Evaluating the effectiveness of Havij for structured query language injection exploitation in web applications

Mahmoud Baklizi[1], Mohammad Alkhazaleh[2], Musab Bassam Yousef Alzghoul[2], Adi Maaita[3], Jamal Zraqou[1], Mohammad AlShaikh-Hasan[4]
[1]Department of Computer Science, Faculty of Information Technology, University of Petra, Amman, Jordan
[2]Department of Computer Science, Faculty of Information Technology, Isra University, Amman, Jordan
[3]Department of Information and Communication Technology, King Hussein Cancer Center, Amman, Jordan
[4]Department of Computer Science, College of Engineering, Design and Physical Sciences (CEDPS), Brunel University of London, London, United Kingdom

Article Info

Article history:

Received May 25, 2025
Revised Oct 1, 2025
Accepted Oct 14, 2025

Keywords:

Havij
Structured query language injection
Structured query language injection tools
Web application
Website vulnerabilities

ABSTRACT

Structured query language injection (SQLi) is still one of the most critical risks to web application security, as it allows attackers to interfere with sensitive data and even a complete database infrastructure. Although many automated tools are available, previous studies usually achieve only descriptive briefs, which do not offer empirical assessments that measure the performance and the usability. This research fills this void by a systematic five-stage experimental analysis of the Havij automated SQLi tool under a controlled and ethical test setup. Confirmation of vulnerability, automated exploitation, data extraction and benchmarking of performance were performed as the methodology, and the results were compared against the industry standard SQLmap tool. It was found that in less than a minute Havij was able to locate the target database, scan its structure, and steal authentication credentials, which is quite efficient and user-friendly. In contrast to the literature, our work presents not only quantitative measures (time-to-exploit, request volume, and success rate) but also a qualitative evaluation (user accessibility and limitations), which gives a comprehensive evaluation. The results highlight trade-offs between the depth and accessibility, the continued dangers of SQLi in practice, and provide recommendations that developers and security experts can implement.

Corresponding Author:

Mohammad Alkhazaleh
Department of Computer Science, Faculty of Information Technology, Isra University
Amman, Jordan
Email: m.alkhazaleh@iu.edu.jo

1. INTRODUCTION

The prevalence of web applications by the government and commercial organizations has changed the service delivery in such a way that, effective utilization of electronic transactions has been facilitated [1]. They are technically structured on database management systems (DBMSs) in order to manage the operations of data with the help of structured query language (SQL) and tend to be developed in languages like PHP and JavaScript [2]. However, this interconnection has very severe security challenges. One of the most persistent and dangerous attack vectors is Structured query language injection (SQLi), and with the help of input fields, attackers can use harmful SQL code to obtain unauthorized access to, manipulate, or steal sensitive data-usernames, passwords, and financial data included [3], [4]. The effect is data breach and complete compromising of the database server which poses a significant risk to organizational integrity and user privacy [5], [6]. There is a great variety of tools developed by the cybersecurity industry to detect and mitigate SQLi vulnerabilities. They consist of open-source command-line tools with wide support of many different injection methods and database support to enable professional testers (SQLmap [7]) and commercial graphical user interface (GUI) scanners (Acunetix [8]) and Netsparker [9]) to be used in auditing at the enterprise level. According to the literature, an apparent trade-off exists between automation, power and accessibility. One such example is that, despite SQLmap supposedly being the de facto standard in its entirety [10], the command-line interface (CLI) is steep to learn. On the other hand, GUI-based tools like Havij [11] and jSQL injection [12] are hailed to be user friendly and simple to learn and operate, making the barrier to entry a major factor. Research, including that by Ibrahim and Kant [13], has empirically observed that tools such as Havij can be used to reveal vulnerabilities in an effective way, usually in combination with manual methods of initial discovery. The available body of work has been successful so far in listing available tools but usually in a descriptive manner in terms of features as opposed to a critical, empirical comparison of their operational effectiveness and their performance under controlled situations.

In spite of the plethora of tools, there is a major void in the empirical and comparative study of the actual performance of the tools. Most of the studies are descriptive but not structured in their evaluation process [14], [15]. Moreover, although the raw capability of a tool is a factor, its practicality, which is determined by such factors as speed, accuracy, and ease of use is equally important to security professionals when making tooling decisions. In particular, the implementation and performance rates of the Havij tool are not typically investigated in the literature with quantitative measures despite the frequent mentioning of the tool [16]. It is strongly lacking in recent, rigorous research that quantitatively compares such accessible tools to a baseline (e.g., SQLmap) on a standardized testbed to get a sense of their specific strengths, weaknesses, and place in the current security toolkit.

This paper seeks to fill these gaps by going beyond a descriptive enumeration to give an empirical assessment. Threefold our new contributions are:

- An experiential, gradual approach: we suggest and implement a step-by-step, five stage approach to systematic review of SQLi exploitation tools within a legal, ethical, and controlled setting.
- A quantitative and qualitative evaluation: we provide the quantitative (e.g., the time-to-exploit) and qualitative (e.g., the user experience) evaluation of the Havij tool, therefore, providing a comprehensive picture of its functioning in a typical testbed.
- Discussion on security implications: we summarize our results to discuss the wider implications of such easily available automated tools to offensive security testing and the changing threat environment and offer practical advice to developers and security practitioners.

The rest of this paper is structured in the following way. Section 2 provides the literature review and critically examines the available SQLi tools and their review in the literature. Section 3 explains the research method in terms of the experimental design, test environment, and five phase evaluation process. Section 4 summarizes and addresses the findings of the comparative study of Havij and SQLmap, both in terms of quantitative performance indicators and qualitative findings. In section 5, the paper will conclude by summarising the findings of the study, limitations of the study, and recommendation of future research.

2. LITERATURE REVIEW

This part critically summarizes the present state of knowledge about SQLi tools, their assessment in the scholarly and practitioner communities, and the defensive situation. Although many studies have enumerated the existing tools, there is still a huge gap as to the rigorous, empirical comparisons of their effectiveness, especially in those tools that are valued due to ease of access rather than raw power. Moreover, the literature tends to discuss offensive tools as discrete entities, and little is done to discuss these capabilities in relation to the contemporary defensive measures.

2.1. Critical overview structured query language injection tools

The SQLi detection and exploitation arsenal is varied, including both the advanced, open-source command-line tools, and commercial graphical scanners. A uniform trade-off of automation, power and usability can be identified in the academic and professional discourse. De facto testing standard is SQLmap [7], an open-source CLI tool that is highly rated in supporting databases (MySQL, PostgreSQL, MSSQL, Oracle, and SQLite) and types of injection (error-based, Boolean-based, and time-based). Its completeness is what makes it the best option when it comes to professional penetration testers [10], however, its CLI is a major barrier to its usability by new users. Conversely, the barrier to entry is reduced by GUI-based tools such as Havij [11] and jSQL injection [12]. Empirical observation has been made in studies like that by Ibrahim and Kant [13] that Havij has the ability to reveal the weaknesses effectively and has been utilized in quick initial analysis. But, as Priyanka and Smruthi [16] remark, such tools as Havij are often cited but have not been evaluated in the recent literature through rigorous and quantitative performance benchmarking. Moreover, Havij is not actively developed anymore, which casts doubts on its effectiveness against modern and defended systems.

Commercial tools such as Acunetix [8] and Netsparker [9] have enterprise level automation and features, and are priced at a high cost, but are not designed as a research tool and are instead designed to run audits of security within an organization. Other tools have narrower niches: SQLNinja [17] against targeted MSSQL attacks, BBQSQL [18] against blind SQLi and NoSQLMap [19] against new NoSQL databases. In addition, general-purpose web security testing tools, such as the OWASP zed attack proxy (ZAP) [20] and Vega [21], contain rudimentary SQLi detection functions as part of a wider set of vulnerability checks, and proxies such as Fiddler [22] are typically only used to test and debug web requests manually.

Table 1 includes a comparative summary of these tools. The decisive observation of this landscape is not simply the existence of tools, but the obvious contrast between tools of significant power and complexity (SQLmap) and tools of power and simplicity (Havij). Most of the literature enumerates these features [14], [15] but does not go further to provide a head-to-head, metrics-driven comparison of their operational performance, which is the gap that this study fills.

Table 1. Comparative analysis of SQLi tools

Tool	Automation	Interface	Supported databases	SQLi types supported	Free/commercial	Best for
Sqlmap	High	CLI	MySQL, PostgreSQL, MSSQL, Oracle, and SQLite	All major types (error-based, blind, and time-based)	Free	Professional penetration testing
Havij	High	GUI	MySQL, MSSQL, Oracle, and PostgreSQL (limited)	Basic types	Commercial (no longer updated)	Beginners, quick testing
jSQL Injection	Medium	GUI	MySQL, MSSQL, Oracle, and PostgreSQL	Boolean-based and time-based	Free	Lightweight testing
SQLNinja	Medium	CLI	Microsoft SQL Server only	Blind and error-based	Free	Targeted attacks on MSSQL
BBQSQL	Medium	CLI	Database-agnostic (blind SQLi only)	Blind-based	Free	Advanced blind SQLi testing
Burp Suite	Limited	GUI	All (via HTTP layer)	Limited (manual or via pro scanner)	Free/pro	Manual and automated web testing
OWASP ZAP	Limited	GUI	All (via HTTP layer)	Basic types via active scan	Free	General-purpose security testing
NoSQLMap	Medium	CLI	MongoDB and CouchDB	NoSQL Injection	Free	NoSQL injection detection
Netsparker	High	GUI	Most major databases	All major types	Commercial	Enterprise-level testing
Acunetix	High	GUI	Most major databases	All major types	Commercial	Enterprise and web app security
Vega	Limited	GUI	Limited (mostly web app layer)	Basic detection	Free	Educational or small-scale testing
Fiddler+ Plugins	Manual	GUI	Web-layer injection only	Manual injection	Free	Manual testing, debugging

2.2. Defensive context: beyond offensive tools

An effective analysis of offensive tools has to be understood within the defensive ecosystem that they are developed to circumvent. A layered defense-in-depth approach is gaining more and more popularity in the protection of modern web applications, which is often overlooked in tool-based reviews.

The most important defensive mechanisms that have a direct influence on the effectiveness of the tools in question are:

- Programming defenses: the most effective method of preventing SQLi at the source is the adoption of parameterized queries (prepared statements) [3], [5]. Lines of defense are added such as sanitization and input validation.
- Runtime application self-protection (RASP) and web application firewalls (WAFs): malicious inbound traffic may be filtered and blocked with the help of a set of predefined rules by installing such tools as the Modsecurity. Their progress will directly challenge the automated tools which will need to develop evasion mechanisms to be useful [23].
- Advanced detection systems: alongside the prevention, the more recent research is focused on the latest AI-driven intrusion detection systems (IDS). Recent research performed by Thalji *et al.* [6] on the AE-Net, a deep feature model that is an autoencoder, demonstrates that machine learning is capable of

detecting new SQLi attack patterns with the highest precision. Likewise, Baklizi *et al.* [4] have examined using ML in detecting the presence of web attack intrusions focusing on the moving target that the offensive tools have to deal with. Such defensive considerations are lacking in most tool-oriented reviews which is a major weakness. Any offensive tool depends on the defenses that it is subjected to, and this dictates its effectiveness. The performance of a tool in relation to an undefensible testbed, though useful in setting a baseline, does not indicate how useful the tool will be in a contemporary production environment.

2.3. Related work synthesis and gap identification

Some of the previous studies have provided a descriptive background but without the critical and empirical depth. Hajar *et al*. [14] offered a review that was based on Sqlmap; however, it concentrated on process description, but not quantitative metrics. Such studies as Umar *et al*. [24] and Crespo-Martinez [23] have integrated several tools and frameworks, but mostly in particular, non-generalizable settings (e.g., Academic Information Systems and Oracle APEX), and their results sometimes demonstrate the ineffectiveness of such tools against sophisticated defenses [23].

Vyamajala *et al*. [15] examined educational tools, finding the most popular to be Acunetix and jSQL, but it was not a performance evaluation of any kind. The availability of SQLmap on mobile platforms was also demonstrated by Fernandes and Lina [25] but once again methodology was emphasized as opposed to a quantifiable performance. The aggregate of the literature shows that there is a huge void: the lack of current and intense literature that will objectively compare the existing GUI tools (e.g., Havij) with the industry standard CLI tool (SQLmap) in a common testbed. Moreover, the majority of research does not critically address the implication of their results in the framework of contemporary defensive technologies, such as WAFs and artificial intelligence-based AI-based IDS [4], [6].

This paper will set out to fill this gap by offering more than a descriptive analysis, but an empirical, metrics based comparison of Havij and SQLmap, in terms of time efficiency, request volume, and success rate. It also adds by directly putting the results into context of the limitation of the study: that there are no defenses, and speaking of the implications to both offensive testing and the threat environment that has matured to include advanced defensive mechanisms.

3. METHOD

This research used a structured, 5 phase experimental research design to measure the effectiveness and efficiency of the Havij tool in vulnerability identification and exploitation of SQLi. In order to have a solid benchmark and respond to the requirement to perform a comparative analysis, we added to our methodology the parallel analysis with the industry-standard SQLmap tool. All these procedures were performed in a regulated laboratory setting so that it is reproducible and ethical.

3.1. Research design and experimental setup

The experimental design was a comparative case study in nature, whereby the performance of two different SQLi tools (Havij and SQLmap) was evaluated with reference to a single and widely known vulnerable target. It was aimed to quantify and juxtapose time efficiency, request volume, and success rate of each tool in performing the same exploitation tasks.

a. Test environment: all the tests were performed on a specific machine with Microsoft windows 10 Pro (Build 19045.4170). The network was a local area network of 1 Gbps, isolated to reduce the latency.
b. Tools and versions:
   - Havij v1.15: this variant was chosen because it is widely known and has been reported to have been used both in the academic and security circles. It was set to defaults except where otherwise.
   - SQLmap v1.7.11: the most recent stable version available at the time of testing was selected as the benchmark because it is the de facto standard when it comes to automated SQLi testing [7]. All tests were run from the command line.
c. Target application: the testing targeted the intentionally vulnerable web application testphp.vulnweb.com, which is the endpoint of the application, namely, the endpoint of the URL, the endpoint of the path, and the endpoint of the parameter, listproducts.php?cat=1. This is a known SQLi benchmark application maintained by Acunetix and is based on Apache/PHP/MySQL stack [8].

3.2. Justification for test site and limitations

The Acunetix test site was selected for this study for several key reasons:

a. Ethical and legal safety: it is specifically intended to be tested on security, and it does not carry any actual user information, therefore, it does not raise legal or privacy issues.

b. Controlled baseline: it gives a known and trustworthy environment, and a fair and reproducible comparison of tool performance without the confounding variables of evolving defenses or network state.
c. Technical relevance: its architecture (PHP/MySQL) is a typical set of web applications and, therefore, findings are applicable to a large portion of the internet.

Nevertheless, we clearly admit one main limitation: the absence of any protective measures (e.g., WAFs and input sanitization). Consequently, the results demonstrate the tools' performance in an ideal exploitation scenario and their raw capability. These findings can only be generalized to present, defended production systems to a limited degree, which is strongly discussed in the discussion section (section 5).

3.3. The five-phase method

The testing procedure was broken down into five sequential phases, executed first with Havij and then replicated with SQLmap. The method is summarized in Figure 1.

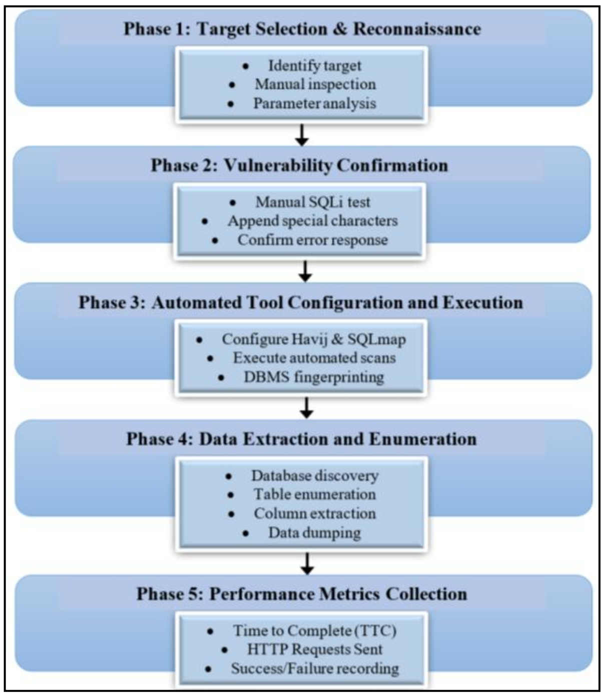

Figure 1. Five-phase method

3.3.1. Phase 1: target selection and reconnaissance

The target URL was determined and opened in a web browser (Google Chrome v123.0). The functionality of the application was learnt through a manual check. The cat parameter in the URL was also pointed out as one of the areas of injection since it is used to dynamically load data into the database.

3.3.2. Phase 2: vulnerability confirmation

An initial manual test was done to verify SQLi vulnerability. A single quote (') was appended to the value of the cat parameter (i.e., cat=1'). The return of a verbose SQL syntax error from the database (MySQL) confirmed that the input was not being sanitized and that the parameter was injectable.

3.3.3. Phase 3: automated tool set up and implementation

The exploitation of each tool was automated by ensuring that each was set up and implemented to operate in its respective operational paradigm. The particular operations were as follows:

– For Havij: the target URL was entered into the main address bar. The Scan button was pressed which automatically triggers the process of the tool to do DBMS fingerprinting, database enumeration, and data extraction using its own algorithms (mainly error-based and union-based techniques).
– For SQLmap: the following command was used to repeat what Havij did as closely as possible: sqlmap.py -u "http://testphp.vulnweb.com/listproducts.php?cat=1" -batch -level=3 -risk=1 -dbs. The

--batch option is useful to run it without interactive mode, the --level and --risk options regulate the depth and risk of the tests.

3.3.4. Phase 4: data extraction and enumeration

The two tools had the responsibility of carrying out a set of standard actions:

– Database discovery: finding out all databases.
– Table enumeration: displaying all the tables in the target database (acuart).
– Column extraction: extraction of the column structure of the critical users table.
– Data dumping: retrieving all the users table records.

3.3.5. Phase 5: performance metrics collection

For each tool and for each action in phase 4, the following quantitative data was recorded:

– Time to complete: the sum of the total time (in seconds) to complete the action successfully.
– HTTP requests sent: the amount of HTTP requests made by the tool to execute the action that are captured by the Burp Suite proxy [26].
– Success/failure: this is binary data of either successful completion or not of the action.

Such a metric-based, strict approach allows one to conduct the direct comparison that transcends the anecdotal data, providing empirical data about the properties of each tool in the working process. The findings of this comparative analysis are given in the following section.

4. RESULTS AND DISCUSSION

The following section will give the results of the comparative analysis of Havij and SQLmap on the chosen vulnerable test environment. The findings demonstrate the usefulness of each tool in the discovery of databases, table enumeration, column extraction, and credential retrieval. The quantitative (execution time, number of HTTP requests) and qualitative (insight into the trade-offs between usability, speed, and depth of exploitation) measures are examined.

4.1. Introductory context

This paper has examined the performance of the Havij tool in SQLi vulnerability attacks and it has also made a comparative analysis of the tool with the industry SQLmap. Although previous studies have listed SQLi tools [14], [27], and outlined their characteristics, a limited number have presented a rigorous, quantitative performance comparison benchmarking a user-friendly, GUI based tool to a powerful CLI based counterpart. Our methodology focused on this gap and measured time efficiency, request volume and success rate at key exploitation stages on a controlled testbed.

4.2. Summary of key findings

The gist of this study is that Havij is a very effective and efficient tool to be used in the automation to exploit SQLi in vulnerable and undefended targets. It was able to steal the full database structure and user sensitive credentials of the test application. However, the relative comparison to SQLmap showed that there is a major trade off, namely the simplicity and speed of Havij at the expense of transparency and control, and SQLmap is producing a larger number of HTTP requests to reach the same outcomes at a more detailed and reliable level.

4.3. Interpretation of results and comparison with literature

The two tools were able to follow through all the exploitation steps and this was a confirmation of the high vulnerability of the test application. Table 2 summarises the quantitative results of this process.

Table 2. Comparative performance of Havij and SQLmap

Exploitation phase	Tool	Time to complete (s)	HTTP requests sent	Success
Database discovery	Havij	~15	12	Yes
	SQLmap	~45	88	Yes
Table enumeration	Havij	~20	18	Yes
	SQLmap	~60	152	Yes
Column extraction (users)	Havij	~10	8	Yes
	SQLmap	~30	71	Yes
Data dumping (users)	Havij	~5	5	Yes
	SQLmap	~25	63	Yes
Total	Havij	~50	43	Yes
	SQLmap	~160	374	Yes

Havij was quite swift, he just made 43 HTTP requests and spent approximately 50 seconds to complete the whole exploitation process as shown in Figures 2 and 3. This is according to the results of Ibrahim and Kant [13] who established Havij to be highly skilled in identifying and exploiting vulnerabilities and to value it on fast analysis. This reflects its design philosophy: it was to be completely automatic to the user, so that complex attacks can be available to beginners. The tool automatically employed a combination of error based and union based SQLi in achieving its goal. On the other hand, the SQLmap was around three times slower (around 160 seconds) and generated around nine times the number of the HTTP requests (374). This is not a mark of inefficiency but is a mark of fullness and wordiness. SQLmap conducts a far more comprehensive set of tests and scans a wider range of injection techniques and provides a significant amount of real-time feedback to the user. This is slower, but more dependable and also flexible to a large range of and potentially obfuscated environments [7]. Our results support the characterization of SQLmap as the more potent and versatile tool, such as targeted and in-depth testing in the studies such as [23], [27].

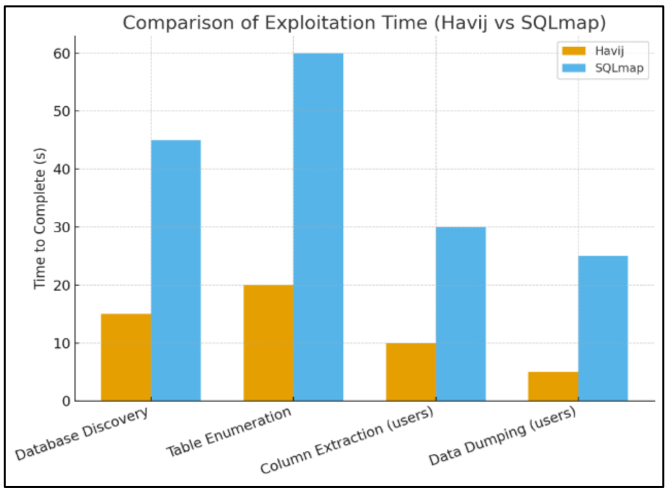

Figure 2. Comparison of exploitation time (in seconds) for Havij vs SQLmap across phases

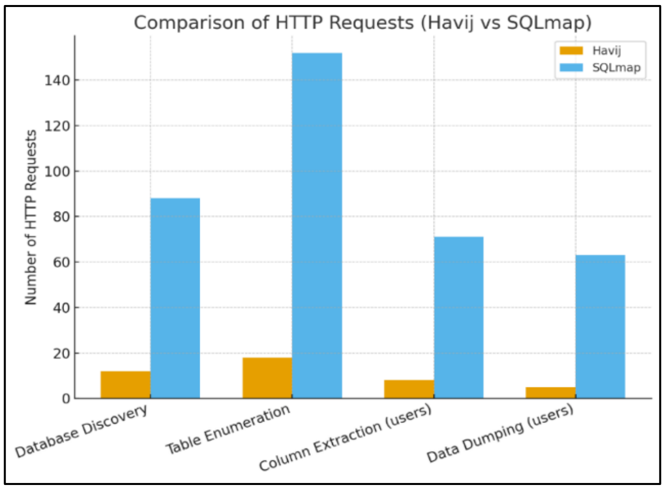

Figure 3. Comparison of HTTP requests sent by Havij vs SQLmap across phases

The fact that plaintext credentials (username: test and password: test) have been successfully retrieved in the vulnerable users table illustrates the severity of SQLi exploitation in the real world. This result shows that a mere unnoticed vulnerability can undermine authentication systems and reveal confidential information. The result also confirms the previous studies [4], [5], which have highlighted the potential real-world dangers of SQLi to the privacy of data and the general security of the system.

4.4. Addressing limitations

It is important to put these findings into perspective in the context of limitations of the study. To begin with, the fact that a single, deliberately weak test environment (testphp.vulnweb.com) is used implies that the results reflect the best tool performance when against a known, weak defense. This environment is not reflective of the sophistication of the contemporary, defended applications WAFs, advanced input sanitization, or object-relational mapping (ORM) engines. As a result, the performance measures presented here cannot be applied to such settings. Second, according to the literature [16], Havij is not actively maintained or developed any longer. Although it still works on older and unprotected systems such as the one tested, its applicability to more recent database versions (e.g., MySQL 8+), security features and application frameworks is probably severely restricted. SQLmap is actively managed and thus keeps up with new defenses. Lastly, the present work was dedicated to the analysis of exploitation features of an offensive tool. The initial title with the concentration on the detection systems was hence a misnomer. The offensive potential is tested in our work and not the defensive efficacy.

4.5. Implications for future research

The present research leaves some opportunities to work in the future. First, to extend these findings, these and other tools should be tested by a broader range of targets, including those secured by both modern WAFs (e.g. ModSecurity) and more general security mechanisms to test their evasion ability. Second, a user study that compares success rate and time-to-exploit of security novices on Havij and other tools such as SQLmap or jSQL Injection may offer interesting information on the usability of these tools in practice. Lastly, studies need to be conducted on hybrid solutions to utilize the simplicity of GUI tools with the capabilities and complexity of advanced and actively maintained tools, and the latest AI-driven scanners [6] to construct more efficient penetration testing processes.

4.6. End of the discussion

To sum up, this comparison analysis shows that Havij can still be a powerful and effective exploit to take advantage of SQLi vulnerabilities in unsecured environments, which explains why it was popularly used in the past to conduct quick tests. Nevertheless, this can be counterbalanced by its outdatedness and basic functionality in a professional security environment with its use in relation to contemporary targets. SQLmap is more advanced, time-consuming and demanding in terms of expertise, but it is also more detailed, reliable, and future-proof. These vulnerabilities being present in the educational environments, underscores the reasons why the developers ought to implement stringent security measures like parameterized queries, input validation, and regular security testing with updated and complete tool sets.

5. CONCLUSION

This research aimed at conducting a viable examination of how the Havij automated SQLi tool works in both a controlled and an ethical environment. It has been determined that Havij is quite handy in its intended application on the weak testbeds with an easy-to-use graphical interface and remarkable speed to accomplish a successful extraction of database structures and sensitive credentials. The significant contribution of the work is the establishment of an empirical base of the SQLi tool performance with the apparent trade-offs between accessibility and completeness. Furthermore, it also introduces vividly the state of a present and severe threat posed by readymade automated exploitation tools, which lower the barriers of entry of attackers. It is based on these that we strongly suggest that security experts and developers must take into consideration a defense-in-depth strategy in an attempt to counter such threats. This must include hard-coding like parameterized queries and extreme input validation. In addition, companies must require periodic penetration testing with a collection of modernized and sophisticated tools, and they must not depend on a single solution to provide the strong security against the changing environment of SQLi attacks.

ACKNOWLEDGMENTS

We honestly thank University of Petra and Al-Isra University the for allowing access to their Cyber Test center facilities and for the vital support that donated to the success of this investigate.

FUNDING INFORMATION

Authors state no funding involved.

AUTHOR CONTRIBUTIONS STATEMENT

This journal uses the Contributor Roles Taxonomy (CRediT) to recognize individual author contributions, reduce authorship disputes, and facilitate collaboration.

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Mahmoud Baklizi	✓	✓	✓	✓	✓	✓		✓	✓	✓			✓	
Mohammad Alkhazaleh	✓	✓	✓		✓	✓		✓	✓	✓	✓	✓		
Musab Bassam Yousef Alzghoul	✓	✓	✓			✓		✓	✓	✓			✓	
Adi Maaita		✓				✓		✓	✓			✓		
Jamal Zraqou	✓	✓		✓	✓			✓		✓			✓	
Mohammad AlShaikh-Hasan	✓		✓	✓			✓			✓	✓		✓	✓

C	: **C**onceptualization	I	: **I**nvestigation	Vi	: **Vi**sualization
M	: **M**ethodology	R	: **R**esources	Su	: **Su**pervision
So	: **So**ftware	D	: **D**ata Curation	P	: **P**roject administration
Va	: **Va**lidation	O	: Writing - **O**riginal Draft	Fu	: **Fu**nding acquisition
Fo	: **Fo**rmal analysis	E	: Writing - Review & **E**diting		

CONFLICT OF INTEREST STATEMENT

Authors state no conflict of interest.

DATA AVAILABILITY

Data availability is not applicable to this paper as no new data were created or analyzed in this study.

REFERENCES

[1] U. Patkar, P. Singh, H. Panse, S. Bhavsar, and C. Pandey, "Python for web development," *International Journal of Computer Science and Mobile Computing*, vol. 11, no. 4, pp. 36-38, 2022, doi: 10.47760/ijcsmc.2022.v11i04.006.

[2] N. Sharma, "Overview of the database management system," *International Journal of Advanced Research in Computer Science*, vol. 8, no. 4, 2017.

[3] J. H. B. Johny, W. A. F. B. Nordin, N. M. B. Lahapi, and Y.-B. Leau, "SQL injection prevention in web application: A review," in *Advances in Cyber Security*, Penang, Malaysia, 2021, pp. 568–585, doi: 10.1007/978-981-16-8059-5_35.

[4] M. K. Baklizi *et al.*, "Web attack intrusion detection system using machine learning techniques," *International Journal of Online and Biomedical Engineering*, vol. 20, no. 3, 2024, doi: 10.3991/ijoe.v20i03.45249.

[5] M. Baklizi, I. Atoum, M. A. S. Hasan, N. Abdullah, O. A. Al-Wesabi, and A. A. Otoom, "Prevention of website SQL injection using a new query comparison and encryption algorithm," *International Journal of Intelligent Systems and Applications in Engineering*, vol. 11, no. 1, pp. 228–238, 2023.

[6] N. Thalji, A. Raza, M. S. Islam, N. A. Samee, and M. M. Jamjoom, "AE-Net: Novel autoencoder-based deep features for SQL injection attack detection," *IEEE Access*, vol. 11, pp. 135507–135516, 2023, doi: 10.1109/ACCESS.2023.10332180.

[7] B. D. A. G andnM. Stampar, "SQLmap: Automatic SQL injection and database takeover tool," Sqlmap, 2011. [Online]. Available: http://sqlmap.org. (Accessed: Oct. 15, 2025).

[8] Acunetix, "Web vulnerability scanner technical documentation, 2023, [Online]. Available: Available: https://www.acunetix.com. (Accessed: Oct. 15, 2025).

[9] Invicti, "Netsparker web application security scanner," 2023, [Online]. Available: https://www.invicti.com. (Accessed: Oct. 15, 2025).

[10] M. Baklizi *et al*., "A technical review of SQL injection tools and methods: A case study of SQLMap," *International Journal of Intelligent Systems and Applications in Engineering*, vol. 10, no. 3, pp. 75–85, 2022.

[11] T. Baig, "Havij: Automated SQL injection tool," [Online]. Available: https://github.com/talhabaig007/Havij. (Accessed: Oct. 15, 2025).

[12] A. K. Mishra and A. Kumar, "Performance-based comparative analysis of open source vulnerability testing tools for web database applications," in *2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, Kharagpur, India, 2020, pp. 1-5, doi: 10.1109/ICCCNT49239.2020.9225324.

[13] A. B. Ibrahim and S. Kant, "Penetration testing using SQL injection to recognize the vulnerable point on web pages," *International Journal of Applied Engineering Research*, vol. 13, no. 8, pp. 5935–5942, 2018.

[14] S. Hajar, A. G. Jaafar, and F. A. Rahim, "A review of penetration testing process for SQL injection attack," *Open International Journal of Informatics*, vol. 12, no. 1, pp. 72–87, 2024, doi: 10.11113/oiji2023.11n2.256.

[15] S. Vyamajala, T. K. Mohd, and A. Javaid, "A real-world implementation of SQL injection attack using open source tools for enhanced cybersecurity learning," in *Proceedings of the 2018 IEEE International Conference on Electro/Information Technology (EIT)*, Rochester, MI, USA, May 2018, pp. 198–202, doi: 10.1109/EIT.2018.8500136.

[16] A. K. Priyanka and S. S. Smruthi, "Web application vulnerabilities: Exploitation and prevention," in *Proceedings of the 2020 Second International Conference on Inventive Research in Computing Applications (ICIRCA)*, Coimbatore, India, Jul. 2020, pp. 729–734, doi: 10.1109/ICIRCA48905.2020.9182928.

[17] A. Alzahrani, A. Alqazzaz, Y. Zhu, H. Fu, and N. Almashfi, “Web application security tools analysis,” in *2017 IEEE 3rd International Conference on Big Data Security on Cloud (Bigdatasecurity), IEEE International Conference on High Performance and Smart Computing (HPSC), and IEEE International Conference on Intelligent Data and Security (IDS)*, Beijing, China, 2017, pp. 237-242, doi: 10.1109/BigDataSecurity.2017.47.

[18] D. P. Mozumder, Md J. N. Mahi, and Md Whaiduzzaman, “Cloud computing security breaches and threats analysis,” *International Journal of Scientific & Engineering Research*, vol. 8, no. 1, pp. 1287–1297, 2017.

[19] M. R. Ul Islam, M. S. Islam, Z. Ahmed, A. Iqbal, and R. Shahriyar, “Automatic Detection of NoSQL Injection Using Supervised Learning,” in *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, Milwaukee, WI, USA, 2019, pp. 760-769, doi: 10.1109/COMPSAC.2019.00113.

[20] J. T.J. and K. S. Babu, “OWASP Zed Attack Proxy,” in *Proceedings of the National Conference on Emerging Computer Applications (NCECA)*, Kottayam, India, Jun. 2021, pp. 106-111.

[21] S. Putit and L. Y. B. Khedif, “Exploring Vega: A tool for scanning vulnerabilities in penetration testing within web applications,” *Borneo Akademika*, vol. 8, no. 2, pp. 176-187, 2024.

[22] E. Lawrence, *Debugging with Fiddler: The Complete Reference from the Creator of the Fiddler Web Debugger*, 2012.

[23] E. Crespo-Martínez, “Vulnerability analysis with sqlmap applied to APEX5 context,” *Ingenius*, no. 25, pp. 104–113, 2021.

[24] R. Umar, I. Riadi, and M. I. A. Elfatiha, “Security analysis of web-based academic information system using OWASP framework,” *Kinetik: Game Technology, Information System, Computer Network, Computer Electronics, and Control*, 2024, doi: 10.22219/kinetik.v9i4.2015.

[25] G. R. Fernandes and I. M. Lina, “Website penetration testing with SQL injection technique using SQLMAP on Termux,” *Jurnal E-Komtek (Elektro-Komputer-Teknik)*, vol. 8, no. 2, pp. 286–293, 2024, doi: 10.37339/e-komtek.v8i2.2074.

[26] R. Choudhary, J. Rawat, and G. Singh, “Comprehensive Exploration of Web Application Security Testing with Burp Suite Tools,” *International Journal for Multidisciplinary Research*, vol. 5, no. 6, Dec. 2023, doi: 10.36948/ijfmr.2023.v05i06.11297.

[27] Z. Savova, S. D. Atanasov, and R. Bogdanov, “Automated web application scanning with Wapiti, Selenium, and SQLMap,” *Security and Future*, vol. 8, no. 2, pp. 57–60, 2024.

BIOGRAPHIES OF AUTHORS

Mahmoud Baklizi 17 years of experience in academia in Jordan. Extensive theoretical and practical expertise in network security, network design, and implementation, with additional proficiency in security, vulnerability assessment, and programming. Several years of managerial experience in the academic sector in Jordan, with strong capabilities in strategic and action plan development. Interdisciplinary and intercultural experience in scientific research, applied research, and research and development within both academic and industrial fields in Jordan and Malaysia. Advanced proficiency in English. He can be contacted at email: mbaklizi@uop.edu.jo.

Mohammad Alkhazaleh received the degree in information technology and computing from Arab Open University (AOU), Jordan in 2009. He received the Master degree in computer science from The Middle East University for the Graduate Studies (MEU), Jordan in 2011. He received the PhD degree in computer engineering from University Malaysia Perlis (UniMAP), Malaysia in 2021. He was a lecturer at Faculty of Applied Studies and Continuing Education at Al-Baha University (2011-2016), and is currently an Assistant Professor in Department of Computer Sciences at the College of Information Technology at Isra University (2022-present). He can be contacted at email: m.alkhazaleh@iu.edu.jo.

Musab Bassam Yousef Alzghoul received his Bachelor of Science in computer science in Don state technical university (DSTU) Russia in 2005. In 2006, he received his Master degree in Computer Science at DSTU, and then in 2009, he received a PhD in Computer Science. He was a member of the Faculty of Information Technology at Zarqa University where he was a lecturer between 2009 and 2012. He was a lecturer in 2012-2023 at the Faculty of Information Technology of Um-AlQura University. He is now an Assistant Professor in the Department of Computer Sciences at the Isra University where he has been teaching since 2023. He can be contacted at email: musab.alzgool@iu.edu.jo.

Dr. Adi Maaita is the ICT Director at the King Hussein Cancer Center (KHCC) and formerly the Dean of Information Technology and an Associate Professor of Software Engineering at Middle East University (MEU) in Amman, Jordan. He holds a Ph.D. in Software Engineering from the University of Leicester, where his research focused on applying design patterns to embedded systems development. He has a broad academic and professional background, having previously served as a faculty member at Isra University and as a research assistant at the University of Leicester. His research interests include cyber security, artificial intelligence, and modern software development methodologies. He can be contacted at email: Am.18080@khcc.jo.

Jamal Zraqou is an Associate Professor at the Department of Computer Science/Virtual and Augmented Reality, University of Petra, Jordan, where he has been a faculty member since 2022. From 2018-2029, he was also the Dean faculty of IT at IU. His research interests include computer vision, virtual and augmented reality, IoT, cyber security, and image processing. He can be contacted at email: Jamal.Zraqou@uop.edu.jo.

Mohammad AlShaikh-Hasan is a researcher in the Department of Computer Science, Brunel University of London, London, UK. He can be contacted at email: Mohammad.AlShaikhHasan@brunel.ac.uk.