

Enhancing Web Application Security:

Integrating SQLMap with OWASP ZAP



Abdullah Hasan

08.04.2025

(97249922)

Introduction:

Web application vulnerabilities remain a significant issue in cybersecurity. One of the most dangerous of these is SQL injection (SQLi), which allows attackers to interfere with the queries that an application makes to its database (OWASP, 2023). This project explores the integration of two powerful tools, OWASP ZAP and SQLMap, to enhance automated vulnerability detection processes. By combining the capabilities of ZAP's web scanning with SQLMap's in-depth injection testing, this integration aims to improve the depth and efficiency of SQLi detection.

The hypothesis that this project builds on is that bypassing ZAP's built in passive SQLi checks in favour of SQLMap's more precise testing engine, will result in a more accurate vulnerability assessment. This paper documents the implementation, evaluation, and future potential of this system.

Initial Assessment:

While looking through the original ZAP SQLi detection code, I noticed that ZAP only tests for the very basic forms of SQLi. For example when testing for Boolean-based SQLi it will specifically look for "SELECT * FROM users WHERE id = '\$id' AND '1'='1';".

The issue here is that an attacker could have the attack work by using any number of different boolean based attacks,

something as simple as blocking `1=1` as an input would still allow an attack like `400=400` and wouldn't flag as an alert on ZAP. This led me to test other kinds of SQLi attacks using other scanners causing me to see the gap in the software.

Project Scope and Objectives:

The scope for this project includes:

- Designing a methods for SQLMap to operate within a ZAP scanning workflow
- Extracting targets dynamically from ZAP's site history
- Running scans of detected URLs
- Comparing the integrated method against standalone tools

The primary objectives for this project is:

- Demonstrate technical integration using Jython
- Show clear results from SQLMap embedded in the ZAP environment
- Uphold cybersecurity ethics and responsible disclosure
- Offer potential enhancements and suggest real world applications

Background and Literature Review:

SQL injection is ranked among the OWASP Top 10 web application security risks (OWASP 2023). Despite its age, it continues due to improper input validation and inadequate testing tools.

OWASP ZAP (Zed Attack Proxy) is a widely used penetration testing tool designed to find vulnerabilities in web applications (Dahan, 2021). While it includes basic SQLi detection, its loosely defined method of trial and error often fails to confirm deeper or more complex injection vectors.

SQLMap, by contrast, is dedicated to SQL injection and database takeover. It automated the process of detecting and exploiting SQLi flaws and taking over database servers (Stasinopoulos, 2020).

Combining these two tools allows leveraging ZAP's crawling and interception capabilities with SQLMap's payload based testing engine. Other efforts like custom ZAP scripts and middleware integrations support this strategy but often lack ease of use or automation (Gupta & Sharma, 2021).

Implementation:

The implementation requires several phases:

Extracting Target URLs from ZAP

To dynamically fetch scan targets, ZAP's history was exported as a CSV file. A python (Jython) script was created to parse this and select the most recent, valid HTTP/S target.

```
1 def get_latest_site_from_csv(csv_path):
2     latest_url = None
3     latest_timestamp = None
4     try:
5         with open(csv_path, mode='r') as file:
6             csv_reader = csv.DictReader(file)
7             for row in csv_reader:
8                 current_timestamp = row['Req. Timestamp']
9
10                try:
11                    current_timestamp = datetime.strptime(current_timestamp, '%a %b %d %H:%M:%S GMT %Y')
12                except ValueError:
13                    print("[WARNING] Skipping row with invalid timestamp: {}".format(current_timestamp))
14                    continue
15
16                if not latest_url or current_timestamp > latest_timestamp:
17                    latest_url = row['URL']
18                    latest_timestamp = current_timestamp
19
20    except IOError:
21        print("[ERROR] The file {} was not found or could not be read.".format(csv_path))
22    except Exception as e:
23        print("[ERROR] An unexpected error occurred: {}".format(e))
24
25    return latest_url
```

Invoking SQLMap via Python

Using the URL fetched from the CSV, the script constructs an SQLMap command and runs it using os.popen(). This is to maintain jython compatibility.

```
1 site = get_latest_site_from_csv(csv_path)
2
3 sqlmap_cmd = "python C:/Users/abdu/sqlmap-dev/sqlmap.py -u {} --forms --crawl=2 --batch --risk=1 --level=1 --dos".format(site)
4
5 process = os.popen(sqlmap_cmd)
6 output = process.read()
7 process.close()
```

Sample output:

```
1 [INFO] Captured Site URL: http://testphp.vulnweb.com
2
3 [INFO] Running SQLMap scan...
4
5
6 ===== SQLMAP OUTPUT =====
7
```

Integration Validation:

To confirm that the integration has been successful and bypasses or superseeds ZAP's internal SQLi check and correctly

provides stand alone analysis to SQLMap, multiple dummy sites and test environments were used, like the one in the example above. <http://testphp.vulnweb.com>. Comparison was then made between ZAP's built-in findings and SQLMap's output.

Technical Rigor and Results

The project demonstrates how external python scripts can control ZAP through CSV exports and integrated deeper tools like SQLMap. Results showed SQLMap identified injection flaws that ZAP passive scanner did not.

Tool	boolean-based blind	error-based	time-based blind	Database Enumerated
OWASP ZAP	✗	✗	✓	✗
SQLMap	✓	✓	✓	✓(MySQL)

**This table is specific to the above website that I tested.*

Security Benefits:

- Deeper scanning due to payload-based injection attempts
- Avoidance of false positives common in heuristic systems
- Ability to script full scans without GUI interaction

Ethical Considerations

This project strictly avoided scanning unauthorised websites, like google.com or

instagram.com. Only safe testing environments like the previous site were used in a professional setting. Such tools must be used only with written authorization to avoid violating ethical or legal boundaries (EC-Council, 2021).

On top of this, the script does not automatically exploit databases but is limited to enumeration, aligning with responsible disclosure principles.

Limitations and Challenges

→ Timestamp format compatibility:

Parsing dates from ZAP CSV exports was unreliable due to timezone variations

→ OS Compatibility:

Integration required windows-specific paths, limiting portability

→ Output Handling/formatting:

SQLMap's output is dense and full of information. Parsing meaningful data requires further scripting

→ Full Automation / Integration:

ZAP as a whole is rather unstable and struggles to manage files. To fully integrate and allow SQLMap to trigger automatically further improvements in ZAP itself is required

Future work and Scalability:

There are several avenues for future improvements:

→ Direct API Integration:

Instead of relying on CSV exports, direct integration with ZAP's REST API could streamline target discovery

→ GUI Plugin:

A custom tab within ZAP's interface could allow users to launch SQLMap scans interactively

→ Output Parser:

A JSON exporter or output parser could convert SQLMap results into alerts that ZAP can display

This approach also had the potential to scale into a full vulnerability management workflow, integrating with CI/CD pipelines and automated testing systems.

Innovation and Comparative Analysis

Compared to already existing passive scanners or commercially available solutions like Burp Suite, this integration provides a greater depth through automation, flexibility due to its open-source nature and lastly its potential for full-stack integration with DevSecOps tools.

This approach is revolutionary in the way it works alongside ZAP's internal flow and embeds a third-party engine. It demonstrates a real benefit in detection

accuracy.

Security Testing Methodology:

To maintain the integrity and technical stance of this integration, a structured testing methodology was applied. The testing process followed a controller penetration testing framework which includes reconnaissance (recon), target acquisition, exploitation and reporting phases.

During the recon phase, OWASP ZAP's spidering and passive scanning features were used to identify accessible endpoints. Once ZAP completed the crawling process and stored the request history, the system moved into the target acquisition phase. My script was used to extract the most recent valid URL from the site history, ensuring that the scan was focused on a current active endpoint.

The exploitation phase was handled by both SQLMap and ZAP. ZAP ran through its pre-installed scripts to test for other exploits alongside SQLi. SQLMap uses advanced payloads to test for SQL injection vulnerabilities. I chose specific command-line flags like `--forms`, `--crawl=2`, `--risk=3`, and `--level=5` to increase the intensity while avoiding denial-of-service or aggressive fuzzing behaviour.

Thorough testing led to these specific flags, as any more intense would completely

stress the system and take too long to load.

Following this, the output of SQLMap was analyzed manually to determine the success of the injection attempts. Vulnerabilities were documented along with the database structure information disclosed by SQLMap.

This structured approach to testing ensured that the integration not only worked, but also adhered to recognised best practices in vulnerability assessment workflows (ENISA, 2023).

Comparison with other tools:

Several commercial and open-source tools were considered during this project for comparative analysis, including Burp Suite, Acunetix, and Nikto

Burp Suite Pro offers a highly sophisticated SQLi detection mechanism, including active scanning with built in payloads. However it is proprietary and not easily extensible in the same manner as ZAP and SQLMap. In contrast, the ZAP-SQLMap integration allows full control over testing logic and open-source transparency, which is preferable in educational or research settings.

Acunetix boasts comprehensive web vulnerability scanning features including SQLi. While it presents user-friendly interfaces and reporting, its detection engine is a black box, making it difficult to validate results or extend its features

programmatically.

Nikto is a lightweight command-line web scanner that lacks any real SQLi detection and focuses more on outdated software of misconfiguration. While useful in reconnaissance, it cannot replace SQLMap’s depth in injection testing.

The table below compares the capabilities of the tools:

Features	OWASP ZAP	SQLMap	Burp Suite	Acunetix	Nikto
SQLi Detection	Basic	Advanced	Advanced	Advanced	Limited
Custom Scripting	✓	✓	Partially	✗	✓
Open-Source	✓	✓	✗	✗	✓
Usability for Students	High	High	Moderate	Low	High
GUI Availability	✓	✗	✓	✓	✗

This integration offers the best of both ZAP and SQLMap, giving students and researchers a hands-on platform to study web security deeply.

Similarly when compared with these other standalone tools, my hybrid tool offers significant advantages in transparency, adaptability and cost efficiency. Unlike

proprietary tools that may hide detection rules or charge for advanced functionality, this open source solution enables full customisation and extensions. While ZAP's built in SQLi detection is functional it lacks depth and parameter testing flexibility that SQLMap is known for. By overriding ZAP's native SQLi checks with SQLMap this project delivers a more comprehensive and aggressive testing approach, bridging the gap between automation and accuracy in dynamic application security testing (DAST)

Real World Applicability:

Beyond the academic implementation, this integration of SQLMap with OWASP ZAP demonstrates strong applicability in real world cybersecurity operations. Many organizations employ DevSecOps pipelines that rely on automation and continuous integration of security testing. By combining SQLMaps proven injection testing capabilities with ZAPs robust interception and scanning features, that developed tool can seamlessly plug into CI/CD environments. For example, exporting report tools in machine-readable formats like JSON or directly triggering scans via the ZAP API enables the system to act as a proactive safeguard during code deployment. This approach not only minimizes manual testing effort but also ensures that common vulnerabilities like SQL injection are constantly checked against evolving web application code.

User Interface and Usability

Considerations:

As of now the technical backend integration is functional however the usability of the current solution is still limited to command-line interactions and manual CSV management. For wider adoption in an educational or professional environment, usability must be improved.

Future work can include designing a custom ZAP extension that adds a visual SQLMap launcher directly into the ZAP interface. This tab would allow users to:

- Select URLs from the site tree
- Choose scanning intensity and SQLMap options
- View scan output directly in a console like window within ZAP

These enhancements would streamline the user experience and allow non-technical users to run complex security scans without command-line knowledge. From an educational perspective this would help bridge the gap between theoretical knowledge and hands-on practice.

Additionally, exporting the results to formats like JSON or CSV would allow easier integration with reporting tools and dashboards. Automation scripts could parse the output and generate remediation suggestions, aligning the toolchain with common workflows in DevSecOps

pipelines.

Looking ahead, the integration could be further enhanced by incorporating machine learning models to assist in vulnerability triage and prioritisation. Currently both ZAP and SQLMap operate on predefined signatures and heuristics, treating each parameter with equal scrutiny during testing. By embedding a lightweight machine learning model trained on historical vulnerability data, the tool could predict which endpoints are most likely to be vulnerable and allocate scanning resources accordingly. This could significantly reduce scan times in large applications while increasing the detection rate of higher impact flaws. On top of this, such models could help differentiate between false positives and real threats, refining the accuracy of alerts raised during automated assessments. This evolution would increase the technical sophistication of the tool and improve its usability in enterprise environments.

Conclusion:

The integration of SQLMap with OWASP ZAP significantly improves SQL injection detection in automated web scanning processes. This project has demonstrated a functional proof of concept, evaluated its performance and proposed a path towards future extensibility.

Bibliography / References:

- Dahan, S. (2021) OWASP ZAP User Guide. OWASP. Available at: <https://www.zaproxy.org/>
 - EC-council. (2021) Ethical Hacking and countermeasures. Albuquerque: EC-Council Press.
 - Gupta, R. and Sharma, A. (2021) 'Automated SQL Injection Detection using SQLMap and Custom Tools', International Journal of Security and Networks, 15(4), pp. 205–213.
 - OWASP. (2023) OWASP Top Ten. Available at: <https://owasp.org/www-project-top-ten/>
 - Stasinopoulos, D. (2020) *SQLMap User Manual*. GitHub. Available at: <https://github.com/sqlmapproject/sqlmap>
-