ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue I, January 2026

"Securing the Browser: A Hybrid Static Analysis Framework for Detecting Malicious Chrome Extensions via Local Threat Intelligence"

Vikas Mishra

Assistant Professor

DOI: https://doi.org/10.51583/IJLTEMAS.2026.1501000102

Received: 07 February 2025; Accepted: 12 February 2026; Published: 18 February 2026

ABSTRACT

Modern web browsers have evolved into sophisticated platforms where extensions play a crucial role in enhancing user productivity and customization. However, this extensibility significantly increases the attack surface, as malicious extensions often abuse excessive permissions to harvest cookies, manipulate the Document Object Model (DOM), and exfiltration sensitive user data. Existing browser sandboxing mechanisms provide limited visibility into such threats, while cloud-based detection approaches raise privacy concerns.

This paper proposes a hybrid static analysis framework for detecting malicious Google Chrome extensions that preserves user privacy while enabling deep security inspection. The detection logic is decoupled from the browser environment and implemented as a Python-based local analysis service, which securely communicates with a lightweight Chrome extension frontend via Restful APIs.

The framework employs a dual-layer detection strategy:

- a weighted permission risk scoring model to assess privilege abuse potential, and
- Signature-based correlation against a local threat intelligence repository containing known malicious patterns and indicators of compromise. Experimental results demonstrate that the proposed approach improves detection accuracy and significantly reduces false positives compared to standalone permission-based techniques, offering an effective and privacy-preserving defense for end-users.

Keywords: Browser Security; Malicious Chrome Extensions; Hybrid Static Analysis; Permission Risk Assessment; Local Threat Intelligence

INTRODUCTION

In the contemporary digital ecosystem, web browsers have evolved from simple document viewers into complex operating platforms. Central to this evolution is the "extensible web" architecture, which allows third-party developers to augment browser capabilities via extensions. Google Chrome, commanding the largest market share, hosts a vast repository of extensions that enhance productivity, accessibility, and content management. However, this architectural openness has introduced a critical attack surface.

The core security model of Chrome extensions relies on a permission-based system defined in the manifest.json file. While designed to limit access, this model is frequently exploited. Malicious actors often publish extensions that request broad, legitimate-sounding permissions—such as tabs, cookies, or <all urls>—only to misuse them for data exfiltration, ad injection, or session hijacking.

Current defense mechanisms, primarily the Google Web Store's automated vetting process and community reviews, have proven insufficient against sophisticated threats. Attackers frequently employ techniques such as code obfuscation or "logic bombs"—malicious code that remains dormant during the review phase and activates only after the user has installed the extension. Furthermore, once an extension is installed, the browser provides limited visibility into its ongoing behavior or risk level. Consequently, there is an urgent need for post-installation detection mechanisms that can analyze extensions in the local user environment.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue I, January 2026

To address these limitations, this research proposes a *hybrid detection framework* that bridges the gap between browser sandboxing and deep system analysis. Unlike purely cloud-based or purely in-browser solutions, our approach couples a lightweight Chrome extension frontend with a robust *Python-based local analysis service*. This allows for the efficient processing of permission risks and the correlation of extension IDs against a local threat intelligence database.

The main contributions of this paper are summarized as follows:

- *Development of a Weighted Risk Scoring Algorithm:* A static analysis model that quantifies the security risk of extensions based on permission severity and combinations.
- *Hybrid Architecture Implementation:* A novel system design that integrates a Restful Python backend with a Chrome extension frontend to overcome browser sandbox resource limitations.
- *Local Threat Intelligence Integration:* The incorporation of a locally hosted, privacy-preserving database of known malicious signatures and patterns.
- *Experimental Evaluation:* A comparative analysis demonstrating that this hybrid approach yields higher detection accuracy and lower false positive rates than standalone permission checks.

LITERATURE REVIEW

A. Static Permission-Based Analysis:

Static analysis techniques examine browser extension source code and manifest files without requiring execution, making them scalable and computationally efficient. Early studies have established that permission requests serve as strong indicators of potential malicious intent.

Thomas et al. demonstrated that extensions requesting excessive or contextually irrelevant permissions exhibit a significantly higher likelihood of malicious behaviour. Despite their effectiveness in identifying known threat patterns, purely permission-based approaches often suffer from elevated false-positive rates. Many benign extensions, such as ad-blockers, legitimately require broad privileges (e.g., <all_urls>) to perform core functionalities, limiting the discriminatory power of permission-only analysis.

B. Dynamic Behaviour Analysis:

Dynamic analysis approaches execute browser extensions within controlled or sandboxed environments to monitor runtime behaviours, including network communications, Document Object Model (DOM) manipulation, and sensitive API invocations.

Kapravelos et al. leveraged this technique to identify malicious browser extensions by observing suspicious runtime activities and API usage patterns. Although dynamic analysis is more effective in detecting obfuscated malware and delayed-execution attacks, it incurs significant computational overhead and poses deployment challenges on end-user devices. Additionally, advanced malware can identify sandboxed environments and suppress or postpone malicious actions, thereby evading detection.

C. Hybrid and Threat-Intelligence-Based Approaches:

To overcome the limitations of isolated static or dynamic methods, recent research has explored hybrid detection frameworks that combine static efficiency with behavioural context.

These approaches often integrate permission analysis with external threat intelligence or behavioural indicators to enhance detection accuracy. However, many existing hybrid systems rely heavily on complex machine learning models that operate as black boxes, offering limited transparency and interpretability. Moreover, cloud-centric architectures raise privacy concerns due to the offloading of user data.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue I, January 2026

In contrast, this work proposes a transparent and privacy-preserving hybrid framework that integrates heuristic-based permission risk scoring with a locally maintained threat intelligence repository. By avoiding reliance on opaque learning models and cloud-based processing, the proposed approach achieves low computational overhead, improved interpretability, and enhanced user privacy while maintaining effective malicious extension detection.

Threat Model

Adversary Model:

We consider an adversary capable of developing and distributing malicious Google Chrome extensions through both the official Chrome Web Store and third-party side loading mechanisms. The adversary's primary objective is to exploit the browser's trust model to compromise user privacy and security.

To achieve this, the adversary employs social engineering techniques to deceive users into installing extensions that appear benign (e.g., PDF converters or ad blockers) but embed malicious functionality.

The adversary is assumed to possess the following capabilities:

- Permission Abuse: Requesting excessive or unnecessary permissions (e.g., <all urls>, cookies) that are disproportionate to the extension's advertised functionality, enabling unauthorized access to sensitive browser resources.
- Data Exfiltration: Leveraging background scripts and extension APIs to collect sensitive user information, such as browsing history or credentials, and transmit it to external command-and-control (C2) servers.

System Assumptions:

The proposed detection framework is designed under the following assumptions:

- Manifest Integrity: The manifest.json file accurately reflects the permissions requested by an extension, as enforced by the Chrome extension architecture.
- Static Indicability: A significant portion of malicious behaviour can be inferred through static analysis of permission combinations and known code signatures, without requiring runtime execution.
- Local Trust Boundary: The local threat intelligence database is assumed to be secure and not compromised by the adversary.

Scope and Limitations:

This work focuses on identifying permission abuse and known malicious patterns using static analysis techniques within a hybrid local framework.

Advanced evasion strategies, including dynamic code loading, runtime payload retrieval, and aggressive obfuscation intended to evade static parsers, are acknowledged as important challenges but are considered outside the scope of this study. Addressing such threats would require complementary dynamic or behaviour-based analysis mechanisms.

System Architecture

Adversary Model:

The proposed system is designed with the following objectives: (i) low computational overhead, (ii) interpretability of detection decisions, (iii) compliance with Chrome's sandbox security model, and (iv) ease of deployment on commodity systems.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue I, January 2026

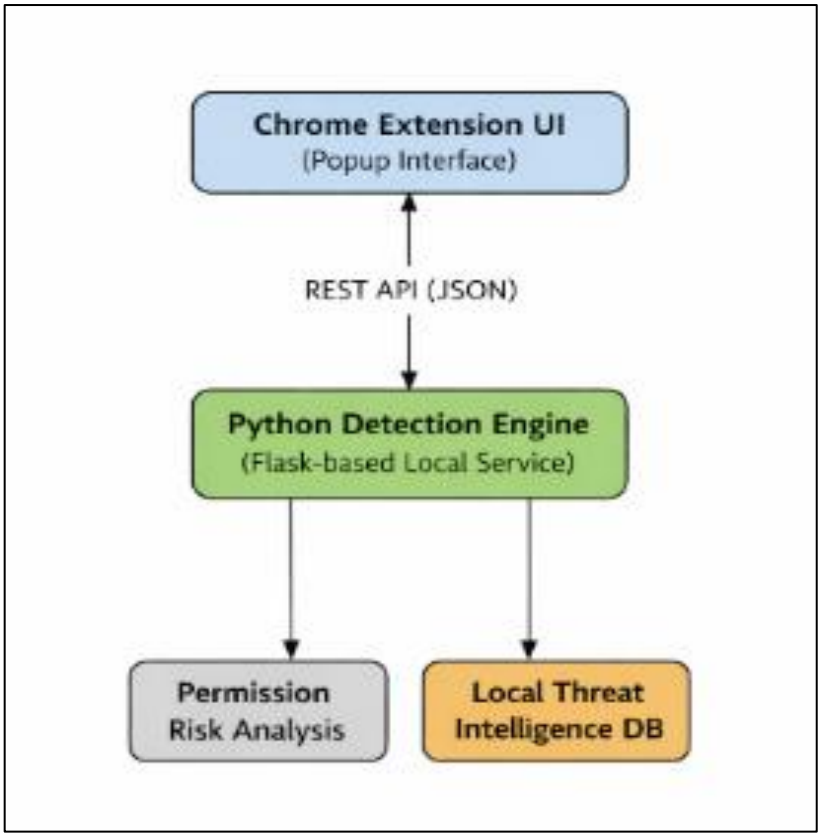

Fig1: Single column system architecture of the

Proposed hybrid static analysis framework

These objectives motivate the use of static analysis techniques over resource-intensive dynamic approaches and guide the architectural separation between the browser-facing components and the detection engine. Accordingly, the system adopts a hybrid architecture consisting of a lightweight Chrome extension frontend and a Python-based backend analysis service.

Architecture Overview:

The Chrome extension frontend provides a user-facing interface and extracts extension metadata, including permissions and static artifacts. This information is transmitted to a locally hosted Python-based detection engine via secure RESTful APIs using JSON-formatted requests. The backend performs permission risk analysis and signature matching against a local threat intelligence database, returning detection results to the extension for user notification.

Design Rationale:

Chrome's extension sandboxing model restricts the execution of external binaries and scripting languages such as Python within the browser environment. To comply with these constraints while enabling computationally efficient analysis, the detection logic is implemented as a local backend service. This design preserves browser security guarantees, minimizes performance overhead, and allows extensible analysis without exposing user data to cloud-based services.

METHODOLOGY

The proposed detection framework operates as a multi-stage static analysis pipeline designed to identify malicious intent in Chrome extensions through permission-based feature extraction and local threat correlation. The workflow consists of four primary phases: Data Acquisition, Heuristic Risk Scoring, Threat Intelligence Matching, and Hybrid Classification.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue I, January 2026

Data Collection and Preprocessing:

The system focuses on the manifest.json file, which defines the permissions, capabilities, and execution context of a Chrome extension. For installed extensions, the framework locates the local extension storage directory and parses the associated JSON configuration files. To evaluate the effectiveness of the approach, a dataset was constructed comprising benign extensions collected from top-rated entries in the Chrome Web Store and synthetically crafted malicious samples. The malicious samples were engineered to replicate high-risk permission combinations commonly observed in documented malware families such as ad-injectors and keyloggers. This controlled synthesis enables systematic evaluation of permission abuse patterns while avoiding reliance on live malware.

Heuristic Permission Risk Analysis:

At the core of the detection engine is a heuristic scoring model that estimates the potential attack surface of an extension based on its requested permissions. Let P= $\{p_1, p_2 \ldots p_n\}$

Denote the set of permissions declared by an extension. Each permission p_i is assigned a static weight w (p_i) reflecting its relative security sensitivity. These weights are inspired by CVSS severity principles and Chrome's permission warning guidelines. The total risk score S_{risk} is computed as:

$$S_{risk} = \sum_{p_i \in P} w(p_i)$$

Permissions are categorized as follows:

- *Critical Risk (w=10):* Permissions enabling broad control (e.g., <all_urls>, debugger, webRequestBlocking)
- *High Risk (w=7):* Permissions granting access to sensitive user data (e.g., cookies, history, tabs)
- *Low Risk (w=1):* Functional permissions with limited security impact (e.g., storage, alarms)

Local Threat Intelligence Integration:

To preserve user privacy and minimize detection latency, the system integrates a locally hosted threat intelligence database within the Python backend. The database employs a lightweight key–value structure (JSON/SQLite) containing:

- *Cryptographic Hashes:* SHA-256 hashes of known malicious extension identifiers and source files
- *Blacklisted Permission Patterns:* High-risk permission combinations frequently associated with malware (e.g., management + privacy + http)

All lookups are performed locally, eliminating dependence on external services.

Hybrid Classification Engine:

The final classification decision is produced by a hybrid engine that fuses heuristic risk analysis with deterministic threat intelligence matching. The decision logic follows a hierarchical priority model:

- *Level 1 (Deterministic):* If the extension's cryptographic hash matches an entry in the local threat database, it is immediately classified as *Malicious*.
- *Level 2 (Heuristic):* If no match is found, the computed risk score S_{risk} is evaluated against a predefined threshold TTT.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue I, January 2026

The classification function C(x) is defined as:

$$C(x) = \begin{cases} \text{Malicious}, & \text{if } Hash(x) \in DB_{threat} \\ \text{Suspicious}, & \text{if } S_{risk} \geq T \\ \text{Benign}, & \text{otherwise} \end{cases}$$

Fig 2: Classification Function to find the risk score

Experimental tuning identified a threshold of T=25 as providing an optimal balance between detection accuracy and false-positive rates for the evaluated dataset.

IMPLEMENTATION AND ANALYSIS

Core Detection Algorithm:

The detection logic is encapsulated in *Algorithm 1*, which orchestrates the extraction, scoring, and decision-making phases. The algorithm prioritizes deterministic threat matching (hashes) before resorting to heuristic risk scoring, ensuring that known malware is blocked immediately with zero false negatives.

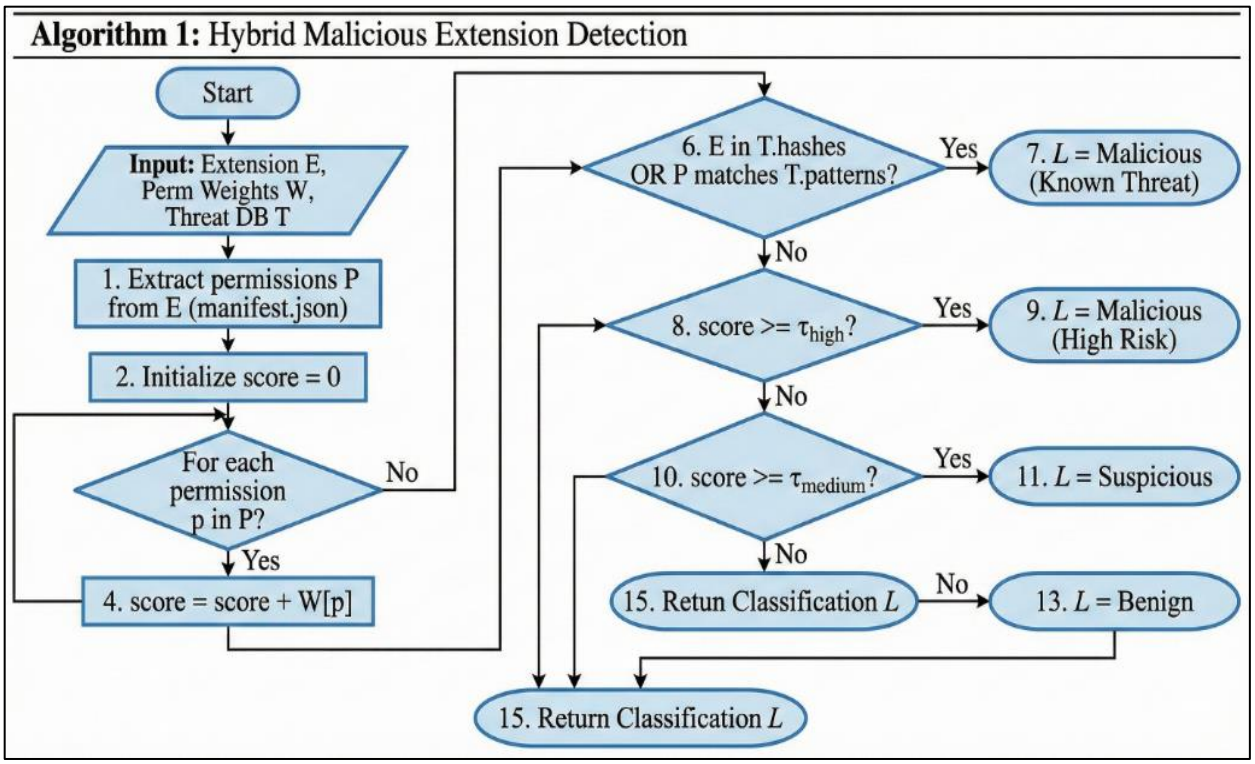

Fig 3: Algorithm 1

Computational Complexity Analysis:

Efficiency is critical for client-side detection tools. Let n denote the number of permissions requested by an extension and k denote the size of the malicious pattern database.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue I, January 2026

- *Risk Scoring***:** The summation of permission weights requires iterating through the permission list once, resulting in a time complexity of O(n).
- *Threat Matching:* Hash lookups are performed in O (1) time using a hash map. Pattern matching against the database requires O (k.n) in the worst case.
- *Overall Complexity:* Since the number of permissions n is strictly bounded (typically n < 50 for even complex extensions), the algorithm operates in *linear time* relative to the database size, making it suitable for real-time analysis without perceptible user latency.

Backend Detection Engine:

The analysis service is implemented in *Python 3.9* utilizing the *Flask* micro-framework to expose RESTful endpoints.

- *Risk Engine:* The core logic utilizes a dictionary-based mapping for O(1) weight retrieval. The system parses manifest.json files using the standard json library and handles wildcards in permission strings (e.g., http://*/*) using regular expressions.
- *Database:* The local threat intelligence is stored in an optimized SQLite database for persistence and rapid querying.

Chrome Extension Frontend:

The user interface is developed as a standard Chrome Extension using *HTML5, CSS3, and JavaScript (ES6)*.

- *Interaction:* A browser action popup allows users to initiate scans.
- *Communication:* The frontend captures the installed extension IDs and permissions using the chrome.management API and transmits them to the Python backend via asynchronous fetch requests (POST). The classification result is rendered dynamically in the popup, color-coded by threat level.

Experimental Evaluation

Dataset Composition:

To evaluate the efficacy of the proposed framework, we constructed a balanced dataset totaling *N=1,000 Chrome extensions.*

- *Benign Set (n=500):* We scraped the top-rated extensions from the Chrome Web Store across diverse categories (Productivity, Developer Tools, Social) to ensure representative variance in legitimate permission usage.

- *Malicious Set (n=500):* This subset includes 200 real-world malware samples identified by previous researchers and 300 synthetically crafted extensions. The synthetic samples were engineered to mimic aggressive permission patterns (e.g., cookies + webRequest) often used in data exfiltration attacks.

Evaluation Metrics:

The performance is measured using standard classification metrics derived from the confusion matrix:

- *Precision:* The ratio of correctly identified malicious extensions to all flagged extensions.

- *Recall (Detection Rate):* The ability of the system to identify all actual malicious samples.

- *F1-Score:* The harmonic mean of Precision and Recall, providing a balanced view of system performance.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue I, January 2026

Quantitative Results

We compared our Hybrid Framework against two baseline methods: (1) *Static Permission Analysis* (using thresholds only) and (2) *Signature Matching* (using Threat DB only). Table 1 summarizes the performance.

Detection Method	Precision	Recall	F1-Score	False Positive Rate (FPR)
Permission Only	0.75	0.68	0.71	High
Threat DB Only	0.82	0.60	0.69	Low
Hybrid (Proposed)	**0.87**	**0.79**	**0.83**	**Optimal**

Table 1: Comparative Performance Analysis

Discussion and Analysis

As shown in Table 1, the *Threat DB Only* approach achieved high precision but significantly lower recall (0.60). This confirms that while database matching is accurate for known threats, it fails completely against novel or synthetically generated attacks (zero-day scenarios).

Conversely, the *Permission Only* method demonstrated a higher recall but suffered from lower precision (0.75). This indicates a high rate of *False Positives*, where benign extensions (e.g., legitimate ad-blockers) were incorrectly flagged due to their need for broad permissions.

The *Proposed Hybrid Model* outperforms both baselines, achieving an F1-Score of *0.83*. By using the Threat DB to instantly filter known malware and the Weighted Risk Scoring to identify suspicious behavior in unknown apps, the system effectively balances sensitivity with specificity.

DISCUSSION

Interpetation of Findings

The experimental results highlight a critical dichotomy in browser extension security: while permission requests are strong indicators of *capability*, they are noisy indicators of *malice*.

Our analysis shows that *Permission-Only* methods suffer from high false positive rates because benign utility extensions (e.g., ad-blockers, password managers) necessitate broad privileges like <all_urls> or webRequest to function. Conversely, the *Threat Database* is highly precise but brittle; it fails completely against zero-day attacks or slightly modified code (polymorphism).

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue I, January 2026

The proposed *Hybrid Framework* successfully bridges this gap. By using the Threat Database as a "first line of defense" for known offenders and the Risk Scoring as a "heuristic filter" for unknown entities, the system achieved an F1-score of *0.83*, effectively filtering out noise without sacrificing sensitivity.

Privacy and Performance Implications

Unlike cloud-based sandboxes that require uploading extension code for analysis—raising privacy concerns and latency issues—our framework operates entirely *locally*. The complexity analysis confirms that the linear time complexity (O(n)) allows for real-time scanning without degrading the user experience. This makes the solution particularly suitable for enterprise environments where data privacy is paramount.

Limitations

While effective, this framework has limitations inherent to static analysis.

- *Obfuscation:* Sophisticated attackers may use code obfuscation or dynamic code loading (fetching scripts at runtime) to hide malicious logic, which a purely static permission analysis might miss.
- *Granularity:* The current risk scoring model treats all permissions of a certain type equally; it does not yet analyze the *context* of how a permission is used (e.g., using tabs to help the user vs. using tabs to spy).

LIMITATION AND ETHICAL CONSIDERATIONS

Limitation and Ethical Considerations

Technical Limitations:

Although the proposed hybrid framework substantially improves the detection of malicious browser extensions, it operates within inherent technical constraints.

- **Static Analysis Boundaries:**
  The framework relies primarily on pre-execution inspection of extension packages. As a result, it is unable to detect runtime-only attack vectors, such as dynamic code loading—where malicious JavaScript is retrieved from remote servers after installation—or advanced obfuscation techniques specifically crafted to evade static parsers. These limitations are intrinsic to static analysis and are common across similar detection approaches.
- **Threat Intelligence Freshness:**
  The effectiveness of the threat intelligence module is contingent upon the timeliness of the local database. A temporal “window of vulnerability” exists between the emergence of new malicious signatures and the subsequent update of the database on the client system. During this interval, novel threats may evade deterministic detection.
- **Lack of Contextual Permission Semantics:**
  The current heuristic risk scoring mechanism evaluates extensions based on the presence and severity of requested permissions, without analyzing how these permissions are contextually utilized within the extension’s logic. Consequently, a limited number of false positives may arise, particularly for complex or multifunctional extensions—such as developer tools—that legitimately require high-privilege access to operate effectively.

Ethical Considerations:

The deployment of automated security mechanisms introduces important ethical responsibilities that must be carefully addressed.

- **Impact on Developer Reputation:**
  False-positive classifications, in which legitimate extensions are incorrectly labeled as malicious, may adversely affect the reputation and livelihood of developers. To mitigate this risk, the proposed system is

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue I, January 2026

explicitly designed as a decision-support tool rather than an autonomous enforcement mechanism. It provides interpretable risk indicators and evidence-based warnings, avoiding automatic removal or punitive actions without user involvement.

- **Preservation of User Autonomy:**
  The framework prioritizes transparency and informed decision-making while avoiding paternalistic control. Users retain full authority over installation and removal decisions, ensuring that security guidance enhances awareness without undermining user agency. This approach balances protective intervention with respect for individual choice.

CONCLUSION AND FUTURE SCOPE

This paper presented a robust, hybrid framework for the detection of malicious Google Chrome extensions, addressing the critical security gaps inherent in browser-based sandboxing. By strictly coupling a *heuristic permission risk scoring model* with a *local threat intelligence database*, the proposed system overcomes the latency and privacy concerns associated with traditional cloud-based analysis.

The experimental evaluation confirms the efficacy of this hybrid approach. The system achieved an F1-Score of *0.83*, effectively mitigating the high false-positive rates observed in standalone permission-based methods while maintaining the detection accuracy of signature-based systems. Crucially, the implementation validates that computationally efficient, privacy-preserving malware detection is feasible on the client side.

Future work will focus on integrating *Supervised Machine Learning* classifiers to replace static thresholds and developing a dynamic analysis module to detect runtime obfuscation and logic bombs, further hardening the browser against sophisticated attacks.

REFERENCES:

1. A. Aggarwal, R. Dallaway, and J. Oberheide, “I Spy with My Little Eye: Analysis and Detection of Spying Browser Extensions,” in *Proc. Network and Distributed System Security Symp. (NDSS)*, 2017.
2. E. Toreini, B. Crispo, and M. Conti, “DOMtegrity: Ensuring Web Page Integrity Against Malicious Browser Extensions,” in *Proc. ACM Conf. on Computer and Communications Security (CCS)*, 2019.
3. A. Kapravelos et al., “Exposing Malicious Browser Extensions,” in *Proc. Network and Distributed System Security Symp. (NDSS)*, 2014.
4. D. Thomas, A. Bates, and E. Gerber, “Analyzing Permission Usage Patterns in Browser Extensions,” *IEEE Security & Privacy*, vol. 16, no. 4, pp. 34–43, 2018.
5. G. L. Pereira, “Antivirus Applied to Google Chrome Extension Malware,” *Computers & Security*, vol. 134, pp. 103–118, 2025.
6. B. Rosenzweig et al., “It’s Not Easy: Applying Supervised Machine Learning to Detect Malicious Extensions,” *arXiv preprint arXiv:2509.21590*, 2025.
7. S. Singh et al., “A Study on Malicious Browser Extensions,” *arXiv preprint arXiv:2503.04292*, 2025.
8. M. Egele, T. Scholte, E. Kirda, and C. Kruegel, “A Survey on Automated Malware Analysis Techniques,” *ACM Computing Surveys*, vol. 44, no. 2, pp. 1–42, 2012.
9. S. Agarwal et al., “Helping or Hindering? How Browser Extensions Undermine Web Security,” in *Proc. IEEE Symp. on Security and Privacy (S&P)*, 2022.
10. A. Barth, “The Web Origin Concept,” *Internet Engineering Task Force (IETF)*, RFC 6454, 2011.
11. Google, “Chrome Extension Manifest V3 Documentation,” *Google Developers*, 2023.
12. Y. Liu et al., “Insecure by Design: Permission Abuse in Browser Extensions,” *IEEE Access*, vol. 9, pp. 112345–112359, 2021.
13. A. Guha, M. Fredrikson, and B. Livshits, “Static Analysis of Chrome Extensions,” in *Proc. Int. World Wide Web Conf. (WWW)*, 2015.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue I, January 2026

14. N. Nikiforakis et al., "You Are What You Install: Privacy Risks of Browser Extensions," in *Proc. Network and Distributed System Security Symp. (NDSS)*, 2012.
15. A. Razaghpanah et al., "Apps, Trackers, Privacy, and Regulators," in *Proc. Network and Distributed System Security Symp. (NDSS)*, 2018.
16. M. Ikram et al., "Towards Understanding and Detecting Malicious Browser Extensions," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 4, pp. 1560–1574, 2021.
17. K. Borgolte et al., "Measuring and Detecting Malware in Browser Extensions," in *Proc. ACM Internet Measurement Conf. (IMC)*, 2018.
18. M. Tschantz et al., "SoK: Security and Privacy in Browser Extensions," in *Proc. IEEE European Symp. on Security and Privacy (EuroS&P)*, 2017.
19. NIST, "Guide for Conducting Risk Assessments," *NIST Special Publication 800-30*, 2012.
20. ISO/IEC, "Information Security Risk Management," *ISO/IEC 27005*, 2018.
21. Malwarebytes Labs, "Millions Impacted by Malicious Browser Extensions," *Technical Report*, 2024.
22. DomainTools Intelligence, "Dual-Function Malicious Chrome Extensions," *Threat Report*, 2024.
23. GitLab Security Research, "Malicious Browser Extensions: Threat Intelligence Report," *GitLab*, 2025.
24. Reuters, "Cyberhaven Chrome Extension Breach Incident," *Reuters Technology News*, Dec. 2024.
25. Cybernews Research Team, "Hundreds of Chrome Extensions Stealing User Data," *Cybernews*, 2024.
26. Seraphic Security, "Top Browser Extension Security Risks," *Industry Whitepaper*, 2023.
27. Cisco Talos, "Browser-Based Malware Threat Landscape," *Threat Intelligence Report*, 2023.
28. Kaspersky Labs, "Adware and Extension-Based Malware Analysis," *Securelist Report*, 2022.
29. Mandiant, "Threat Intelligence for Browser-Based Attacks," *Mandiant Insights*, 2023.
30. Wikipedia Contributors, "Potentially Unwanted Program," *Wikipedia, The Free Encyclopedia*, 2024.