Improving the method of monitoring the state of the website by detecting user interface vulnerabilities and browser errors

Ihor Martyniuk[1,*,†], Hanna Martyniuk[1,2,†] and Yaroslav Yevchenko[3,†]

[1] *State Scientific and Research Institute of Cybersecurity Technologies and Information Protection, Maksym Zalizniak Str., 3/6, Kyiv, 03142, Ukraine*

[2] *Mariupol State University, Preobrazhenska Str., 6, Kyiv, 03037, Ukraine*

[3] *National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Beresteyskyi Ave., 37, Kyiv, 03056, Ukraine*

Abstract

The paper presents an improved method of monitoring the state of a website with a focus on the security of the client interface (front-end). An architectural solution is proposed based on the use of Selenium WebDriver in combination with Chrome DevTools Protocol (CDP) to emulate user interaction with the web interface, intercept internal browser events, and then transfer the collected data to the Zabbix monitoring system for further analysis and alerts. The methodology allows detecting security breach indicators, such as JavaScript errors, XSS injections, violations of CSP/CORS policies, cases of mixed content, vulnerabilities related to dangerous cookie flags, and other typical signs of compromise or incorrect user interface implementation.

Keywords

monitoring, Zabbix, Selenium, Chrome DevTools, website, UI vulnerabilities, monitoring automation

1. Introduction

Today, in the context of the rapid development of information technology and the growing digitalisation of business, the role of websites as one of the main channels of interaction with users, partners and customers is becoming increasingly important. Modern IT systems are constantly growing and becoming more complex and dynamic, which necessitates the implementation of effective solutions for monitoring their status and operation. Most monitoring solutions are focused primarily on the infrastructure level: monitoring server availability, network interfaces, resource consumption, performance metrics, etc. At the same time, the front-end part of websites, which is straight interacted with by the customers, remains out of focus of traditional monitoring [1].

In practice, it means that potentially dangerous security incidents or failures will remain undetected for a considerable period of time [2]. Such issues include JavaScript errors that prevent basic user interface functions from working; cross-site scripting (XSS) vulnerabilities that can be used to attack users; errors or incorrect implementation of content security policies (CSP); the usage of mixed content (HTTP resources on HTTPS pages), which decreases the entire security level; non-compliance with recommendations of correct cookies usage (lack of Secure, HttpOnly, SameSite attributes). In combination, these factors pose significant risks to both the reputation and cyber resilience of organisations providing services through websites.

In view of this, there is a need to create solutions that can monitor the client side of websites in real time, with a focus on security, stability and compliance with modern web development standards. Of particular relevance are approaches that allow integrating such solutions into the existing infrastructure

CH&CMiGIN'25: Fourth International Conference on Cyber Hygiene & Conflict Management in Global Information Networks, June 20–22, 2025, Kyiv, Ukraine

[*] Corresponding author.

[†] These authors contributed equally.

imartyniukiv@gmail.com (I. Martyniuk); ganna.martyniuk@gmail.com (H. Martyniuk); yevchenko.yaroslav@lll.kpi.ua (Y. Yevchenko)

0009-0003-5565-0828 (I. Martyniuk); 0000-0003-4234-025X (H. Martyniuk); 0000-0141-2256-8798 (Y. Yevchenko)

of an enterprise, ensuring continuous verification of the behaviour of the web interface in the production environment.

The purpose of this study is to improve the methods of monitoring the state of a website by means of an architectural solution that combines tools for emulating user behaviour with mechanisms for reading internal browser events, with subsequent analysis and transmission of the results to the monitoring system. This approach allows automated control over the state of the client side of websites, detection of indicators of security breaches or functional malfunctions, and prompt notification of potential incidents to the responsible persons.

2. Problem statement

Most website health monitoring methods usually focused on a limited set of metrics, such as resource availability monitoring (checking HTTP status codes), performance monitoring (tracking page load times, execution times of certain requests, etc.), and the use of external services (UptimeRobot, Pingdom, Datadog, etc.). There are also approaches based on static code analysis or server log checking, which are mostly focused on business logic or backend vulnerabilities and only partially address the behaviour of the client side.

This problem is especially relevant for one page websites that are created with React, Angular or Vue.js, where the client's logic is important for the correct operation of the interface. Request delays or form failures may cause critical errors that will not be detected by traditional monitoring tools.

That is why it is necessary to improve and develop new methods that allow you to solve these limitations and use monitoring with more efficiency by implementing integration with the browser interface, providing real-time tracking of various internal events and errors that impact the stability, security and functionality of the website front-end.

3. Analysis of recent research and publications

In recent years, the website monitoring industry has seen the emergence of many individual solutions, each of which addresses a part of the security problem. At the same time, most of them are focused either on the backend or on the initial stages of CI/CD (Continuous Integration/Continuous Deployment), while active monitoring of the client layer in the production environment remains poorly understood.

Most approaches, such as integrating tests into CI/CD [3, 4], proxy scanning (e.g., OWASP ZAP - a free tool for automated security testing of websites and web applications) [5, 6], or using Content Security Policy (CSP) [7, 8], help at the design or configuration stage, but do not capture incidents at runtime. The absence of CSPs or incorrect headers, although detected by tools such as SecurityHeaders (an online service for checking the security of HTTP headers) [9], are still only signals, not part of the operational analysis.

Modern monitoring systems, such as Zabbix 7.0 with Selenium support [10], already include basic accessibility tests. However, such tests are usually limited to checking HTTP responses only. At the same time, specific vulnerabilities at the DOM level, such as DOM-based XSS (cross-site scripting), uncontrolled redirects, or dangerous cookies, are not detected by such tools.

Some studies consider the possibility of using the Chrome DevTools Protocol (CDP - a set of tools from Google for remote browser control) for deeper analysis of browser behaviour [11, 12, 13], including interception of JavaScript errors, warnings about deprecated APIs, CSP violations, etc. This approach opens the way to building real-time client monitoring, but it is currently hardly implemented in standard security tools.

Approaches based on the MITRE ATT"&"CK framework (a matrix of known attacker tactics and techniques) remain relevant for modelling attack scenarios [1, 14, 15], but are limited to analysing backend or network activities. The use of automatic log handlers from the client that store and transmit the results through agents to Zabbix or other monitoring systems [16, 17] opens up new opportunities for integration into SIEM (Security Information and Event Management) [18, 19] and visualisation of vulnerabilities at the browser level.

Tools such as OWASP ZAP, ModSecurity with OWASP Core Rule Set (CRS), as well as Snyk (dependency and vulnerability analytics in code) or SonarQube (code quality analysis platform) [5, 6, 20, 21] are useful in identifying code or query issues, but do not have access to browser rendering.

In total, the analysis of sources allows to identify the following aspects:

- The growing popularity of DevTools API in scientific research [11, 12, 13], but the lack of production implementations;
- Emphasis on CI/CD verification without support for continuous monitoring [3, 4];
- Lack of integration between Selenium, DevTools, and the monitoring system in the security context;
- The need for automatic incident notification [16, 17, 18, 22].

These challenges are addressed by the architecture proposed in this paper, which combines UI monitoring, DevTools logic, real-time analysis, and centralised response via Zabbix monitoring system.

4. Results of the research

4.1. Architecture of the proposed solution

To solve the problem of detecting threats, errors and other security events on the user side, it is proposed to implement website monitoring at the level of user interface usage, based on the following components:

1 Selenium Webdriver – for launching a browser session and simulation human interactions.
2 Chrome DevTools Protocol (CDP) is a channel for accessing browser logs in real time.
3 Python script - initiates the launch of the Chrome browser in headless mode using Selenium. Then it checks internal browser events through the Chrome DevTools Protocol (CDP): JavaScript errors in the browser console, CSP, traffic, etc. The script can also run XSS tests or another custom scripts to inspect the functionality of a website. This script also performs error classification and generates statuses for the monitoring system.
4 Zabbix Server is the main monitoring system that receives the results of checks from the Python script and generates the appropriate triggers, alerts, and reports. It also analyses statuses from monitoring agents or external sources.
5 Notification System is a component for automatically informing responsible persons about detected problems by sending email, SMS or other types of notifications. Based on the specified triggers, Zabbix generates notifications that can be sent via API or webhooks for integration with third-party services (e.g. Jira, PagerDuty).

The proposed architecture is aimed at creating passive and continuous security monitoring at the UI level. The idea is to integrate user action emulation tools (Selenium) with Chrome DevTools interfaces to receive notifications of errors, unsafe actions, or suspicious behaviour without interfering with the application code. The results of the checks are sent to the monitoring system (Zabbix), which creates alerts and graphs. This allows for full monitoring in a live browser, including console errors, DOM events, CSP policies, and other risk signals that are not captured by traditional logging systems or a Web Application Firewall (WAF), as shown in Figure 1.

4.2. Testing methodology

The testing methodology consists of running a Python script that opens a web page in a real browser (Chrome) using analysis via DevTools Protocol. The received messages are filtered and classified. This ensures the collection of a comprehensive set of diagnostic data that allows you to identify both functional and security issues in the client side of the website. After that, the script analyses them (as shown in Table 1):

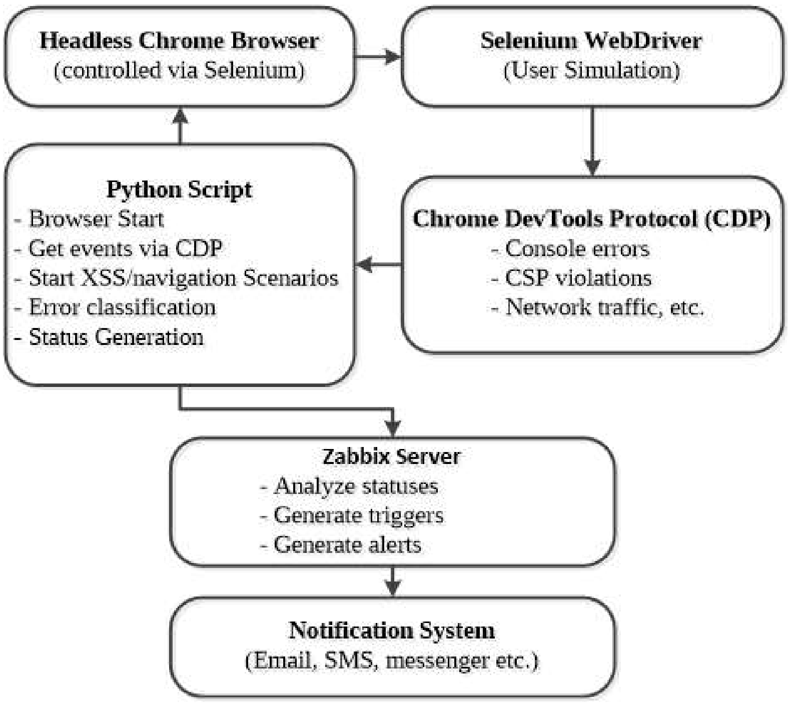

Figure 1: Architecture of the proposed solution based on Selenium, Chrome DevTools, and Zabbix API specifications.

Table 1
Security Test Cases

Category	Looking for	Risks	Expected detection
JavaScript Errors	Uncaught Exceptions, TypeError	logic block or data leakage	console.error with stack trace
DOM-based XSS	Embedding via innerHTML or eval	May lead to the execution of malicious code	DOM event + alert() in CDP logs
No CSP available	Insufficient Content-Security-Policy	Inline scripting vulnerability	Warning in the Security /CDP log tab
Mixed Content	HTTP resources on the HTTPS page	Compatibility issues, MITM	Error in DevTools about mixed content
Insecure	Cookies without Secure, HttpOnly	Ability to steal sessions	DevTools warning in the Application -> Cookies tab
Redirect Loops	Infinite redirects	Performance degradation, DoS	CDP: Navigation entries exceeded
Deprecated API	document.write, sync XHR	Security breach, warning in DevTools	DevTools warning about about deprecated usage

- whether there were any JavaScript errors;
- whether any security risks have been detected (insufficient CSP, mixed content, etc.);
- the extent to which the existing risk signs are repeated on different pages;
- the impact of load or other scenarios on the security of the interface.

The methodology involves automated page crawling using a headless browser, with the results analysed in DevTools via CDP. Each check is performed by a script and analysed by an external handler that generates a report. Then the result is transferred to Zabbix, where a trigger is generated. If the triggers are fired, notification system send message to responsible persons via a specified notification channel (email, Slack, SMS, etc.).

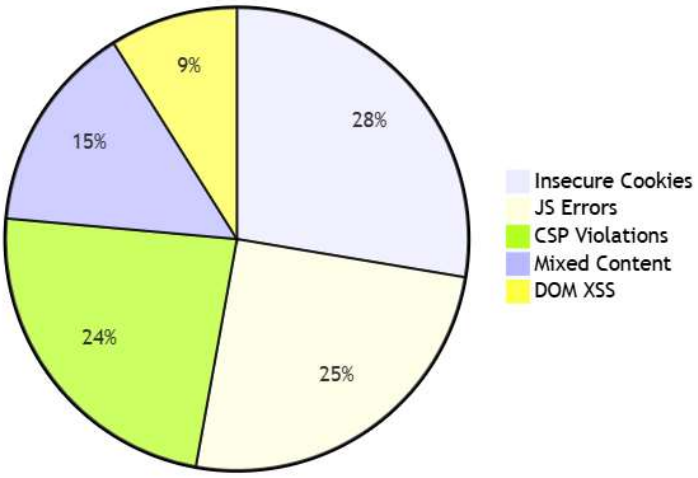

Figure 2: Testing results of UI incidents.

4.3. Testing procedure

The testing procedure includes:

- Connecting the Selenium WebDriver to a CDP-enabled browser.
- Run the scripts to open the landing pages.
- Collect logs through DevTools (console, network, security).
- Processing and classification of messages.
- Sending a summary status to Zabbix.
- Generating notifications, saving history for audit.

As part of the experiment, we conducted our own tests with a sample of 50 websites that were tested according to the above procedure, including corporate websites (React, Vue.js, Angular), e-commerce solutions (WooCommerce), information portals (WordPress/Joomla), forums (phpBB). The incidents were categorised as shown in Figure 2: CSP Violations, JavaScript errors, mixed content, DOM XSS events and insecure cookies.

The proposed approach can be used for providing risk assessment guidelines, for developing specific programs to improve the resilience of their websites, etc.

5. Conclusions

Analysis of capabilities and limitations of existing traditional monitoring methods shows that there is a huge number of different metrics, but they do not use tools for continuous UI monitoring. The proposed approaches for detecting threats at the UI level with including automation script provides continuous auditing without the needing of manual interaction.

Overall, the paper contributes to the improvement of monitoring methods, suggesting new approaches that can be useful for developers and site owners and will help responsible persons to prevent new categories of IT incidents and ensure fast detection of them.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

[1] SAIBERSOC, Mitre att and ck-based synthetic attack simulation, 2020. URL: https://saibersoc.com/blog/automated-threat-injection.

[2] M. Zaliskyi, R. Odarchenko, S. Gnatyuk, Y. Petrova, A. Chaplits, Method of traffic monitoring for DDoS attacks detection in e-health systems and networks, in: CEUR Workshop Proceedings, volume 2255, 2018, pp. 193–204.

[3] A. Kanaoka, S. Hiura, Real-time detection of multi-file dom-based xss vulnerabilities using static analysis, Proceedings of the 11th International Conference on Information Systems Security and Privacy 1 (2025) 191–198. doi:10.5220/0013109300003899.

[4] B. Garcia, F. Ricca, J. M. del Alamo, M. Leotta, Enhancing web applications observability through instrumented automated browsers, Journal of Systems and Software 203 (2023).

[5] Checkmarx, Zed attack proxy, 2020. URL: https://www.zaproxy.org/docs/.

[6] R. Filipe, F. Araujo, Client side monitoring techniques for web sites, Proceedings IEEE 15th International Symposium on Network Computing and Applications (2016) 363–366. doi:10.1109/NCA.2016.7778642.

[7] D. F. Some, N. Bielova, T. Rezk, On the content security policy violations due to the same-origin policy, Proceedings of the 26th International Conference on World Wide Web 2017 (2017) 877–886. doi:10.1145/3038912.3052634.

[8] W3C Web Application Security Working Group, Content security policy level 3, 2025. URL: https://w3c.github.io/webappsec-csp/.

[9] Security Headers, Analyse your http response headers, 2020. URL: https://securityheaders.com.

[10] J. Pikkarainen, What's up, home? – monitor your new selenium, 2024. URL: https://blog.zabbix.com/whats-up-home-monitor-your-new-selenium/28394/.

[11] Google, Chrome devtools protocol docs, 2020. URL: https://chromedevtools.github.io/devtools-protocol/.

[12] P. Chen, N. Nikiforakis, C. Huygens, L. Desmet, A dangerous mix: Large-scale analysis of mixed-content websites, Lecture Notes in Computer Science 7807 (2013) 354–363.

[13] J. M. Moreno, N. V. Rodríguez, J. Tapiador, Chrowned by an extension: Abusing the chrome devtools protocol through the debugger api, 2023 IEEE 8th European Symposium on Security and Privacy (EuroSandP) (2023) 832–846. doi:10.1109/EuroSP57164.2023.00054.

[14] D. Mohan, What is insecure https cookies and their risks?, 2024. URL: https://prophaze.com/blog/what-is-insecure-https-cookies-and-their-risks/.

[15] M. Zaliskyi, et al., Heteroskedasticity analysis during operational data processing of radio electronic systems, in: S. Shukla, A. Unal, J. V. Kureethara, D. Mishra, D. Han (Eds.), Data Science and Security, volume 290 of *Lecture Notes in Networks and Systems*, Springer, Singapore, 2021, pp. 168–175. doi:10.1007/978-981-16-4486-3_18.

[16] Selenium Project, Selenium documentation, 2025. URL: https://www.selenium.dev/documentation.

[17] J. C. Pazos, J. S. Legare, I. Beschastnikh, W. Aiello, Precise xss detection and mitigation with client side templates, arXiv preprint (2020). doi:10.48550/arXiv.2005.07826.

[18] P. Wang, J. Bangert, C. Kern, If it's not secure, it should not compile: Preventing dom-based xss in large-scale web development with api hardening, 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE) (2021) 1360–1372. doi:10.1109/ICSE43902.2021.00123.

[19] MDN, Secure cookie configuration, 2024. URL: https://developer.mozilla.org/docs/Web/Security/Practical_implementation_guides/Cookies.

[20] M. Squarcina, P. Adao, L. Veronese, M. Maffei, Cookie crumbles: Breaking and fixing web session integrity, Proceedings of the 32nd USENIX Security Symposium (2023) 5539–5556.

[21] P. Bernado, et al., Web platform threats: Automated detection of web security issues with wpt, 33rd USENIX Security Symposium (2024) 757–774. doi:10.5555/3698900.3698943.

[22] Y. Averyanova, et al., UAS cyber security hazards analysis and approach to qualitative assessment, in: S. Shukla, et al. (Eds.), Data Science and Security, volume 290 of *Lecture Notes in Networks and Systems*, Springer, Singapore, 2021, pp. 258–265. doi:10.1007/978-981-16-4486-3_28.