COMBAT: Continuous Monitoring of Browser Extensions against Post-Installation Threats

Nisha Rathi[1], Ananya Sharma[2], Mahika Dadheech[3], Raj Padiyar[4*], Shambhavi Pandit[5], Namit Rathi[6]

[1,2,3,4,5]Acropolis Institute of Technology and Research, Indore (M. P.) India
[6]PES University, Bengaluru, Karnataka, India
nisharathi@acropolis.in, rajpadiyar240805@acropolis.in

Abstract. Browser extensions are small add-ons for the browsers that make everyday browsing faster and easier. System permissions make these extensions powerful and easy target for attackers. Many extensions seem perfectly safe when first installed but quietly turn malicious through updates down the line. Most defenses catch threats before installation of these extensions but fail to detect threats once an extension is already running on user's browser. Extensions can dodge security checks by sneaking in harmful updates, quietly grabbing extra permissions, or waiting to activate their harmful behavior until later. COMBAT is a continuous monitoring system that sits inside the browser and keeps watching extensions even after their installation. It keeps an eye on updates, monitors permission shifts, and watches how extensions use the network and browser APIs to identify any suspicious activity. Unlike existing tools that only check extensions upfront, COMBAT provides ongoing protection in the background. The proposed monitoring system works alongside existing security tools.

Keywords: Browser Security, Extension Security, Malware Detection, Continuous Monitoring, Update Analysis.

1 Introduction

Browser extensions are common part of web browsing that helps to do things block ads and manage passwords. Extensions need exclusive system permissions that allow them to access sensitive data like cookies, keystrokes, page content and browsing history [1]. Malicious extensions can secretly collect user's data, show unwanted ads, or send information to outside servers without user's knowledge. Many extensions start out harmless and build a loyal user base, but turn malicious once they have built enough trust and users [3, 5].

This behavioral shift helps attackers bypass one-time security checks. Extensions can sneak in malicious code through updates, ask for unnecessary permissions, or wait before showing harmful behavior. By the time anything is detected, millions of users are already affected. This clearly shows a major gap in how browser extension securi-

© The Author(s) 2026
S. Bhalerao et al. (eds.), *Proceedings of the 2nd International Conference on Recent Advancement and Modernization in Sustainable Intelligent Technologies & Applications (RAMSITA-2026)*, Advances in Intelligent Systems Research 207,
https://doi.org/10.2991/978-94-6239-678-4_24

ty works today. Figure 1 illustrates the typical attack timeline of a malicious browser extension from distribution to exploitation.

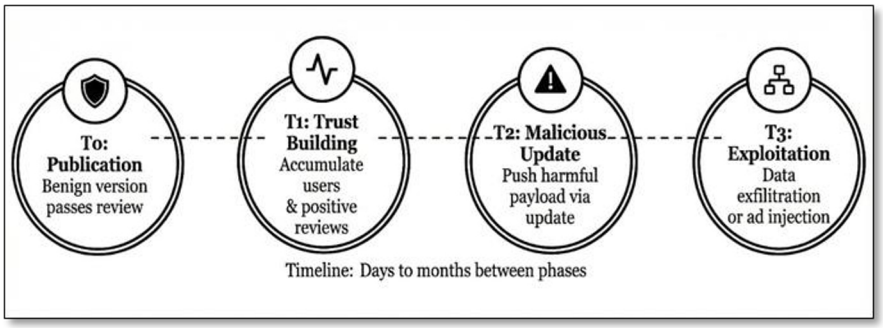

Fig.1. Typical attack timeline for malicious browser extensions

Security of extensions is assessed by static code analysis, permission verification and automated or manual security reviews. Several ML approaches have been developed to classify extensions [2, 4] but they show limited effectiveness against real-world attacker strategies. Strategies like code obfuscation, delayed activation patterns, and behavioral change help in bypassing security assessments [4]. The challenge of detecting malicious updates has been done through update delta analysis. Pantelaios et al. [3] revealed that comparing permission sets and code changes between versions can tell where extensions turn harmful over time. Existing research highlights three fundamental limitations in the existing system. First, continuous monitoring of extensions does not occur after installation. Second, malicious updates and permission expansion are difficult to detect as they occur in real-time. Third, ML-based detection systems show little to no efficiency when attackers change style of attacks with time due to concept drift [2, 4].

Through COMBAT, we propose a continuous monitoring system that runs directly inside the browser and keeps a close eye on extensions after they are installed. Our system introduces four key contributions. Continuous Post-Installation Monitoring Architecture keeps checking extensions regularly, even after updates. Update Delta Analysis records and compares what changes between versions to catch anything suspicious. Behavior Analysis Module watches network activity and sensitive API usage to spot potential risks. Hybrid Risk Assessment Strategy brings all these signals together along with past behavior to determine how dangerous an extension really is.

By bringing all these parameters together, COMBAT protects users against extensions that turn malicious after an update. It works fully within modern browser security rules and keeps user privacy intact. All analysis happens locally on the user's device and it never monitors what users do on web pages. It looks at metadata, code structure, permissions, and browser signals to identify risky behavior. This makes it compatible with existing browser security systems while keeping user data private.

2 Background and Related Work

2.1 Browser Extension Security Model

Based on permission model the browser extensions request browser features and user data permissions. It use browser APIs to do things like modify web pages, monitor network activity, store data, and manage tabs. This level of access is necessary but can be dangerous if misused. Table 1 shows common extension permissions and the security risks that come with them.

Table 1. General Browser Extension Permission and the Related Security Risks

Permission	Capability	Security Risk
Tabs	Read URLs, titles of open tabs	Browsing history tracking
WebRequest	Intercept & modify network traffic	Man-in-the-middle attacks
Cookies	Read/write browser cookies	Session hijacking
storage	Persistent local data storage	Data exfiltration staging
<all_urls>	Access content on all websites	Credential theft, ad injection
declarativeNetRequest	Modify requests declaratively	Search hijacking, redirects

The manifest file, background scripts and content scripts are the important components of extension architecture The manifest file defines the extension's structure and the permissions it needs. Background scripts run quietly in the background to handle events and keep the extension working. Content scripts run inside web pages and allow the extension to read and change page content. All these parts communicate with each other through message passing APIs to keep everything working together.

2.2 Threat Landscape and Malicious Behavior

Both technical weaknesses and user behavior play a role in the spread of malicious extensions. The Batten Cyber guide focuses on permission misuse and how attackers trick users into installing harmful extensions [1]. A data-driven study by Singh et al. [5] reveals the latest evade methods and types of malicious code found in recent years. Malicious extensions can secretly collect data, push unwanted ads, alter search results, steal passwords [1, 5]. These methods harm user security and privacy.

In gradual permission escalation, extension starts by asking for very little and slowly requests more permission through updates. Most users never notice these changes, giving the extension more and more access until it shifts from something useful to something harmful [3]. In delayed activation, the extension starts safely, passes store reviews, and builds a solid user base. Once it has enough trust and good ratings, the developer quietly pushes an update with malicious code that either activates right away or kicks in after some time [3,5]. Figure 2 shows how these malicious behaviors are distributed across browser extensions.

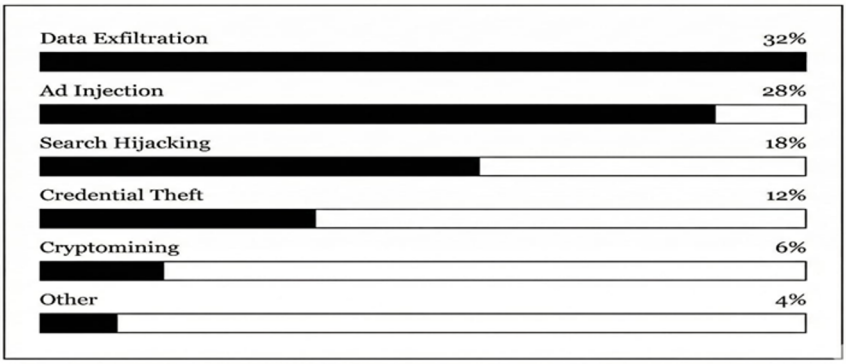

Fig.2. Browser extensions' malicious performance

2.3 Detection Approaches: Static Features and Machine Learning

One popular approach to detecting malicious extensions is to analyze extension code and use machine learning to classify safe extensions from malicious ones. It was founded by researchers [2] that studying static code features and appropriate indicators can catch many malicious extensions in controlled tests. On labeled data, Random Forest and XGBoost performed well when evaluated.

Table 2. Extension detection ML classifiers' Performance contrast on labeled data

Classifier	Lab Accuracy	Real-World Accuracy	F1-Score (Lab)
Random Forest	94.2%	67.3%	0.91
XGBoost	95.8%	71.2%	0.93
Neural Network	93.5%	64.8%	0.89
Logistic Regression	88.7%	69.5%	0.84
SVM	91.3%	66.1%	0.87

Most browser extension stores try to catch malicious extensions before they ever reach users. They scan the code for red flags like hidden scripts, suspicious behavior, or permission requests that seem too broad. Store operators also review extensions manually as an extra layer of protection. Table 2 breaks down how well different machine learning classifiers perform at spotting these threats using standard evaluation metrics.

Detection of malicious extension is harder than it seems. Researchers put machine learning detection to the test on a large Chrome Web Store dataset and ran into a real-world problem. Their models looked great in the lab but struggled when exposed to actual data. The issue was concept drift, where the nature of malicious extensions keeps evolving in ways the model was never trained to recognize [4].

2.4 Update Delta Analysis and Temporal Detection

Some extensions begin as perfectly harmless tools and turn dangerous only after an update. Researchers studied and tackled this problem by introducing update delta analysis, a method that compares one version of an extension to the next to spot suspicious changes over time [3]. A single version of an extension can look completely clean on its own, but tracking how it changes across updates can reveal patterns that hint at harmful intent. Their study confirmed this works, successfully identifying past cases where extensions turned malicious after installation.

This method has a notable limitation. Most systems relying on this approach only work with historical data. It means they analyze old records offline rather than monitoring extensions as updates happen. For everyday users, that leaves a real gap in live protection [3].

2.5 Methodological Lessons and Open Challenges

Feature selection plays a critical role in detection effectiveness. Static code and metadata features can help, but they have limits. Attackers often hide their code or wait before doing anything harmful. Looking at how extensions change over time works better. Update delta analysis i.e., analyzing changes over time, addresses this gap by catching behaviors that single-version analysis would miss. Real-world deployment conditions are equally important. Data changes over time and models trained on old data become less useful. This problem is called concept drift and it needs to be handled carefully.

There are still many challenges left to solve. Current detection systems find it hard to deal with attackers who hide code or delay harmful actions [2,3,4]. Some attackers even manipulate the training data itself. There is also a shortage of long term datasets. Researchers also lack standard ways to study concept drift properly. Combining update delta analysis, machine learning and human review into one smooth system is not easy. Keeping false alarms low makes it even harder. Most studies only focus on Chrome extensions. Other browser ecosystems have not received much attention yet and need more research going forward.

3 System Design

3.1 Architecture Overview

The proposed system works as a browser extension that watches over other installed extensions. Among the main parts that work together, the first part is the static analysis module. It checks for changes between extension updates to spot anything suspicious. The second part is the behavioral monitoring module. It keeps an eye on what extensions are doing while the browser is running. The third part is the risk assessment module. It collects signals from the other two parts and uses them to detect potentially harmful behavior.

The system is designed to be light and fast. It only focuses on important security events instead of tracking every small activity. This keeps the browser running smoothly without slowing it down. Everything stays private and local at all times. All the monitoring happens inside the user's own browser. No data is sent outside or shared with anyone.

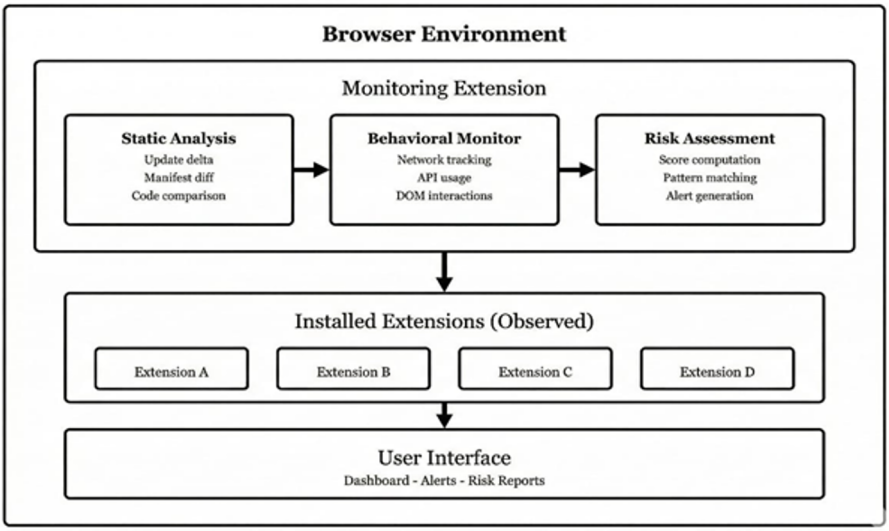

Fig.3. Uninterrupted extension observation architecture

3.2 Update Delta Analysis

When an extension updates, the proposed system saves both the old and new versions. This way it can clearly see what has changed between the two. The system first looks closely at the manifest file. It checks what permissions the extension is asking for. It also looks at which scripts are running and what web resources the extension can access.

If the update asks for new permissions or tries to reach more websites, the system takes note. If any security rules have changed, the system flags it right away. The user is then informed that something important has changed. The system also compares the code inside background and content scripts. It looks for any meaningful differences between the old and new versions.

Some changes are treated as risky signals like new network requests and changes in how the extension interacts with web page. Differences in how scripts communicate with each other are also watched closely. All of this process is described step by step in Algorithm 1.

Algorithm 1: Update Delta Analysis

Input: Extension ID, Old Version, New Version
Output: Risk Score, Alert List

```
deltaManifest ← CompareManifests(oldVersion, newVersion)
newPerms ← deltaManifest.addedPermissions
riskScore ← 0
alerts ← []
for each perm in newPerms do
if perm.isHighRisk() then
riskScore ← riskScore + perm.riskWeight
alerts.append(CreateAlert(perm))
end if
end for
deltaCode ← CompareScripts(oldVersion, newVersion)
if deltaCode.hasNetworkCalls() then
riskScore ← riskScore + NETWORK_RISK_WEIGHT
alerts.append(CreateNetworkAlert(deltaCode))
end if
if deltaCode.hasObfuscation() then
riskScore ← riskScore + OBFUSCATION_RISK_WEIGHT
end if
return riskScore, alerts
```

3.3 Behavioral Monitoring

The behavioral monitoring watches extension runtime activities so that it could catch any risky patterns. Network Monitoring keeps track of all internet connections made by extensions. It notices when an extension tries to connect to a domain it has never visited before. It also watches for data being sent to suspicious IP addresses. Any unusual pattern of data being quietly sent out is flagged right away. API usage monitoring keeps a close watch on how extensions use browser APIs. Some extensions make unusual or risky calls to APIs that are considered important and sensitive. This monitoring includes access storage APIs indicating data collection and the frequent use of web Request APIs indicating traffic interception. It also includes checking unusual patterns in opening and closing of tabs indicating click fraud and unexpected access to cookies or login APIs. The system also tracks how extensions interact with web pages and how it finds any suspicious changes to the DOM. System searches for behavior like inserting ads, watching form fields that may take passwords or changing page scripts which might inject harmful code into trusted websites. All the API groups that are checked along with the thresholds used to identify suspicious behavior are listed in Table 3.

Table 3. API categories, suspicious performance thresholds and risks

API Category	Monitored Operations	Threshold	Risk Level
Network	fetch(),XMLHttpRequest	>50 requests/min	High
Storage	chrome.storage.set()	>100 KB/session	Medium
Cookies	chrome.cookies.get()	>20 cookies/min	High
Tabs	chrome.tabs.create()	>10 tabs/min	Medium
WebRequest	onBeforeRequest	>100 intercepts/min	High

3.4 Risk Assessment Module and User Notification

This module calculates a risk score for extensions by using data from static analysis and behavior monitoring. It checks the extension's behavior over time including delayed actions and past activity. This helps in understanding whether the behavior is normal or potentially harmful. By putting all this information together, it generates an overall score that reflects the level of risk. It calculates the score using the following summation method:

$$RiskScore = \alpha \cdot \Delta P + \beta \cdot \Delta C + \gamma \cdot N + \delta \cdot A \quad (1)$$

In this formula, permission alteration is indicated by ΔP , code changes by ΔC f, unusual network behavior by N, and A is used for irregular API usage. The symbols α, β, γ, and δ are weight factors. Figure 4 shows how the risk scores are distributed and the thresholds used to classify them.

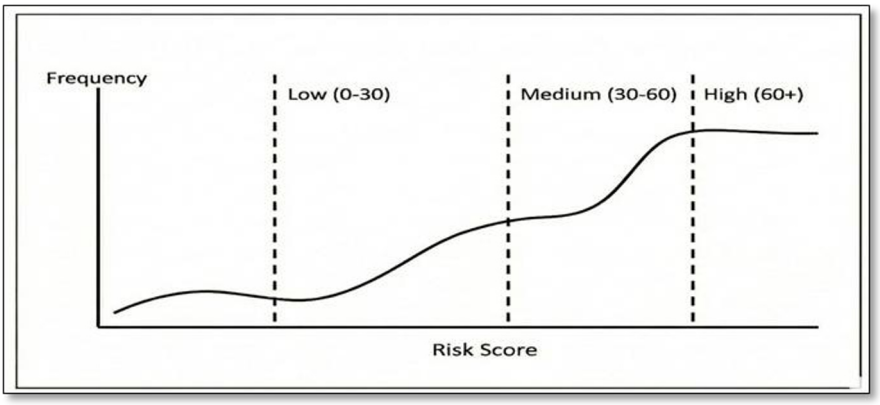

Fig.4. Distribution of Risk score and thresholds

The system keeps an eye on all extensions and gives each one a risk score. When the score gets too high it means the extension might be doing something harmful. The user then sees a simple alert on their screen. The alert is written in plain language that anyone can understand. It tells the user exactly what the extension did wrong and why it could be a problem for their privacy or security. The user can then choose to turn

off the extension or delete it completely. No technical knowledge is needed and the user stays in full control at all times.

4 Discussion

4.1 Advantages and Limitations

COMBAT keeps monitoring extensions even after they are installed so no threat goes unnoticed. It also catches weak spots where an extension starts behaving badly after an update. The proposed system is lightweight which means it runs smoothly in the background without slowing down the browser. All the analysis happens directly on the user's device so no personal data is ever sent outside. This keeps user privacy fully protected.

Malicious behavior of extensions is often hidden by using tricks like permission escalation and delayed activation to make harmful extensions look safe. Our system is designed to handle these tricks effectively. It works best when it is kept updated with the latest threat patterns. It can catch real threats without raising too many false alarms.

4.2 COMBAT's Integration with Open Defenses

COMBAT does not replace existing security tools but works alongside them to make protection stronger. Store level vetting does a good job of stopping many harmful extensions before they even reach the user. Machine learning helps by spotting suspicious patterns across a large number of extensions.

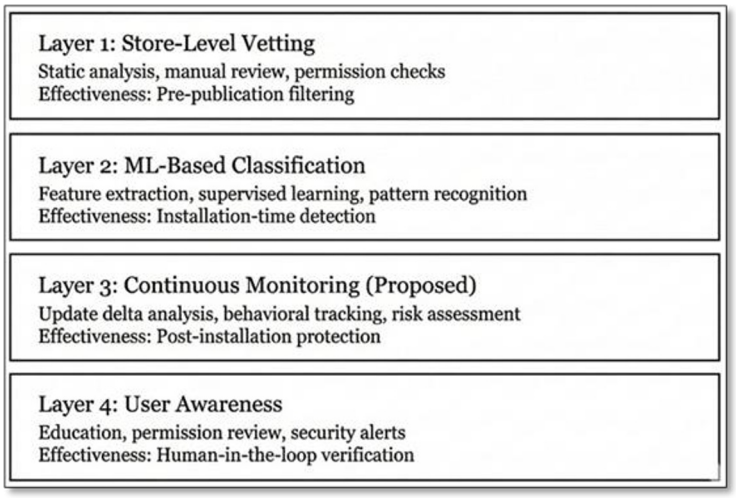

Fig.5. Browser Extension's 4-Layers Defense Architecture

The proposed system keeps protecting the user even after the extension is installed. This fills the gap that other tools often miss. Together these layers create a strong defense against advanced attacks that manage to get past the initial screening. Figure 5 shows how this layered defense approach works to keep browser extensions secure.

COMBAT effectiveness will be increased by integration with existing browser security features. There will be faster removal of confirmed malicious extensions through sharing threat data and analysis with extension stores. A strong layer of protection against attacks will be added through the integration with browser-level security protocols like site isolation and content security policies

4.3 Future Research Directions

Future research should focus on combining temporal, static and dynamic features in a single system as it could lead to more accurate detection than current methods. Building models that keep learning over time and can handle concept drift is also important [4]. To test these systems properly better and well organized labeled data is needed. Research on how humans and machines can work together would also help in making these systems more practical in real life.

5 Conclusion

Malicious browser extensions are a continuous threat. Attackers use the advantage that extensions are checked for safety only once that is when they are installed. They make the extensions behave safe at first and then turn it malicious later. It increases the chances of compromise in sensitive user data. The significant observation from past studies includes strong lab performance by static ML classifiers, detection of indicators of malicious intent through update-delta analysis and inaccuracy introduced by concept drift in real deployments [2,3,4].

COMBAT provides continuous security to extension. It combines behavioral monitoring with update delta analysis which keeps a check on the extension at regular intervals. Our approach tackles a real threat in browser extension security without slowing down the browser or compromising user privacy. COMBAT uses a layered architecture that brings together risk evaluation, behavioral observation and static investigation to deliver complete protection after an extension is installed. The proposed algorithm demonstrates how update delta analysis works within this process. The monitoring framework keeps track of network movements and API handling and how code talks and controls the web page content to spot any suspicious behavioral patterns. All these indications are then combined with suitable weights in the risk assessment model to generate clear and actionable alerts for the user.

The major focus should be on testing the system against real world malicious extensions and improving the risk scoring models. Since the extensions for browsers grow more complex, uninterrupted monitoring are becoming more necessary to protect users from post-installation threats.

References

1. Batten Cyber: Browser Extension Security: Avoid Malicious Add-ons. https://battencyber.com/prevention-guides/browser-extension-security/ (Accessed: December 2025)
2. Rydecki, J., Tong, J., Zheng, J.: Detecting Malicious Browser Extensions by Combining Machine Learning and Feature Engineering. In: Latifi, S. (ed.) Advances in Intelligent Systems and Computing, vol. 1445, pp. 105-113. Springer (2023). https://doi.org/10.1007/978-3-031-28332-1_13
3. Pantelaios, N., Nikiforakis, N., Kapravelos, A.: You've Changed: Detecting Malicious Browser Extensions through Their Update Deltas. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS 2020), pp. 477-491 (2020). https://doi.org/10.1145/3372297.3423343
4. Rosenzweig, B., Dalla Valle, V., Apruzzese, G., Fass, A.: It's Not Easy: Applying Supervised Machine Learning to Detect Malicious Extensions in the Chrome Web Store. ACM Transactions on the Web (2025). https://doi.org/10.1145/3770852
5. Singh, S., Varshney, G., Singh, T.K., Mishra, V., Verma, K.: A Study on Malicious Browser Extensions in 2025. arXiv preprint (2025). https://arxiv.org/abs/2503.04292

Open Access This chapter is licensed under the terms of the Creative Commons Attribution-NonCommercial 4.0 International License (http://creativecommons.org/licenses/by-nc/4.0/), which permits any noncommercial use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.