SSH Connection During Web Automation Testing

Oleksii Cherkashyn
Blynk Technologies Inc., Brickell Avenue 951, 33131 Miami, FL, USA
oleksii.sciencepapers@gmail.com

Keywords: SSH, Web Application Testing, Web Test Automation, WebdriverIO, JavaScript.

Abstract: Web automated testing has become increasingly relevant as web applications continue to grow in complexity and serve as critical components for businesses and organizations. Ensuring the reliability of server responses during test execution, as well as validating the presence or absence of uploaded files, is a key aspect of maintaining application integrity. This study demonstrates an approach using the WebdriverIO framework in combination with the ssh2 library to perform automated verification of server logs and file states in real time. The results show that web automated tests reliably detect both the presence and removal of files on the server, while the proposed solution proves stable across different operating systems, including Ubuntu, Windows 10, and macOS. Furthermore, it supports execution through CI/CD pipelines, such as Jenkins, enabling seamless integration into automated development workflows. Overall, this approach provides a practical and efficient method for enhancing the accuracy and reliability of web application testing, particularly in scenarios requiring server log monitoring and file validation. Across 790 test executions, the proposed solution demonstrated high stability, with 787 successful runs and 3 failures, resulting in an overall stability rate of approximately 99.62%.

1 INTRODUCTION

Web applications are now widespread, enabling essential operations across areas such as e-commerce, finance, healthcare, and education. With their growing complexity and scale, maintaining their reliability, performance, and security has become increasingly demanding. Web testing—which involves both verification and validation of web-based systems—remains a key factor in ensuring the overall quality of these applications [1].

In recent years, the number of websites has grown rapidly, making it essential to evaluate them against various quality criteria to ensure they meet expected standards [2]. Modern software development is marked by a rapid increase in the complexity and scale of web applications, which requires quality assurance (QA) specialists to adopt efficient testing methods to maintain reliability and security. Traditional manual testing approaches are no longer sufficient to satisfy industry demands, driving the adoption of automated testing frameworks [3]. The web applications is therefore usually more complex than testing other software products, and methods and tools must be found to make testing more efficient [4]. While automated web testing is well-developed, several unresolved challenges still exist. The present study contributes by addressing the verification of diverse server logs and the presence or absence of files on the server via SSH connection in the context of automated web tests. The WebdriverIO [5] framework was chosen as the basis for web automation due to its stability and reliability. WebdriverIO is a Node.js framework, available through npm [6], offering flexibility and compatibility with a wide range of testing tools and environments.

SSH (Secure Shell) offers users a secure channel to connect to computers over potentially insecure networks. It is widely used by system administrators to perform remote logins and execute commands through a command-line environment [7]. This study utilizes the SSH2 client module to establish SSH connections, enabling required access during automated web testing [8]. In this context, SSH connections target the Ubuntu servers [9] directly. The deployment of a web application on this or any other server is irrelevant, as the connection is needed only on the server where the required tests and verifications will take place.

2 RELEVANCE

Web applications now constitute a major part of software produced by organizations. Proper testing is vital to uphold their quality, as they are instrumental in enhancing brand visibility and improving communication with users [4]. Automated testing plays a vital role in modern software development, enabling faster releases and higher-quality software [10]. Web applications are especially difficult to test because of their dynamic nature, asynchronous behavior, and constantly changing interfaces - challenges widely recognized in the literature. As these systems rapidly evolve to meet user needs, testing approaches must also adapt to ensure reliability and efficiency [11].

Given the diversity of web applications, file upload functionality is often present in multiple parts of a system. In the context of web automation, automating file uploads is not always a straightforward task. A clear understanding of the fundamental steps involved in the file upload process is essential for identifying UFU vulnerabilities. The client-side handling of file uploads—such as allowing users to select a file and initiating its transfer—is defined by HTML and HTTP specifications implemented within the browser. In contrast, server-side processing may vary significantly depending on the web application's implementation and the configuration of the web server [12]. While it is certainly possible to validate the server response in automated tests (e.g., checking for a 200 status code) and perform visual verification, not all relevant information can be obtained from the browser console. For example, it is not always possible to confirm whether the file is stored in the expected directory on the server, whether its size matches the expected value, or whether duplicate files have been created. Similarly, in the case of file deletion, only direct verification on the server side can ensure that the file has actually been removed and that no residual copies remain.

A similar situation arises after executing automated tests: not all relevant information can be verified through browser logs alone. Although this largely depends on the server architecture, many systems maintain server-side log files. This study contributes by leveraging SSH connections during the execution of web tests to perform the kinds of server-side validations described above.

3 ANALYSIS OF CURRENT RESEARCH

In the context of server log verification in web automation, scientific studies are typically focused on validating server responses through browser network traffic or by analyzing logs obtained from the browser console after the execution of specific actions by automated web tests.

The authors Tunc and Kulcu, in their research, propose an approach for analyzing server logs to reconstruct user navigation paths in web applications. Their method includes preprocessing steps such as log cleaning, session identification, and path extraction to transform raw log data into structured sequences of user actions. By analyzing these sequences, the authors identify common navigation patterns and detect inefficiencies in user interaction with the system. The study demonstrates that server logs can serve as a reliable source of information about actual system usage without requiring direct user involvement. This approach highlights the potential of server logs for evaluating system behavior and supporting further analysis in web application environments [13].

The authors Tan, Yu, Leners, and Walfish propose an approach for verifying the correctness of web application execution using server logs in untrusted environments. In their work, server logs are treated as auxiliary and potentially untrusted data that help reconstruct the execution order and dependencies between requests. They introduce the Efficient Server Audit Problem, where logs are combined with recorded requests and responses to validate that the server behaved correctly. The proposed solution relies on re-executing requests and comparing the results with original outputs, using logs to guide and optimize this verification process. Additionally, they improve efficiency through reduplicated re-execution, allowing similar execution paths to be verified together rather than individually [14].

The authors Dharmaadia, Athanasopoulos, and Turkmen in their work survey fuzzing frameworks for server-side web applications and emphasize how server-side feedback, including logs and responses, can be used to assess application behavior. They discuss techniques for generating diverse request inputs that exercise different server paths and produce varied server log traces. The study highlights the importance of extracting meaningful feedback from the server, such as HTTP status codes and log information, to guide further fuzzing iterations. By analyzing how different fuzzing tools interpret and respond to server output, the authors illustrate how

logs can serve as a valuable oracle for detecting unexpected or erroneous server behavior. Overall, the paper proposes that effective fuzz testing must leverage server-generated information, including logs, to improve coverage and identify security and reliability issues in web applications [15].

Suryaningrat, Ramayanti, Taberima, and Kurniawan propose a multi-layered approach to secure file uploads that can be effectively applied in testing [16]. They emphasize validating file extensions and MIME types, which suggests testing that invalid or disguised files are properly rejected. They also highlight file size restrictions, making it important to verify boundary values and ensure oversized files are not accepted. The authors recommend sanitizing file names, so test cases should include special characters, long names, and potentially malicious inputs. In addition, they propose validating upload paths and storage locations, which can be tested by attempting unauthorized access or path manipulation. Finally, they suggest using unique identifiers (e.g., hashing), which implies testing that files are stored uniquely and not overwritten.

The authors Anwar, Fadlil, and Riadi, in their research, propose a multi-layered validation approach for SVG file uploads that can be applied in testing scenarios. They emphasize checking the file's magic number, ensuring that uploaded files are truly SVGs and not disguised malicious files. They also recommend analyzing the XML structure using the DOM, which allows testers to verify that files are well-formed and comply with SVG standards. Their approach suggests creating combined test cases that validate extension, MIME type, magic number, and DOM structure simultaneously to detect partially invalid or malicious files. They further highlight the importance of negative testing with corrupted or manipulated SVG files to evaluate the system's robustness. Finally, they propose black-box testing techniques to assess how the application handles potentially dangerous files from an external user perspective [17].

The authors Srimathi, Smilin, and Gowri [18], in their research, propose a security framework for direct user-to-cloud file upload systems, which can be applied in testing file upload functionality. They emphasize validating file types and sizes, ensuring that unauthorized or oversized files are correctly rejected. They also recommend testing that existing files cannot be overwritten without proper permissions. Their approach suggests creating negative test cases for invalid or forged temporary tokens, verifying that the system properly blocks unauthorized uploads. Additionally, they highlight the importance of checking server responses to incorrect or manipulated callback notifications after file upload. Finally, they propose testing all stages of the upload process - token generation, file transfer, and callback handling - to identify potential failures and security vulnerabilities.

It can be concluded that existing research is generally focused on validating uploaded files and server logs, primarily through responses observed in the browser console and network traffic. This study contributes to the field and addresses a research gap by introducing validation of server-side logs and uploaded files directly on the server via an SSH connection during the execution of web automated tests.

4 SOFTWARE CONFIGURATION

4.1 Test Infrastructure Configuration

The study utilized Node.js version 20.19.4, along with WebdriverIO version 8 for automated testing. The Mocha test runner was employed, and the test scripts were implemented in JavaScript [19]. Mocha provides a flexible and adaptable structure that can be tailored to various testing approaches, including support for asynchronous testing, which is essential for modern web applications that rely on dynamic content [20]. The full list of dependencies and libraries is specified in the package.json file, which is included in Appendix (Listing A1).

4.2 Preparing an SSH Connection

First, it is necessary to generate SSH keys. This task is handled by the built-in program ssh-keygen [21], which is available on almost all operating systems. The key generation can be performed using the command below. The example is given for an Ubuntu server, and it is important to run the command under the same user account that will execute the tests.

Bash command:

```
ssh-keygen -t rsa -b 4096
```

After executing this command, two files are created: id_rsa and id_rsa.pub in the ~/.ssh/ directory. The SSH public key from id_rsa.pub must be placed on the Ubuntu server where the web application is deployed. It should be added to the authorized_keys file located in the ~/.ssh/ directory for the root user. t is also important to ensure the security of the connection. Modern providers that offer virtual Ubuntu servers typically allow configuring access

restrictions in their web console. You should restrict access by IP address - specifically, allow only the IP of the Ubuntu server where the tests are executed, which will be connecting to the Ubuntu server hosting the web application.

4.3 Test Preparation

In Appendix (Listing A2), an example of the exported checkQuery function used in the test is provided. It is also important to copy the previously created id_rsa key into the appropriate directory, the path to which is specified in the code above. Within the sshClient.exec method of the function described above, any Ubuntu bash command can be specified. In this particular case, an example using the tail command is provided. After establishing the SSH connection, it monitors the log file in real time for the presence of Command 1 and also verifies the number of occurrences of this command after certain test actions are performed. An example of using the exported function in a test, as well as an example of the test itself, is provided below.

JavaScript code:

```
await checkQuery();
await browser.pause(1500);
const clickBtn = await $('#btn');
await clickBtn.click();
```

In the context of the example above, the function is executed first, establishing an SSH connection and reading the log file while searching for Command 1. Then, a 1.5 seconds wait is introduced, followed by a button click, with the expectation that the required command will be written to the log file after the click. This approach is implemented with consideration for the asynchronous nature of web tests. Within the previously described sshClient.exec method, the tail command can be replaced with any other Bash command. For example, it can be used to search for and count uploaded files during tests. The one-and-a-half-second delay is an average and approximate value, and it depends on the implementation and the server's speed in printing the expected commands to the logs. The command below, for instance, will check the number of files in the specified directory.

JavaScript code:

```
ls -1 uploads/files | wc -l
```

5 RESULTS AND DISCUSSION

In the context of this study, React web applications with a Java server architecture and an Express.js server deployed on an Ubuntu server were tested. The tests themselves were similarly executed on the Ubuntu server and on local machines running Windows 10 and macOS.

The results demonstrate a generally high level of stability of the proposed approach, although not all test executions were fully successful. A key factor influencing reliability is the specific implementation of server-side logging and the level of access to files stored on the server. The tests were executed through Jenkins, which was responsible for running the automated test suites, and this setup similarly demonstrated stable overall behavior. An important requirement is that SSH keys must be generated for the Jenkins user to ensure proper authentication and controlled access.

In this study, 158 tests were conducted on each server architecture described above, with each test executed five times, resulting in a total of 790 test executions per configuration. The study also showed that, for efficient real-time log monitoring, the tail command should include sufficient contextual information, such as IP addresses. However, the effectiveness of this approach strongly depends on the logging implementation to ensure that only relevant log entries are captured, without interference from other users' commands.

Although the overall system demonstrated high stability, three failures were observed during execution. One test failed due to the detection of a duplicate command in the log output, which affected the expected validation logic. Another failure was caused by an unstable internet connection, which interrupted the test execution flow. The third failure occurred because the test did not complete within the defined 1.5-second waiting threshold, which was likely also influenced by temporary network instability on the browser side. Overall, out of 790 test executions, 787 were successful and 3 failed, resulting in a stability rate of approximately 99.62%.

In addition to the approach based on generating SSH keys under the root account, scenarios involving users with restricted permissions were also analyzed. In general, the required permission set depends on the specific task being performed. For example, in the case of reading log files, granting read access to the corresponding files is sufficient. From a security perspective, however, the recommended approach is to generate and use SSH keys for a user account with limited privileges rather than for the root account.

The secure storage of private SSH keys remains a subject of ongoing discussion from the perspective of selecting the most effective security strategy. Since private keys provide direct authentication capabilities, their protection is critically important and requires strict access control mechanisms, secure storage policies, and minimization of unauthorized exposure risks. For access to the Jenkins web interface, it is strongly recommended to enforce IP-based access restrictions in combination with strong user authentication credentials, thereby providing an additional layer of protection against unauthorized access. Furthermore, periodic SSH key rotation should be incorporated into the overall security policy, as regular key regeneration reduces the potential impact of key compromise and limits the exposure window of leaked credentials. The optimal key rotation interval depends on the security requirements, infrastructure scale, and organizational risk model; however, security best practices generally recommend performing key renewal on a regular basis and immediately after any suspected security incident or personnel change.

6 CONCLUSIONS

Although some studies focus on server log verification and validation of uploaded files, this study contributes to the topic of SSH connections during the execution of web automated tests, as well as the verification of logs and files on the server. Based on the results of this study, the following conclusions can be drawn:

- Web automated tests using the WebdriverIO framework with the ssh2 library reliably perform verification of server logs and the presence or absence of files on the server after they are uploaded or deleted;
- The solution is stable, regardless of the operating system on which the tests are executed;
- The solution supports test execution through CI/CD pipelines, for example, using Jenkins.

REFERENCES

[1] K. Kapula, “A Survey on Web Testing: on the Rise of AI and Applications in Industry,” Journal of Information Systems Engineering and Management, vol. 10, no. 2, 2025, , doi: 10.52783/jisem.v10i2.9469.

[2] M. Kundra, “Selenium – A Trending Automation Testing Tool,” International Journal of Trend in Scientific Research and Development (IJTSRD), vol. 4, no. 4, pp. 1321-1324, Jun. 2020, [Online]. Available: https://www.ijtsrd.com/engineering/software-engineering/31202/selenium-–-a-trending-automation-testing-tool/manav-kundra

[3] A. Tymoshchuk, “The Evolution of Test Automation: From Selenium to Playwright. A Comparison of Automation Tools: Selenium vs. Playwright vs. Cypress,” International Journal of Computer (IJC), vol. 54, no. 1, pp. 37-45, Apr. 2025, [Online]. Available: https://ijcjournal.org/InternationalJournalOfComputer/article/view/2355

[4] S. Balsam and D. Mishra, “Web application testing - Challenges and opportunities,” Journal of Systems and Software, vol. 219, Jan. 2025, Art. no. 112186, , doi: 10.1016/j.jss.2024.112186.

[5] O. Cherkashyn, “Application Test Automation in Headless Android Emulator,” Proceedings of International Conference on Applied Innovation in IT, vol. 13, no. 5, pp. 445-452, 2025, , doi: 10.25673/123064.

[6] H. Sun, A. Rosà, D. Bonetta and W. Binder, “Automatically Assessing and Extending Code Coverage for NPM Packages,” in IEEE/ACM International Conference on Automation of Software Test (AST 2021), pp. 40-49, 2021, doi: 10.1109/AST52587.2021.00013.

[7] D. D. Tran, K. Ogata, S. Escobar, S. Akleylek and A. Otmani, “Formal Analysis of Post-Quantum Hybrid Key Exchange SSH Transport Layer Protocol,” IEEE Access, vol. 12, pp. 1672-1687, Dec. 2023, , doi: 10.1109/ACCESS.2023.3347914.

[8] A. Lekova, P. Tsvetkova, A. Andreeva, G. Dimitrov, T. Tanev, M. Simonska, T. Stefanov, V. Stancheva Popkostadinova, G. Padareva, K. Rasheva and D. Vitanova, “A Design Based Research Approach to Streamline the Integration of High Tech Assistive Technologies in Speech and Language Therapy,” Technologies, vol. 13, no. 7, p. 306, Jul. 2025, , doi: 10.3390/technologies13070306.

[9] S. Sadeq, W. B. Abdulaziz, R. N. A. Alsaadi and M. M. A. Algherini, “Enhancing Server Security: Implementation and Evaluation of the Port Knocking Method on Ubuntu Virtual Servers,” Journal of Engineering and Sustainable Development, vol. 29, no. 5, Nov. 2025, , doi: 10.31272/jeasd.2809.

[10] R. Barua, S. K. Ghosh and M. Andalibur Rahman, “The Future of Test Automation: A Comparative Analysis of Selenium vs. AI Driven Tools,” International Journal of Data Science, Bioinformatics and Cyber Security, vol. 1, no. 1, pp. 28-45, May 2025, , doi: 10.46610/IJDSBCS.2025.v01i01.003.

[11] M. Rahman, S. K. Ghosh and R. Barua, “AutoQALLMs: Automating Web Application Testing Using Large Language Models (LLMs) and Selenium,” Computers, vol. 14, no. 11, p. 501, Nov. 2025, , doi: 10.3390/computers14110501.

[12] S. Neef and M. Oudeh, “Bringing UFUs Back into the Air with FUEL: A Framework for Evaluating the Effectiveness of Unrestricted File Upload Vulnerability Scanners,” in Proceedings of the 21st International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA 2024), Lausanne, Switzerland, Jul. 2024, pp. 207-226, , doi: 10.1007/978-3-031-64171-8_11.

[13] S. K. Tunc and O. Kulcu, “A Web Application Path Analysis through Server Logs,” in Proceedings of the 10th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2018) - Volume 1: KDIR, pp. 427-430, 2018, , doi: 10.5220/0007231804270430.

[14] C. Tan, L. Yu, J. B. Leners and M. Walfish, “The Efficient Server Audit Problem, Deduplicated Re-execution, and the Web,” in Proceedings of the 26th Symposium on Operating Systems Principles (SOSP ’17), pp. 546-564, Oct. 2017, doi: 10.1145/3132747.3132760.

[15] P. A. Dharmaadi, E. Athanasopoulos and F. Turkmen, “Fuzzing frameworks for server side web applications: a survey,” International Journal of Information Security, vol. 24, article no. 73, Feb. 2025, doi: 10.1007/s10207-024-00979-w.

[16] A. Suryaningrat, D. Ramayanti, G. M. Taberima and P. P. Kurniawan, “File Upload Security: Essential Practices for Programmers,” CCIT Journal, vol. 17, no. 2, 2024, , doi: 10.33050/ccit.v17i2.3172.

[17] F. Anwar, A. Fadlil and I. Riadi, “Validation Analysis of Scalable Vector Graphics (SVG) File Upload using Magic Number and Document Object Model (DOM),” International Journal of Advanced Computer Science and Applications (IJACSA), vol. 11, no. 11, 2020, , doi: 10.14569/IJACSA.2020.0111133.

[18] S. Srimathi, K. Shanthini Smilin and G. Gowri, “A Security Framework for Direct User-to-Cloud Upload Systems with Automation for Vulnerability Detection,” International Journal for Research Trends and Innovation, vol. 10, no. 12, pp. a192-a198, Dec. 2025, [Online]. Available: https://ijrti.org/papers/IJRTI2512025.

[19] A. W. Putra and N. Legowo, “Greedy Algorithm Implementation for Test Case Prioritization in the Regression Testing Phase,” Journal of Computer Science, vol. 21, no. 2, pp. 290-303, 2025, , doi: 10.3844/jcssp.2025.290.303.

[20] E. Kesavan, “Comparative Study of JavaScript-Based Testing Frameworks Integrated with Selenium WebDriver,” International Scientific Journal of Engineering and Management, vol. 1, no. 5, Dec. 2022, , doi: 10.55041/ISJEM00087.

[21] F. Rascoussier, “Predicting SSH keys in Open SSH Memory dumps,” in University of Passau, Germany; INSA Lyon, 2024, p. 148, [Online]. Available: https://hal.science/hal-04562462v1.

APPENDIX

The Appendix contains the scripts and sections of programming languages code used in the study.

```
{
  "name": "app_name",
  "type": "module",
  "devDependencies": {
    "@wdio/cli": "^8.32.3",
    "@wdio/local-runner": "^8.32.3",
    "@wdio/mocha-framework": "^8.16.3",
    "@wdio/spec-reporter": "^8.16.3",
    "chai": "^5.1.0",
    "chromedriver": "^144.0.0",
    "geckodriver": "^4.3.0",
 "wdio-chromedriver-service": "^8.1.1",
    "ssh2": "^1.15.0"
  },
  "scripts": {
    "test": "wdio run ./wdio.conf.ts"
  },
}
```

Listing A1: Package.json.

```
import { Client } from 'ssh2';
import fs from 'fs';

const sshConfig = {
host: 'server_ip',
    username: 'root',
    privateKey:
fs.readFileSync('./sshkey/id_rsa')
};

export async function checkQuery ()
{
    const sshClient = new Client();
    let lastOutput = '';
    let outputCount = 0;
    sshClient.on('ready', () => {
    console.log('SSH connection
established');

    sshClient.exec('tail -f logs.log
| grep --line-buffered "Command 1"',
(err, stream) => {
    if (err) throw err;

    stream.on('data', data => {
    const output = data.toString();
    if (output !== lastOutput) {
    lastOutput = output;
    outputCount++;
    console.log('Output:', output);
    if (outputCount > 1) {
    sshClient.end();
    throw new Error (`Wrong count
Command 1. Count is ${outputCount}`)
    }
}

    }).stderr.on('data', data => {
    });
});
     setTimeout(() => {
     sshClient.end();
     console.log('SSH connection
closed after 3 seconds');
        }, 3000);
    })
    sshClient.connect(sshConfig);
}
```

Listing A2: JavaScript code.