Any Classifier," in Proc. 22nd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD), 2016, pp. 1135–1144. doi: 10.1145/2939672.2939778.

[16] Google Chrome Developers, "Manifest V3," Chrome Extensions Documentation. [Online]. Available:

https://developer.chrome.com/docs/extensions/develop/migrate