A Reproducible Evaluation of Rule-Based Browser Behaviour Monitoring: Distinguishing Observation, Alerting, and Execution Prevention

Ezany M. F.[1],* and Brian Joe[1]

[1] Faculty of Artificial Intelligence and Cyber Security, Universiti Teknikal Malaysia Melaka, Melaka, Malaysia

***Corresponding Author**

DOI: https://doi.org/10.47772/IJRISS.2026.101100257

Received: 24 October 2026; Accepted: 29 October 2026; Published: 26 November 2026

ABSTRACT

Rule-based monitoring can be used to monitor JavaScript and page activity. However, an alert alone is insufficient to establish malicious intent or show that the underlying action was prevented. In this research, we evaluate a browser plugin assigning score values to each of six operational categories: dynamic evaluation, cookie read/write operations, mutation of child lists, rapid clicks, external referrer traffic, and dynamic script insertion. The main experiment contained 20 scenarios with five configurations, new browser profiles, independent task-completion oracles, and three stability runs for each scenario-configuration combination. All 231 planned experiments were performed, and all integrity checks were successful. Both detector variants raised alerts in 9 of 27 runs across nine researcher-defined benign primary scenarios, while a controlled unauthorized action sequence completed in every detector run without an alert. For the same task being permitted and prohibited, the detector information was the same for these tasks if their browser operations were the same, which means that the authorization information was not included in the detector input. The visual overlays did not stop the tested scheduled JavaScript, whereas NoScript deny mode prevented both permitted and prohibited tasks. After peer review, a second frozen experiment was run on 18 sessions on a React single-page application and cross-origin analytics workload. All tasks were completed; the analytics workload resulted in warnings in all detector runs, and neither workload generated enforcement overlay. The SPA was always completed; there were no Original warnings and only one Corrected-v1 warning, which demonstrates how repeated DOM and script manipulation could interact with timing-sensitive rules. The results confirm evaluating observation, risk classification, visual response, task completion, and functional cost as separate outcomes. These findings characterize the tested mechanisms and workloads rather than population-level malware-detection accuracy.

Keywords: browser security, JavaScript monitoring, browser extension, runtime detection, controlled evaluation

INTRODUCTION

JavaScript security analysis can consider either structure of code, behavior during runtime, or both. IceShield was based on browser instrumentation in combination with heuristics, whereas ZOZZLE and JStap used structure to identify malicious JavaScript [3], [4], [6]. Engine-level systems like VisibleV8 and FV8 consider execution monitoring and coverage [7], [8]. In view of that, browser instrumentation and weighting approach are well-known technologies; therefore, the novelty of the current research consists of evaluation of the approach rather than introducing a new detection algorithm.

This work evaluates a browser extension which classifies operations performed by browser in terms of risk threshold. Some operations include dynamic evaluation, cookies access, child-list mutation, rapid clicking, external referrer, and dynamic script injection. These actions can occur during legitimate and prohibited

operations. Same operation can be allowed in one case and forbidden in another case, yet the detector does not get any information about task permissions.

The evaluation separates four outcomes: observation, risk classification, visible warning or overlay, and independently verified task completion. A warning does not by itself show that an action was prevented. The tested fileless-attack indicators are controlled micro-behaviours associated with browser-side execution; they are not complete attack lifecycles. Claims are consequently limited to the bounded, reproducible capabilities of the evaluated extension and the specified workloads.

Research Questions and Contribution

The evaluation addresses the following research questions:

1. What are the observable families of browser behaviors associated with the extension among the primary scenarios under control?
2. How does the correlation between the risk-based decisions, warning and overlays provided by the extension and independently observed task completion on researcher-controlled benign and unauthorized actions occur?
3. What impact do the timing of the startup, frame location, and whitelist have on the observations in the tested setup?
4. What are the security and functional consequences of the tested warning/overlay and constrained NoScript deny/permit approach?

In this research, the method used to evaluate the effectiveness is a repeatable mechanism-level evaluation approach instead of introducing a new detector algorithm. Some of the experimental controls include frozen runs, fixture hashing, new browser profiles, authorization-context pair, separate oracle for completion, probe for lifecycle/frame, and separate logs for observation, decision, observable action, and execution. These will help differentiate whether the constraint is due to missing context, insufficient observation, decision-making process, or inadequate action.

RELATED WORK AND SYSTEM CONTEXT

Related Work

The research into the area of JavaScript browser security includes such areas as heuristic monitoring, structure classification, trace of execution, information flow analysis, extensions analysis, and intervention. The IceShield approach included such elements as browser hooking, heuristic decisions, and freeze [6]. Contrary, ZOZZLE and JStap included into their analysis of lexical properties, abstract syntax tree properties, control-flow properties, and data-flow properties of JavaScript [3], [4]. It is clear that the mentioned above elements show that browser hooking and numerical scores are proven approaches.

The scope of observations is defined by where and how the instrumentation takes place. In particular, VisibleV8 performs the observations on access to native APIs and properties in V8, while FV8 observes the conditional behavior through forced execution [7], [8]. JSFlow collects the information through JavaScript and browser API models [5]. Newer approaches show the better context model, where Arcanum explicitly uses sources and sinks for observing page-content flows through new Chrome extensions [10], and NoT.js builds the model of function-level execution context to distinguish tracking from functionality [2]. Since they serve different purposes and gather different observation units, their performance measures are not comparable with the one presented here.

Evaluation design is important. TESSERACT shows how sampling and partitions can cause bias in malware classifications [9]. JavaScript blocking research involves not only the effect of such blocking but also possible damage to legitimate websites [1]. Modern web measurements prove variability of JavaScript execution between platform devices [11]. Current researches about deobfuscation involve structure analysis and control flow execution of programs and not only activity enumeration [12].

The current research represents the assessment of bounded rule-based extensions. The importance of this research resides in the usage of frozen control applications, authorization-context pairs, lifecycle/frame probes, external completion oracles, and stored run traces for cases when there is a discrepancy between detector results and completed operations. The current study does not attempt to represent the discussion of information flow tracking, a new classification approach, forcing execution, and a cutting-edge malware detection technique.

Implemented System

The tested extension includes four modules: background module, content script, page injection monitor, and dashboard. The injection monitor is used for tracking dynamic evaluation and cookie access, while the content script monitors mutations of child list, click timing, referrer origin, and inserted script nodes. Since these events work on separate units of observations, the number of events of each of them is tracked separately.

As for the inspected implementation, in case the websites are not included in the registry, then the base increments of risk values are estimated as follows: 20 points for dynamic evaluation, 15 points for cookie access, 10 points for DOM events, 10 points for qualifying rapid clicks, 5 points for an external referrer, and 15 points for script insertions. In accordance with the rule-specific readiness, suppression, decay, and whitelisting requirements, the total score cannot be counted as the sum of API calls. Scores of 30 or below are classified as SAFE, scores above 30 through 80 as SUSPICIOUS, and scores above 80 as DANGEROUS.

The asynchronously loaded domain list suppresses or reduces selected rules and changes score decay. Detector readiness therefore had to be observed before releasing most tasks. Alerts, overlays, scores, and task completion were logged independently. Dashboard resource readings were not treated as extension-specific overhead measurements.

The original method remains in use. The separate version correction fixes the category/score and overlay status recording problems without changing any scoring weight, threshold, decay function, or whitelist functionality. Hence, the correction represents the internal consistency fix instead of being a new detection method. This assessment makes descriptive comparisons with upper limits only. It does not claim population-level validity.

METHODOLOGY

Study Design and Evaluation Boundary

A controlled mechanism-level analysis was performed by Chrome extension that keeps track of browser behavior in real time. The experimental design identified four phenomena that are often confused: observing browser behavior, risk assessment performed by the extension, presentation of a warning/overlay, and successful completion of the page task. The experiment did not include any live malware, infection, persistence, personal data, or system compromises.

Each browsing session was specified with a frozen manifest. For each scenario-configuration pair, three trials were performed in order to assess deterministic behavior stability; these were not considered independent samples of malware. Therefore, results are presented at the level of scenarios and as fractions of runs.

System Under Evaluation

The extension contains a background component, a content script, a page-injected monitor, and a dashboard. It tests six types of signal families: dynamic evaluation, cookie access, child-list mutation, rapid clicking, external referrer, and dynamically loaded scripts. As for the domains that are not listed explicitly, the inspected implementation uses the following baseline values of 20, 15, 10, 10, 5, and 15, correspondingly.

Scoring classifications are: a score of 30 or below is labeled as SAFE, the scores above 30 and less than 80 are labeled as SUSPICIOUS, and a score higher than 80 is labeled as DANGEROUS.

The Original arm retained the same scoring algorithm inside the instrumented experimental version. The Corrected-v1 only changed the consistency of recorded categories, scores, and overlays but left the signal weights, thresholds, decay, and whitelisting behavior unchanged. The detector-disabled arm offered functional controls. The limited intervention arm used NoScript 13.6.32 in deny and permit modes.

Scenario Construction and Researcher-Defined Labels

The controlled suite consisted of 20 scenarios: 12 researcher-defined benign tasks (B01–B12), four controlled unauthorized-action scenarios (A01–A04), three coverage probes (P01–P03), and one response probe (P04). For each scenario, the permitted or prohibited task was defined, along with the setup and task completion oracle. The same six detector families were used which were derived from the original six signal types.

Three scenarios involving different permissions but the same operation—B02/A03, B03/A01, and B11/A02—were used as the task-authorization input was not available to the detector. For each pair, the detector tested if the identical operations could lead to the different detection outcome. P01 involved comparing the startup execution with the execution when detector readiness was established. P02 involved comparing the same module execution at the top-level page and in a same-origin frame. P03 involved comparing the same action on unlisted and listed origins.

Scenario-intent labels were assigned by the research team before outcome interpretation from the written task authorization: actions explicitly permitted by the scenario specification were labelled benign, while actions explicitly prohibited by it were labelled unauthorized simulations. The completion oracles were based on task outcome and not on detector output. These labels and oracles were not independently verified; therefore, the results based on labels are descriptive in this created set only.

Experimental Configurations and Execution Controls

The frozen experiment contained 231 planned sessions across five configurations: detector disabled (C0), Original (C1), Corrected-v1 (C2), NoScript deny (C3), and NoScript permit (C4). The primary, context-information, coverage, and response panels used the detector configurations specified in the manifest. The intervention panel used B03, B11, A01, and A02 under the two NoScript policies. Run order was determined by a SHA-256 ordering of the frozen run descriptors.

In all cases, the tests used headful Chromium version 151.0.7922.34. Each test started with a new browser profile and a closed extension dashboard. Except for the startup case in P01, the detector waited until the ready-checkpoint was successfully completed before releasing the page task. In each session, a fxed observation window of 12 seconds was used. The runner logged task invocations, oracle outputs, signal observations, decision classes, any warnings and overlays, blocked states stored, events timestamps, errors, and deviations. All runs were retained. Infrastructure configuration failures could be repeated only without discarding the failed run, runtime failures remained outcomes unless the logs established an infrastructure cause.

Fixtures and the implementation plan were finalized using SHA-256 hashes before the first batch. The runner checked the specified artifacts and refused combinations not listed in the manifest. The smoke tests ran paths related to the representative detector, NoScript, the start point, listed origin, and the child frame before the entire batch. Development meetings that helped find the reason of the contamination of automation were kept separately.

Post-Review Dynamic-Web Extension

After peer review, a custom frozen extension created two additional good dynamic-web workloads without any change or consolidation of the initially 231-session experiment. R01 leveraged React 18.3.1 and ReactDOM 18.3.1 locally vendored to create one-page dashboard that had four asynchronous rerenders, 120 keyed rows, History API route change, and dynamic lazy-module import. R02 simulated the integration of third-party analytics by dynamically importing the script from another loopback origin which writes and reads dummy first-party identifier and dispatches the local pageview receipt event. Neither actual analytics service nor any user data were involved.

The frozen extension comprised 18 pre-specified sessions: two scenarios, three configurations—detector-disabled, Original, and Corrected-v1—and three stability repetitions per scenario-configuration cell. All sessions used the new profile, detector readiness check where necessary, 12 seconds of observation time, fixtures hashed as well as external oracle for session completion detection. The R01 oracle had the final dashboard route, lazy-module detection signal, completion signal, and 120 rendered rows. The R02 oracle needed completion signal and server-side pageview receipt.

Completion Oracles and Recorded Outcomes

Every experiment included an oracle which worked independently of the results obtained from the detector. As per the scenario, the oracle returned either an expected result or a marker value, precise DOM-items identification and number, input timing information, navigation and referrer information, resource trace information, bounded local receiver receipt, evidence of being whitelisted, or sequence of counters taken under overlay presence. A warning, score, or overlay was never considered as evidence of the completion or impairment of the task.

The results of the primary panel consisted of oracle independent of the output, detector alerts, warnings, overlays, maximal risk, recorded reasons, and detector version. The researcher-defined benign alert fraction used 27 runs: three repetitions for each of nine benign primary scenarios. A04 was reported separately as a controlled unauthorized simulation, and the matched authorization-context pairs were excluded from that fraction.

Integrity Checks and Analysis

Batch validation checked for run identifier uniqueness, repetition numbers, event stream hashes, frozen artifact hashes, P03 whitelist status, and oracle finish. The results were generated based on validated data, with no manual transcription. Tables 1 to 3 show the exact figures, and Figures 1 to 3 are summaries of benign alerting, differences in coverage, and NoScript trade-off.

Because the suite deliberately exercised selected rules, its descriptive fractions apply only to the tested scenarios and runs. No population inference is attempted here. A separate task-window performance pilot was retained as method-development evidence, but one synthetic workload and three trials per cell were insufficient for a final CPU or extension-specific overhead claim.

RESULTS

Batch Integrity and Task Completion

All 231 frozen manifest entries completed, producing 231 unique run identifiers and three repetitions for every planned scenario–configuration cell. Event-stream hashes, frozen-artifact hashes, and whitelist states passed validation. All 219 sessions outside the NoScript-deny configuration satisfied their applicable functional-completion oracle. None of the 12 NoScript-deny sessions satisfied its applicable script- or export-completion oracle, as expected for that intervention configuration. No launched main-experiment session was removed or replaced.

Primary-Panel Alert Behaviour

In nine of 27 runs of the researcher-defined benign primary scenarios, both versions of the detector triggered warnings (33.3%). Warnings occurred only in scenarios B04, B08, and B12, and were seen in all three instances for both versions of the detector (Table 1; Figure 1). B04, an approved local preference write/read task, reached a maximum risk score of 15 and resulted in warning without an overlay. B08, an approved rapid-click task, resulted in a warning in all runs. An overlay was displayed in 3 out of 3 Original runs and in 1 out of 3 Corrected-v1 runs, while its task oracle was successful in all 6 runs with the detector. B12, an approved five-module application, received a maximum risk of 73 and resulted in a warning without an overlay in all instances. The other researcher-defined six benign primary scenarios did not produce any warning or overlay.

The 33.3% value is a fraction within the selected challenge suite, not an estimate of the false-positive rate on general websites. The scenarios were intentionally constructed, repetitions assessed stability, and intent labels were assigned internally from task specifications.

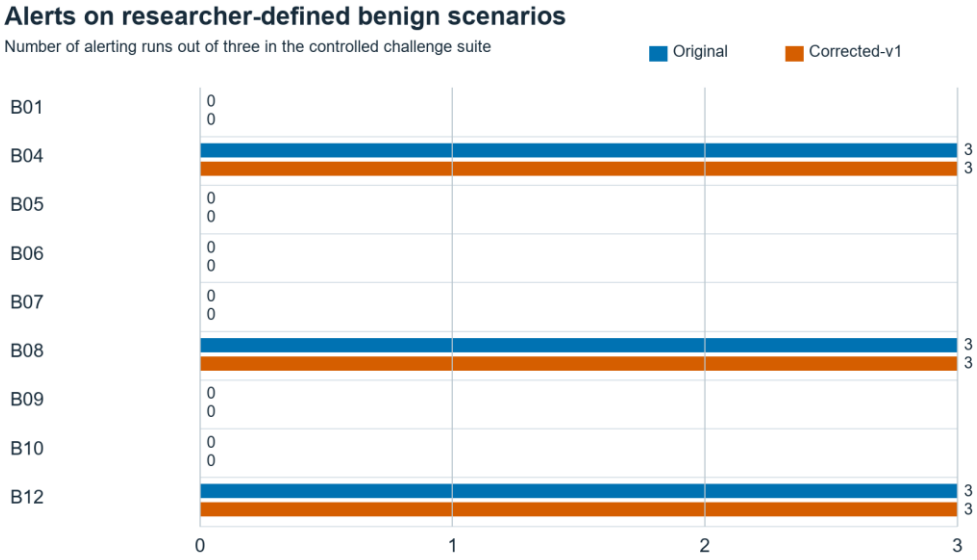
Controlled scenario A04 completed in all three runs for both detector versions. In each instance, there was a dynamic-script-insertion signal and a maximum risk score of 15, but no warning or overlay was produced by either of the detectors. Hence, the dummy action sequence was completed without a warning and without obstacle.

Table 1: Primary scenario outcomes by detector version

scenario	provisional_role	detector	runs	oracle_completed	alerts	warnings	overlays	max_risk_range	recorded_reasons
B01	benign	Original	3	3	0	0	0	20–20	Use of eval()
B01	benign	Corrected-v1	3	3	0	0	0	20–20	Use of eval()
B04	benign	Original	3	3	3	3	0	15–15	Cookie access detected
B04	benign	Corrected-v1	3	3	3	3	0	15–15	Cookie access detected
B05	benign	Original	3	3	0	0	0	0–0	None
B05	benign	Corrected-v1	3	3	0	0	0	0–0	None
B06	benign	Original	3	3	0	0	0	0–0	None
B06	benign	Corrected-v1	3	3	0	0	0	0–0	None
B07	benign	Original	3	3	0	0	0	0–0	None
B07	benign	Corrected-v1	3	3	0	0	0	0–0	None
B08	benign	Original	3	3	3	3	3	88–88	Non-human fast interaction
B08	benign	Corrected-v1	3	3	3	3	1	78–88	Non-human fast interaction
B09	benign	Original	3	3	0	0	0	0–0	None
B09	benign	Corrected-v1	3	3	0	0	0	0–0	None
B10	benign	Original	3	3	0	0	0	5–5	External referrer detected
B10	benign	Corrected-v1	3	3	0	0	0	5–5	External referrer detected

B12	benign	Original	3	3	3	3	0	73–73	Dynamic script injection
B12	benign	Corrected-v1	3	3	3	3	0	73–73	Dynamic script injection
A04	controlled simulation	Original	3	3	0	0	0	15–15	Dynamic script injection
A04	controlled simulation	Corrected-v1	3	3	0	0	0	15–15	Dynamic script injection

Figure 1: Alerts across researcher-defined benign primary scenarios



Authorization-Context Pairs

In all authorization-operation pairs, the detector behaved identically for both permitted and prohibited tasks. For example, B02/A03 achieved a risk of 20, B03/A01 achieved 15, and B11/A02 achieved 15; however, no warnings or overlays were generated by any pair.

These pairs show an information limitation that exist in the tested system design: the identical operations performed in the browser would generate the same evidence for the detector regardless of the different authorizations provided. The results do not show that authorization can be inferred from the operations alone.

Timing, Frame, and Whitelist Coverage

The probes identify coverage constraints associated with timing, frame location, and whitelisting. P01 recorded a maximum risk of 15 after verified readiness but 0 during startup; it does not measure the exact hook-installation time or a general startup-failure rate. P02 recorded 15 in the top page and 0 in the same-origin child frame. P03 reduced the same script-insertion task from 15 on an unlisted origin to 0 on the listed origin. External oracles confirmed completion in every condition, separating an unobserved action from an action that did not occur.

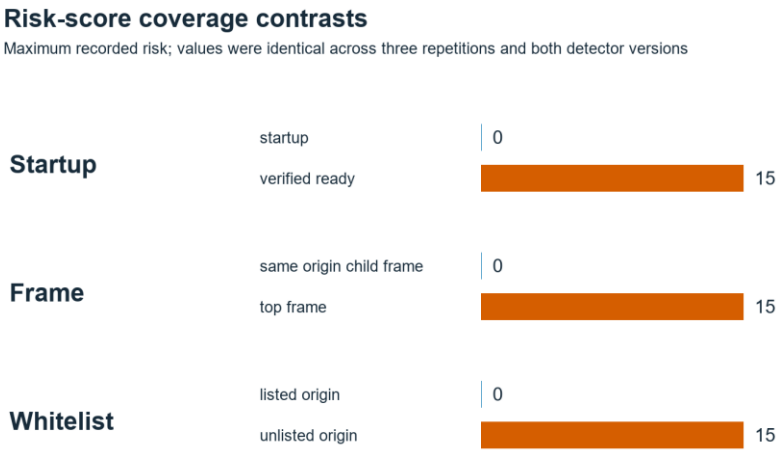
Overlay Response

P04 produced a warning and visible overlay in all three repetitions of both detector variants, reaching a maximum risk of 90. The continuation oracle nevertheless completed in all six runs because the scheduled counter continued to increase while the overlay was present (Table 2). The overlay therefore did not stop all previously scheduled JavaScript in this test.

Table 2: Coverage and response probes

probe	condition	detector	runs	oracle_completed	alerts	overlays	max_risk_range
P01	startup	Original	3	3	0	0	0–0
P01	startup	Corrected-v1	3	3	0	0	0–0
P01	verified-ready	Original	3	3	0	0	15–15
P01	verified-ready	Corrected-v1	3	3	0	0	15–15
P02	top-frame	Original	3	3	0	0	15–15
P02	top-frame	Corrected-v1	3	3	0	0	15–15
P02	same-origin-child-frame	Original	3	3	0	0	0–0
P02	same-origin-child-frame	Corrected-v1	3	3	0	0	0–0
P03	unlisted-origin	Original	3	3	0	0	15–15
P03	unlisted-origin	Corrected-v1	3	3	0	0	15–15
P03	listed-origin	Original	3	3	0	0	0–0
P03	listed-origin	Corrected-v1	3	3	0	0	0–0
P04	overlay-trigger-and-continuation	Original	3	3	3	3	90–90
P04	overlay-trigger-and-continuation	Corrected-v1	3	3	3	3	90–90

Figure 2: Timing, frame, and whitelist coverage contrasts



NoScript Intervention and Functionality

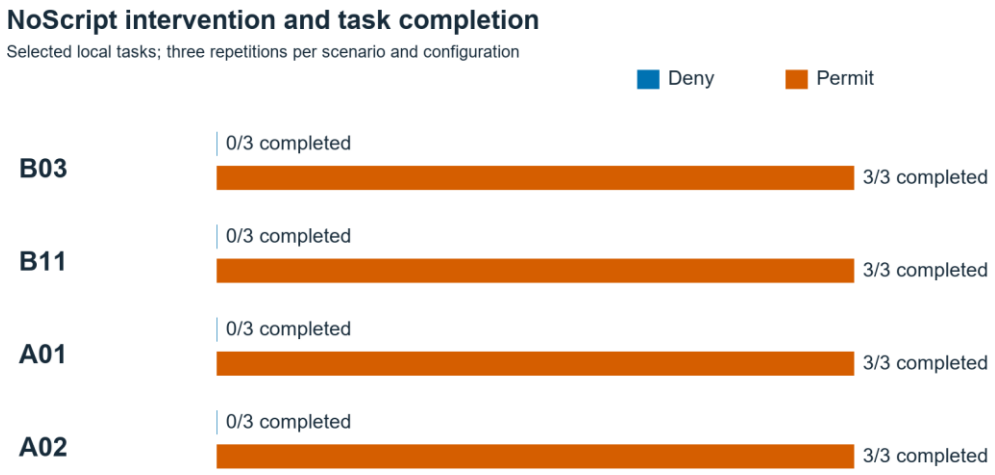
NoScript deny prevented task completion in all 12 selected deny runs, covering the permitted B03 and B11 tasks and their identically implemented unauthorized counterparts A01 and A02. NoScript permit allowed

completion in all 12 corresponding runs (Table 3; Figure 3). The result demonstrates enforcement accompanied by a functionality trade-off. It does not establish that NoScript classified the underlying intention.

Table 3: NoScript intervention outcomes

scenario	configuration	runs	tasks_completed	tasks_not_completed
B03	NoScript deny	3	0	3
B03	NoScript permit	3	3	0
B11	NoScript deny	3	0	3
B11	NoScript permit	3	3	0
A01	NoScript deny	3	0	3
A01	NoScript permit	3	3	0
A02	NoScript deny	3	0	3
A02	NoScript permit	3	3	0

Figure 3: NoScript task-completion and functionality trade-off



Dynamic-Web Extension Outcomes

After completion of all eighteen post review sessions, oracle requirements have been satisfied and tests passed for fixture and detector artifacts integrity check. In Run 01 (R01) the Original version showed maximum risk value 25 in all three attempts without giving any warnings, while the Corrected-v1 version had maximum risk value between 25 and 33 in all three repetitions with one warning shown. SPA tasks for React SPA were completed, and there was no any overlay. Causes for that were dynamic script insertion and excess DOM changes.

In Run 02 (R02) both detector versions had maximum risk value of 30 and gave warning in all three repetitions. Analytics tasks and local pageview receipt were completed, and neither version produced an overlay. The causes for that were cookie access and dynamic script insertion. Thus it is possible to conclude that common behavior of applications and analytics does trigger the tested heuristic rules without interfering with the intended application tasks performance. And the single warning in R01 for Corrected-v1 in one of three repetitions also shows the timing issue with batched DOM changes within detector evaluation period.

Original and Corrected-V1 Comparison

Original and Corrected-v1 produced the same numbers of researcher-defined benign alerts, A04 responses, and coverage-probe maximum-risk results. B08 overlay frequency differed: 3/3 Original runs and 1/3 Corrected-v1 runs displayed an overlay, although every run produced a warning and completed its task. Corrected-v1 should therefore be interpreted as a recording-consistency correction, not as a more accurate detector.

DISCUSSION

What the Detector's Alerts Represent

The results show that the prototype identifies selected browser operations rather than malicious intent. Three of the nine researcher-defined benign primary scenarios generated alerts in every repetition, covering legitimate preference handling, approved rapid clicking, and an approved five-module application. Conversely, the controlled A04 action chain completed without an alert even though the extension recorded one script-insertion signal. These outcomes indicate that the numerical score depends on which monitored operations accumulate and how the rules apply, rather than on whether the task is authorized or harmful.

It is important to mention that this restriction applies not only to the JavaScript monitoring technique. The problem lies in the information used for this particular implementation. Other systems provide richer representations. ZOZZLE and JStap rely on structure/semantics of the code [3], [4], JSFlow collects information about information flow between sources and sinks [5], while Arcanum and NoT.js use page-content flow/function-level execution context [2], [10]. They have different security goals and cannot be treated as performance baselines. However, they show that mere change in weight values on the same six signals would not bring the needed authorization context.

Dynamic Workloads and Heuristic Selectivity

The post-review workloads extend the observation beyond small static fixtures while preserving controlled oracles. The React SPA combined asynchronous rerendering, navigation-state change, and lazy script loading. Its task completed in every run, while warning behaviour varied by detector version and repetition. The analytics integration consistently produced warnings because cookie access and dynamic script insertion are normal parts of the controlled workload. These cases reinforce the main interpretation: operational signals may be observable and repeatable without establishing harmful intent.

The two workloads do not represent the diversity of production SPAs or commercial analytics systems. They instead provide reproducible counterexamples to any assumption that these signals uniquely characterize attacks. Broader deployment claims would require independently sampled applications, additional frameworks and libraries, and outcome definitions appropriate to each application.

Authorization as Unavailable Information

The three matched pairs provide the clearest data on the intentions. Each pair involves performing the same action using the observable browser interface according to two task descriptions: one permitted and one forbidden. The detector generated the same score and reaction in all the three pairs. As there was no authorized context supplied, there was nothing to differentiate between the two cases.

Thus, the possible design implication is limited to what is necessary in the future. There should be a reliable source of context, such as expected origin of scripts, permissible destinations for the data, user actions, or application-specific contexts. This source should be authenticated and should have a clear policy for resolving conflicts. This experiment shows that there is no specific approach to context awareness that is bound to be successful; it is known from the research on information flow [5] and current source-sink analysis [10].

Observation Coverage Is Part of Detector Validity

In examples P01 and P02 it can be seen that actions might be carried out despite the lack of assigned risk by the extension since the execution of the relevant action takes place beyond the observed coverage of testing. In both cases, the inserted module had the highest risk score of 15 in the post-check and on the top page, yet 0 for the startup stage and a same-origin child frame. External oracles confirmed completion under all circumstances. Such comparison separates a negative detector output from the absence of the action.

VisibleV8 and FV8 [7], [8] use a larger amount of execution instrumentation and study executed-path visibility, showing that the observation layer is indeed a significant design decision. The current experiment does not include any direct comparison of its extension with VisibleV8 and FV8. The novelty of the experiment consists in the explicit consideration of lifecycle and frame coverage within the prototype evaluation. Future claims about the efficiency of the detection mechanism must clarify observable documents, frames, workers and lifecycle stages.

Whitelisting Changes Visibility Rather Than Establishing Safety

In P03, the intentional inclusion on the whitelist changed the maximum risk of the exact same script insertion task from 15 to 0. The result is consistent with the rules of the prototype, but has one important interpretive implication: a lower score in this case indicates the trust configuration, not any independent validation of benign behavior. Thus, a corrupted listed origin or some third-party resources might require different trust judgments than those suggested simply by origin inclusion.

No evaluation of a production reputation service or calculation of whitelist failure rates was performed in this study. The result defines the whitelist as a boundary of evidence for the detector. Further evaluation will have to determine what evidence will be collected about listed origins, and how trust can be applied to origins, resources, script origin or an action.

Warning, Overlay, and Prevention Are Different Outcomes

P04 produced warnings and overlays in every detector repetition, yet the scheduled counter satisfied its completion oracle. The overlay was therefore a user-interface response, not evidence that the underlying execution had stopped. A result should be described as blocked only when an action-specific oracle shows non-completion.

For the NoScript panel, the opposite trade-off holds: denial mode blocks all 12 task completions, while permit mode blocks none. The blocked tasks include both permitted tasks and the same prohibited ones. It is enforcement indeed, but it is carried out by disabling capabilities and could interfere with legitimate functionality. In conclusion, the security benefits of the application and its functional utility need to be evaluated together [1].

A prevention mechanism needs a control which mediates a particular execution path and does not overlay the whole page. The Content Security Policy can allow scripts by sources, hashes, and nonces. Trusted Types can enforce the use of policy-created values in the selected injection sinks [13], [14]. Interception of requests or scripts is possible on a browser level and could suspend and cancel selected operations. However, this requires privilege and proper policy design. Each control operates at a different layer and can't substitute another one: CSP manages allowed resource execution, Trusted Types limit selected injection sinks, while interception manages selected browser actions. Any potential detector needs to detect the action, its source and an oracle of prevention.

Meaning of the Corrected-V1 Comparison

Corrected-v1 had the same number of alerts, A04 responses, and coverage-risk relationship when compared to the Original group. However, its difference in B08 overlay frequency was observed even with alerts raised and task completion throughout all the iterations. This is a clear indication that the code has been

modified to function correctly only in terms of being a tool to fix the issue of inconsistency in states but not a better detector. In other words, labeling Corrected-v1 as a better detection scheme would be misleading.

The above-mentioned results show why it was crucial to include event histories and oracles in the experiment. There might be discrepancies between the last recorded category, an overlay and actual task completion.

Research Contribution and Practical Implications

The main contribution of this research is in its evaluation approach and in the mechanism-level evidence it provides. The research used a frozen run manifest, hashed fixtures, new browser profiles, explicit scenario panels, authorization-context pairs, independent completion oracles, coverage probes, and persistent event logs. This approach allows seeing limitations that would go unnoticed in an evaluation where the only verification criterion was a warning existence.

For the developer community, the lessons learned from the experiment are about the need for early and context-specific instrumentation, explicit provenance and policy inputs, logging even in cases when exception in the trust is made, and evidence of the enforcement of specific actions. For evaluators, these results validate observing, classifying, responding, completing tasks, and the cost of legitimate function as separate results. The available evidence is not enough to provide the accuracy of the population malware detection or to say that this prototype is better than any existing systems. The scientific value of this paper is in its transparent case study and evaluation pattern that can be used elsewhere.

LIMITATIONS

The main evaluation used 20 purposefully constructed local scenarios, and the post-review extension added only two controlled dynamic-web workloads. Although the latter included a React SPA and a cross-origin analytics integration, neither set represents a probability sample of production websites or independently sourced malware. The four controlled unauthorized-action scenarios used dummy operations and did not reproduce an end-to-end fileless attack involving exploitation, persistence, credential compromise, or endpoint execution. The findings therefore characterize the tested mechanisms and workloads rather than general fileless-malware detection accuracy.

Second, three repetitions were conducted per scenario-configuration cell in order to test for the stability of runs, but this methodology did not provide for the creation of independent scenarios or observations from a larger population of threats. It is essential to note that the observed researcher-defined fraction of benign alerts 9/27 was calculated for a particular chosen panel only and cannot be considered as a population-level false positive rate. The controlled experiment setting allows for the precise comparison of descriptive measures within the suite, but does not provide population-level estimates of sensitivity, specificity, prevalence, or error of deployment.

Third, both scenario intents and completion oracles were internally developed and not reviewed by anyone externally. Since scenario intents were developed based on the permission letters and oracles were shielded from detectors' output, the classification is based on the given task definition and, hence, the research considers the classifications to be researcher-defined ones. Thus, label-based results are restricted to the tested scenarios.

Fourth, the study evaluates one self-designed extension with fixed heuristic weights. Corrected-v1 improves recorded-state consistency but does not change the detector weights, thresholds, decay, or whitelist. NoScript is included only as a limited deny/permit intervention comparator and is not a comparable behavioural classifier. The findings therefore cannot be generalized to all browser extensions or security products.

Fifth, the tests used headful Chromium 151.0.7922.34 in one local environment, with a fresh profile for every run. Browser/OS versions, device capabilities, extensions, page layout, cross-origin frames, workers,

and networking can all influence the results and responses seen. R01 and R02 target some SPA and analytics features, but they do not cover all the frameworks and environments that might be encountered in production systems.

Sixth, the readiness probe measures execution time relative to the observed readiness checkpoint, not the exact moment when all monitoring hooks were active. This test demonstrates a reproducible early execution counterexample in the test environment but not the general window during which the startup is blind or the failure rate. Likewise, the whitelist probe was based on an explicitly listed local origin. This test demonstrates how being in the whitelist affects the tested signal but not the actual errors in the reputation list.

Seventh, the P04 response probe revealed the fact that one benign scheduled counter is present in case of overlay usage. This result shows that tested overlay is not the same as stopping page scripts. It also does not prove that every malicious behavior will persist or that the overlay has no protecting power under other circumstances.

Eighth, the performance evaluation was exploratory in nature.. A single synthetic DOM benchmark was used with three trials per cell. Even though elapsed time method seemed to be possible to use, the browser-process memory data obtained correspond to full browser setup, whereas CPU counters were too coarse-grained for estimating overhead.

Future Work

The first technical goal is the improvement of coverage. Monitoring shall be launched as soon as possible inside the browser environment and be evaluated with regard to document start execution, top level documents, same-origin and cross-origin frames, workers, dynamically start execution contexts, and navigation events. Coverage evaluation must keep independent external oracles for distinguishing unobserved execution from failed execution.

The second goal is reevaluation of risk assessment with respect to sequences and context instead of individual operations. Evaluation of operations like eval, cookie access, DOM modification, clicks, referrer manipulation, and script injection may also occur in legitimate programs. In future evaluations, it is necessary to consider temporal relationships, source/destination relations, origin of script, frame identity, user actions, data flow, and site/organization policies. As it can be seen from the authorization-context pairs evaluated in this work, the threshold cannot provide information which is not present in the detector input. Thus, any policy aware construction must state where the authorization information comes from, how it is authenticated, and how it influences decisions.

Priority number three involves the need to distinguish between a warning and an enforcement overlay. The warning overlay should be described as a UI response unless a fully autonomous oracle shows that the action in question could not have been performed because it was blocked. Future designs will need to consider the security impact and functional interference of a variety of narrowly focused controls, including cancellation of a pending operation, isolation of a frame, prevention of a particular network request, and stopping of a particular script.

Performance evaluation should use several representative workloads, blocked or paired run ordering, more repetitions, observed sampler intervals, and a longer CPU-active task or higher-resolution instrument. Browser-process memory should remain described as a configuration-level quantity unless extension-specific allocation can be isolated. The experiment should also be repeated across supported browser versions and operating systems.

There needs to be a more thorough benchmark that includes independently gathered harmless programs, realistic attack traces, scenario families to hold out, and all within proper legality and ethics. The ground truth needs to be decided before analyzing the detector results and should include signs of successful actions. Analysis should cover per-family coverage, alerts, effectiveness of response, uncertainty, and cost

of functions. Comparisons should be done on maintained tools or explicitly defined baselines doing the same task with the same observation limits.

For whitelisting approaches, an explicit trust model is necessary. Further work can look into how whitelisting is done, the methods of how it gets refreshed and deletion, types of trust (i.e., originating from scripts, resources, actions), and what kind of signaling is disabled. Popularity should not detract from logging and forensics unless there is a good reason.

Future replication research need to include independent, blinded assessments of the scenario, family mappings, fixture behavior, and oracle adequacy. Any errors found in the assessment process must be recorded through versioning of the mappings and repeated runs under the specific scenarios.

CONCLUSION

This study used a reproducible controlled design to evaluate a rule-based browser-monitoring extension while separating observed signals, risk classifications, visible warnings or overlays, and independently verified task completion. The frozen main experiment completed all 231 planned sessions, and the post-review dynamic-web extension completed all 18 additional sessions without modifying the original evidence.

NoScript in this comparison demonstrates increased enforcement of the chosen script execution and export actions, while simultaneously prohibiting the permitted capabilities. Thus, the security measures in browsers should be evaluated through the security impact and possible application breaking, not through the mere presence of interference. Corrected-v1 made the recording more consistent without modifying the core detection rules and addressing the issues associated with them.

The extension responded selectively to operational browser events but showed limitations in intent, coverage, and enforcement. Three of nine researcher-defined benign primary scenarios produced alerts, while controlled scenario A04 completed without one. Matched permitted and prohibited tasks produced identical detector evidence because authorization was absent from the input. Startup timing, child-frame location, and whitelist membership changed observation. In the added workloads, the React SPA completed with no Original warnings and one Corrected-v1 warning, while the analytics integration produced warnings in every detector run; no overlay prevented either workload.

The defensible contribution is a mechanism-level case study and evaluation approach comprising authorization-context pairs, explicit coverage probes, realistic controlled workload extensions, independent completion oracles, frozen run manifests, and preserved event evidence. The study does not establish population-level fileless-malware detection accuracy. It instead provides evidence for improving observation coverage, authenticated context, heuristic selectivity, and action-specific enforcement.

ACKNOWLEDGEMENTS

The authors acknowledge the institutional support provided by Universiti Teknikal Malaysia Melaka.

Data and Materials Availability

The experimental materials and data are available from the corresponding author upon reasonable request.

Funding

This research received no external funding.

Conflicts of Interest

The authors declare no conflict of interest.

Ethics Statement

The study used controlled local browser scenarios, synthetic values, and no human participants, personal data, live malware, or compromised systems. Institutional human-subject ethics approval was therefore not applicable.

REFERENCES

1. Abdul Haddi Amjad; Zubair Shafiq; Muhammad Ali Gulzar. "Blocking JavaScript without Breaking the Web: An Empirical Investigation." *Proceedings on Privacy Enhancing Technologies*, Vol. 2023, No. 3, 2023, pp. 391–404. https://doi.org/10.56553/popets-2023-0087
2. Abdul Haddi Amjad; Shaoor Munir; Zubair Shafiq; Muhammad Ali Gulzar. "Blocking Tracking JavaScript at the Function Granularity." *arXiv preprint arXiv:2405.18385*, 2024. https://doi.org/10.48550/arXiv.2405.18385
3. Charlie Curtsinger; Benjamin Livshits; Benjamin G. Zorn; Christian Seifert. "ZOZZLE: Fast and Precise In-Browser JavaScript Malware Detection." *20th USENIX Security Symposium*, 2011. https://www.usenix.org/conference/usenix-security-11/zozzle-fast-and-precise-browser-javascript-malware-detection
4. Aurore Fass; Michael Backes; Ben Stock. "JStap: A Static Pre-Filter for Malicious JavaScript Detection." *35th Annual Computer Security Applications Conference (ACSAC 2019)*, 2019, pp. 257–269. https://doi.org/10.1145/3359789.3359813
5. Daniel Hedin; Arnar Birgisson; Luciano Bello; Andrei Sabelfeld. "JSFlow: Tracking Information Flow in JavaScript and its APIs." *Proceedings of the ACM Symposium on Applied Computing*, Vol. 2, 2014, pp. 1663–1671. https://doi.org/10.1145/2554850.2554909
6. Mario Heiderich; Tilman Frosch; Thorsten Holz. "IceShield: Detection and Mitigation of Malicious Websites with a Frozen DOM." *14th International Symposium on Recent Advances in Intrusion Detection (RAID 2011)*, LNCS 6961, 2011, pp. 281–300. https://doi.org/10.1007/978-3-642-23644-0_15
7. Jordan Jueckstock; Alexandros Kapravelos. "VisibleV8: In-browser Monitoring of JavaScript in the Wild." *Proceedings of the ACM Internet Measurement Conference (IMC 2019)*, 2019, pp. 393–405. https://doi.org/10.1145/3355369.3355599
8. Nikolaos Pantelaios; Alexandros Kapravelos. "FV8: A Forced Execution JavaScript Engine for Detecting Evasive Techniques." *33rd USENIX Security Symposium*, 2024, pp. 3747–3764. https://www.usenix.org/conference/usenixsecurity24/presentation/pantelaios
9. Feargus Pendlebury; Fabio Pierazzi; Roberto Jordaney; Johannes Kinder; Lorenzo Cavallaro. "TESSERACT: Eliminating Experimental Bias in Malware Classification across Space and Time." *28th USENIX Security Symposium*, 2019, pp. 729–746. https://www.usenix.org/conference/usenixsecurity19/presentation/pendlebury
10. Qinge Xie; Manoj Vignesh Kasi Murali; Paul Pearce; Frank Li. "Arcanum: Detecting and Evaluating the Privacy Risks of Browser Extensions on Web Pages and Web Content." *33rd USENIX Security Symposium*, 2024, pp. 4607–4624. https://www.usenix.org/conference/usenixsecurity24/presentation/xie-qinge
11. Ahsan Zafar; Junhua Su; Sohom Datta; Alexandros Kapravelos; Anupam Das. "Same Script, Different Behavior: Characterizing Divergent JavaScript Execution Across Different Device Platforms." *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, 2025, pp. 1889–1903. https://doi.org/10.1145/3719027.3765202
12. Dongchao Zhou; Lingyun Ying; Huajun Chai; Dongbin Wang. "From Obfuscated to Obvious: A Comprehensive JavaScript Deobfuscation Tool for Security Analysis." *Network and Distributed System Security Symposium (NDSS 2026)*, 2026. https://doi.org/10.14722/ndss.2026.242198
13. World Wide Web Consortium. "Content Security Policy Level 3." W3C Working Draft, 2026. https://www.w3.org/TR/CSP3/
14. World Wide Web Consortium. "Trusted Types." W3C Working Draft, 2026. https://www.w3.org/TR/trusted-types/