International Journal of Creative and Open Research in Engineering and Management
ISSN: 3108-1754 (Online)
Volume 02 Issue 10 October-2026 | Impact Factor: 3.5



Cyber Sheild – An Intelligent Browser Activity Monitoring and Threat Analyzer

CHETHAN KUMAR M
Department of CSE
VTU University
Belagavi, India
chethankumarm987@gmail.com

HEMANTH GOWDA T M
Department of CSE
VTU University
Belagavi, India
hemanthgowdatm453@gmail.com

KIRAN K R
Department of CSE
VTU University
Belagavi, India
kirankirukr2814@gmail.com

JEEVAN KUMAR B R
Department of CSE
VTU University
Belagavi, India
E-mail

Guide : **Dr. Pushpa R** ph.D.,
Professor and Head , Department of CSE , VTU University Belagavi , India
pushpa.aitcse@gmail.com

ABSTRACT

How to Cite this Article:

M, C. K., M, H. G. T., R, K. K. & R, J. K. B. (2026). Cyber Sheild – An Intelligent Browser Activity Monitoring and Threat Analyzer. International Journal of Creative and Open Research in Engineering and Management, 02(10), 1-9. https://doi.org/10.55041/ijcope.v2i10.218

.

License:

This article is published under the terms of the Creative Commons Attribution 4.0 International License (CC BY 4.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original author(s) and the source are credited.

© The Author(s). Published by International Journal of Creative and Open Research in Engineering and Management.







https://doi.org/10.55041/ijcope.v2i10.218

Routine web browsing regularly exposes users to phishing domains, deceptive redirects, and other malicious web entities, yet stored browser histories give no indication of the risk those visits carried. This paper presents Cyber Shield, a browser activity monitoring and threat analysis suite that audits historical Google Chrome browsing records and assigns each visited URL a risk estimate. The system pairs a React and Vite front end with a Django REST Framework backend and keeps session logs and analysis results in SQLite. On explicit administrative request, a collector extracts URL history from the local Chrome profile database. Each link is evaluated by a rule-based scoring engine that flags unencrypted HTTP, suspicious keywords, raw IP hostnames, irregular domain formatting, and excessive URL length. A supervised classifier built on character-level TF-IDF vectorization and logistic regression adds a complementary phishing prediction, and, when API credentials are supplied, Virus Total reputation data is retrieved. The web dashboard presents aggregate statistics, risk classifications, and recent activity, and the platform supports credential management, log filtering, persistent threat records, and PDF audit reports. The implemented capabilities are demonstrated through interface snapshots and a sample report of 35 records. The outputs are probabilistic risk estimates and not guarantees of safety, and the machine learning component has not yet been benchmarked because the supplied training dataset is too small.

INDEX TERMS : Cybersecurity, Browser Logs, Threat Assessment, Phishing Identification, Supervised Learning, React, Django REST API, SQLite.



I. INTRODUCTION

The global digital network provides unprecedented access to collaborative platforms and digital commerce, but it concurrently exposes users to fraudulent portals, phishing links, and deceptive redirects aimed at credential harvesting and data theft. While client-side web browsers naturally archive records of navigated web addresses, these unprocessed browser histories fail to convey the inherent risks of visited destinations, leaving users largely unaware of past exposures. Manual inspection of such vast digital footprints is highly inefficient and lacks methodical risk quantification, creating a significant gap in proactive user security awareness.

To address this challenge, this paper presents a unified cybersecurity diagnostic tool engineered to audit historical browsing trajectories and compute potential risk indices. The proposed solution establishes a decoupled full-stack architecture, bridging a reactive React client interface with a secure Django REST API backend and SQLite persistence. Rather than functioning as a continuous real-time intrusion prevention system (IPS), the platform operates as a post-navigation diagnostic utility that safely extracts stored artifacts from local Google Chrome environments upon explicit administrative command.

Once the historical logs are harvested, the system evaluates the web addresses using a rigorous, multi-layered approach. It applies an algorithmic heuristic evaluation engine to classify URLs based on predefined vulnerability markers, alongside a predictive machine learning pipeline utilizing TF-IDF vectorization and logistic regression for intelligent phishing detection. By integrating these local evaluations with external threat intelligence from the VirusTotal API, the system centralizes log analysis within an interactive dashboard. Ultimately, this tool drastically improves auditing efficiency and demonstrates the practical advantages of combining applied machine learning, algorithmic heuristics, and stateless RESTful API design to enhance digital hygiene.

II. LITERATURE SURVEY

Recent research highlights both the progress and the ongoing challenges in identifying malicious websites. Tian et al. [1] note that while machine learning models are powerful in theory, they often struggle when deployed in real-world applications because they lack practical administrative tools and verifiable auditing. To better understand these detection methods, Orozco Fonseca et al. [2] categorized systems into AI-driven and rule-based approaches, pointing out that cybercriminals easily bypass static rules using URL obfuscation. Furthermore, Aljabri et al. [4] showed that traditional machine learning algorithms can effectively catch malicious links using lexical features, but they still struggle to adapt to new threats without being paired with external threat intelligence.

To improve detection rates without overloading computer systems, researchers have explored various extraction methods. Safi and Singh [3] reviewed advanced deep learning models like Convolutional Neural Networks (CNNs) for phishing detection. While these models are highly accurate, they are computationally expensive and act as "black boxes" that cannot easily explain why a URL was flagged. As an alternative, Khan et al. [7] demonstrated that using generic, lightweight lexical features can successfully classify malicious URLs without the need for deep webpage payload inspection. This lightweight approach reduces computational overhead and allows for much faster real-time screening.

Beyond real-time detection, post-incident forensic tools play a vital role in auditing web security. Dija et al. [6] investigated offline browser forensics, showing how sensitive user activity and searched keywords can be recovered by querying hidden SQLite databases used by browsers like Google Chrome. However, extracting this data safely is critical. Janloy and Boonyopakorn [5] benchmarked various forensic tools against international standards, concluding that automated extraction is highly effective for investigations only if it can process historical logs without inadvertently modifying or locking the active source database.

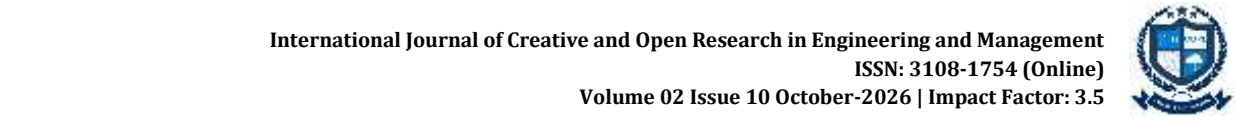
III. SYSTEM ARCHITECTURE

A. REQUIREMENTS AND TECHNOLOGY STACK

The suite runs on standard consumer hardware (a multi-core 64-bit CPU and 8 GB of RAM recommended) with broadband access for package installation and reputation lookups. The client uses ReactJS, Vite, JavaScript, Tailwind CSS, Axios, and React Router. The server uses Python 3, Django, Django REST Framework, SQLite, scikit-learn, and ReportLab, with the Virus Total API and Google Chrome as the history source. The system must support registration and credential validation, dashboard tiles, filtered and paginated history, heuristic and ML evaluation, optional reputation lookup, staff-only history collection, persistent results, PDF generation, and a public health-check endpoint. Non-functional goals are usability, maintainability, reliability, security, performance, and transparency through explicit disclaimers that outputs are risk estimates

B. ARCHITECTURE OVERVIEW

As shown in Fig. 1, the system follows a strictly decoupled client–server model. The React front end issues asynchronous HTTP requests to the Django REST backend, which validates input, runs the business services, interacts with SQLite, and replies with JSON payloads or binary PDF streams. Fig. 1 is a logical



blueprint and not a network packet trace. The front end is organized into page-level views for authentication, dashboard, history, reports, and settings, built from reusable components. The back-end monitor application holds the models, permissions, views, and a service layer that separates snapshot-based history querying, heuristic scoring, ML inference, reputation querying, and PDF rendering.

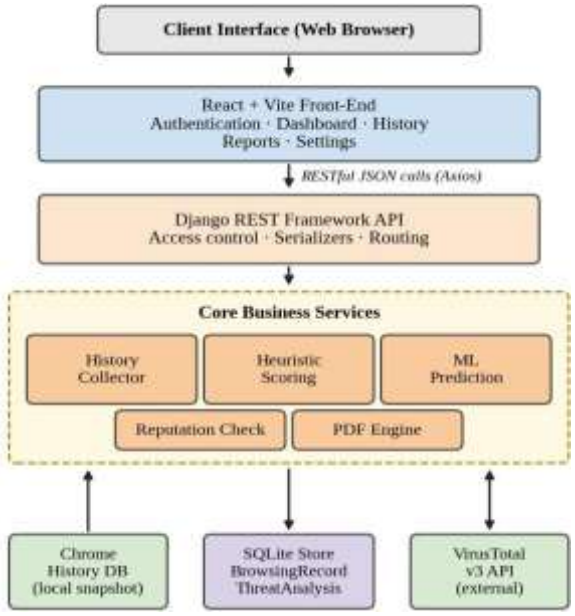


FIGURE 1. Logical architecture of the proposed Cyber Shield application

C. DATABASE AND API DESIGN

Persistence uses SQLite with two entities. BrowsingRecord stores the URL, page title, visit time, risk score, status, ingestion source, and creation time, with a uniqueness constraint on URL and visit timestamp to prevent duplicates across extractions. ThreatAnalysis references its parent record through a cascading foreign key and stores the risk score, status, heuristic reasons, reputation source and data, and analysis timestamp. Table I lists the REST endpoints exposed through Django routers.

TABLE I

BACKEND API ENDPOINTS

Verb	Endpoint	Purpose
GET	/api/health/	Service uptime and health check
POST	/api/auth/register/	New user registration
POST	/api/auth/login/	Credential validation, token issuance
GET	/api/dashboard/	Aggregated metrics and statistics
GET	/api/history/	Paginated browsing log retrieval
POST	/api/analyze/	Synchronous heuristic and ML evaluation
POST	/api/reputation/	Cloud reputation intelligence query
POST	/api/collect/	Administrative history harvesting trigger
GET	/api/reports/download/	Binary PDF report stream
POST	/api/analysis/save/	Persist analysis findings

IV. METHODOLOGY

A. WORKFLOW AND CHROME HISTORY COLLECTION

A privileged administrator triggers collection from the settings panel. The service targets Windows Chrome installations through the default App Data profile directory. To avoid file-lock conflicts while Chrome is running, it first creates a SQLite backup snapshot and only then reads from it. It joins the internal URL and visit tables, keeps only HTTP and HTTPS schemes, converts WebKit epoch timestamps into time zone-aware datetimes, and skips entries that already exist. Each harvested link then passes through the heuristic filters and is stored; manual queries can additionally invoke the ML model and the Virus Total lookup.

B. RULE-BASED THREAT ANALYSIS

The scoring engine tests each URL against ten criteria: malformed or deeply nested structure; plain-text HTTP; an @ symbol in the authority field; a numeric IP hostname; Punycode sequences; excessive length; excessive subdomain nesting; known URL-shortener domains; suspicious path keywords; and embedded user-info syntax. Triggered rules add penalty points up to a cap of 100, mapped to the classes in Table II. These are heuristic indicators and not statistically verified probabilities of malicious intent.

TABLE II

RULE-BASED RISK CLASSIFICATION THRESHOLDS

Classification	Score range	Analytical meaning
Low Risk	0–24	Minimal or negligible indicator flags detected
Suspicious	25–49	Moderate presence of anomalous traits
High Risk	50–100	Multiple severe vulnerability markers identified

C. MACHINE-LEARNING PREDICTION

The supervised module is a scikit-learn pipeline combining character n-gram TF-IDF vectorization (3 to 5 characters) with logistic regression. Training reads a



CSV file of URLs and class labels, validates that both classes (legitimate and phishing) are present, enforces a minimum sample size, applies a stratified train–test split when possible, and serializes the model with joblib. The sample dataset supplied with the project is too small for robust training, so a larger, accurately labeled corpus is required before production-grade evaluation can be claimed.

D.REPUTATION,AUTHENTICATION AND REPORTING

Reputation checks use the VirusTotal v3 API with a key stored in an environment variable, handle timeouts, missing credentials, and malformed JSON, and look up existing records instead of forcing deep scans to conserve quota. Because URLs are sent to an external provider, this should be disclosed to users. User management uses Django's built-in authentication with hashed passwords, and history collection is restricted to staff accounts; production use would require review of token handling, TLS, password policies, and authorization rules, for example against the OWASP Web Security Testing Guide. ReportLab builds a landscape A4 PDF containing aggregate counts, the average risk score, a browsing-record table, and a mandatory disclaimer, with path-traversal checks on output directories.

V. IMPLEMENTATION AND TESTING

The Login page provides credential inputs, a registration toggle, and client-side password-match validation that complements server-side checks. The Dashboard shows tiles for total records, safe sites, threats, and average risk, a recent-activity feed, and a manual URL-analysis box. The History page offers searchable, filterable tables, the Reports page downloads server-generated PDF audits, and the Settings page exposes system health and the administrator-only collection trigger. On the server, Django routing maps API paths to views, serializers produce JSON, and a permissions module combines authenticated-or-read-only access with a staff-user permission.

Automated tests use Django's TestCase with DRF's APIClient and targeted mocks. Thirteen test cases cover HTTPS and HTTP URL scoring, the health endpoint, registration (HTTP 201), bad-password login (HTTP 401), anonymous dashboard access, dashboard statistics, history retrieval, combined heuristic and ML analysis, invalid-scheme rejection (HTTP 400), saving analyses, and collection as a standard user (HTTP 403) and as staff. They run with python manage.py test monitor. At the time of writing, execution results have not been recorded, so these cases are specifications and not reported outcomes.

VI. RESULTS AND DISCUSSION

A. IMPLEMENTED CAPABILITIES AND SAMPLE OUTPUT

The repository implements a complete full-stack application with authentication, dashboard, logs, reports, and settings views; a DRF backend; SQLite persistence; rule-based scoring; a modular TF-IDF and logistic regression component; Virus Total integration; staff-restricted Chrome history extraction; and landscape PDF generation. These results describe implemented functionality and not verified benchmark performance. Fig. 2 shows the dashboard for the sample data: 35 stored records, 29 low-risk records, 6 threats (suspicious or high risk), and an average risk score of 11.0%, with the quick URL-analysis tool. Fig. 3 shows the recent-browsing table and threat alerts, where flagged entries include raw-IP URLs and a lookalike PayPal login address. The PDF report (Fig. 4) summarizes the same 35 records as 29 Low Risk, 5 Suspicious, and 1 High Risk, with an average score of 11%. These figures illustrate system output on a small sample and are not accuracy measurements.

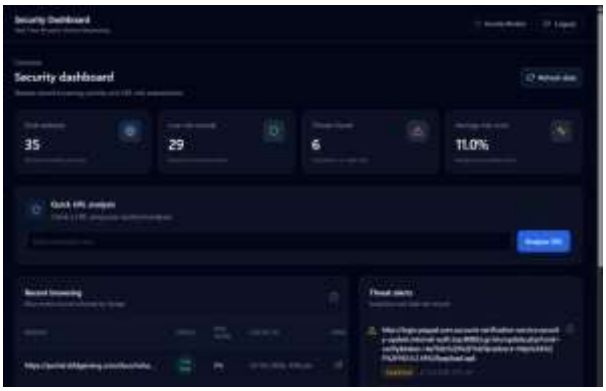


FIGURE 2. Security dashboard with overall metrics and quick URL analysis tool

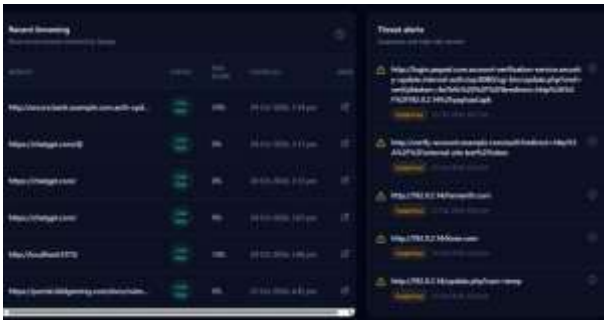


FIGURE 3. Recent browsing table and threat alerts panel



FIGURE 4. Generated PDF security report with summary and browsing records

B. *INTERPRETATION AND ML EVALUATION*

Because the engine reports the rules behind each score, administrators can see why an alert was raised. However, benign sites sometimes use long query strings or legacy HTTP, while sophisticated malicious domains often use valid HTTPS and ordinary-looking names. In the sample output, a URL beginning with secure.bank.example.com scored 20% and was labeled Low Risk, below the 25-point threshold of Table II, so heuristic scores are best treated as preliminary screening indicators. The ML pipeline is fully integrated, but the sample CSV has too few examples and too little class variety, so accuracy, precision, recall, F1-score, and confusion-matrix results cannot be reported here. A scheduler module also exists, but automated periodic collection depends on startup configuration and operating-system service management and should not be described as a running background service unless verified.

VII. LIMITATIONS

The current implementation depends on hardcoded Windows Chrome profile paths, so other browsers and UNIX-based systems need adaptation. It parses stored database files and is not a browser extension, so it cannot intercept real-time navigation, and ingestion is manual unless scheduling is configured. Rule-based scoring yields false positives on complex legitimate sites and false negatives on clean-looking phishing domains, and the sample dataset is too small to train the classifier effectively. Virus Total lookups need network access, are rate limited, and depend on existing reputation data. Browsing logs contain sensitive personal information and need encryption and role-based access control, and the authentication layer requires a security review before enterprise deployment.

VIII. CONCLUSION AND FUTURE WORK

This paper presented Cyber Shield, which combines local Chrome history extraction, transparent heuristic scoring, a modular TF-IDF and logistic regression classifier, optional Virus Total lookup, and PDF reporting behind a React and Django REST dashboard with SQLite persistence. It serves as a diagnostic risk-screening assistant, provided users understand that its scores are estimates and not guarantees of safety. Future work includes support for more browsers and operating systems, a browser extension for real-time monitoring, a verified background service, training and benchmarking the classifier on a large labeled corpus with precision, recall, and F1-score on a held-out set, additional feeds such as Phish Tank and Open Phish, two-factor authentication and stronger authorization, database indexing, audit logging with retention policies, and CSV and Excel report export.

REFERENCES

[1] Tian, Y., Sun, X., Wang, Y., & et al. (2025). From past to present: A survey of malicious URL detection techniques, datasets and code repositories. Computer Science Review, 58, 100810. https://doi.org/10.1016/j.cosrev.2025.100810.

[2] Orozco Fonseca, D., et al. (2024). Taxonomy of Malicious URL Detection Techniques. In: Cybersecurity and Privacy in the Digital Age, Springer, 135–152. https://doi.org/10.1007/978-3-031-54235-0_7.

[3] Safi, A., & Singh, S. (2023). A systematic literature review on phishing website detection techniques. Journal of King Saud University- Computer and Information Sciences, 35(4), 590-611. https://doi.org/10.1109/ACCESS.2022.3222307

[4] Aljabri, M., Mirza, S., Qabil, I. B., Iliyasu, A. M., & et al. (2022). Detecting Malicious URLs Using Machine Learning Techniques: Review and Research Directions. IEEE Access, 10, 121516-121535. https://doi.org/10.1109/ACCESS.2022.3222307



[5] Janloy, K., & Boonyopakorn, P. (2022). The Comparison of Web History Forensic Tools with ISO and NIST Standards. 2022 37th International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC), 1–4. https://doi.org/10.1109/itc-cscc55581.2022.9894903.

[6] Dija, S., Ajana, J., Indu, V., & Sabarinath, M. (2021). Web Browser Forensics for Retrieving Searched Keywords on the Internet. 2021 3rd International Conference on Advances in Computing, Communication Control and Networking (ICAC3N). https://doi.org/10.1109/icac3n53548.2021.9725457.

[7] Khan, S., et al. (2019). Identifying Generic Features for Malicious URL Detection System. 2019 International Conference on Computing, Mathematics and Engineering Technologies (iCoMET), 1–6. https://doi.org/10.1109/UEMCON47517.2019.8992930.