

Article

Development of an SQL injection detection and prevention model using genetic algorithm

Samson Daniel^{1,*}, Edim Azom Emmanuel¹, and Felix Ukpai Ogban¹

¹ Department of Computer Science, University of Calabar, Cross River State, Nigeria, edimemma@gmail.com (E.A.E), fogban@gmail.com (F.U.O.)

* Correspondence: samsondaniel135@gmail.com (S.A.)

Abstract: SQL injection is a common and dangerous attack vector in web applications that allows attackers to execute malicious SQL queries to gain unauthorized access to the database. We aim to develop a more adaptive and resilient system that can dynamically evolve and adapt to new attack patterns. SQL injection detection and prevention has the potential to significantly improve the security of web applications and provide better protection against SQL injection attacks. Intrusion detection and prevention systems (IDPS) play a critical role in safeguarding computer networks from malicious activities and security breaches. Traditional IDPS solutions often struggle to adapt to evolving threats and exhibit limitations in accurately detecting and preventing sophisticated attacks. This approach is for enhancing IDPS capabilities through the integration of a hybrid genetic algorithm (HGA). By combining the evolutionary search capabilities of genetic algorithms with the domain-specific knowledge and rules of intrusion detection systems, the proposed HGA offers a robust framework for improving detection accuracy and reducing false positives. The hybridization process involves incorporating genetic operators, such as crossover and mutation, into the rule-based detection mechanisms of IDPS. Additionally, the HGA dynamically adjusts detection thresholds and parameters based on real-time network traffic analysis, enabling adaptive and proactive defense mechanisms against emerging threats.

Keywords: SQL Injection detection, prevention model, genetic algorithm, web application, hacking, vulnerability, intrusion detection system.

Received: 24 October 2025; Revised: 22 Nov. 2025; Accepted: 21 December 2025; Published: 29 January 2026



Copyright: ©2026 the Author(s). Published by JSSCI. This is an open-access article distributed under the terms of the Creative Commons Attribution 4.0 International License (CC BY 4.0).

Journal Abbreviation: J. Stat. Sci. Comput. Intell.

1. Introduction

SQL Injection Attacks (SQLIAs) stand out as one of the most prevalent threats to the security of database-driven applications, posing significant risks to the confidentiality and integrity of information. SQLIAs, a type of code injection attack, exploit the absence of user input validation, allowing attackers to manipulate their

illegitimate input as components of the final query string processed by databases. This vulnerability particularly puts financial web applications or secret information systems at risk, jeopardizing their authority, integrity, and confidentiality. Despite developers implementing defensive coding practices to address this vulnerability, their efforts have proven insufficient. SQL Injection attacks are unfortunately widespread due to the combination of prevalent vulnerabilities and the attractiveness of the target databases containing critical data for applications [1, 2, 3, 4].

SQL injection attacks are a major threat to web applications, making up more than 70% of security breaches according to a [5] study [OWASP]. In this attack, hackers sneak malicious code into a website's forms or hidden areas. This code can then trick the database behind the website into revealing sensitive information or even taking actions the attacker wants. If a website has this vulnerability, attackers can potentially steal data, corrupt information, or even control parts of the system.

In the current data-driven economy, the value of data is paramount, underscoring the critical need for its protection. While the impact of a successful SQL injection attack varies depending on the targeted application and how it processes user-supplied data, the threat of SQL injection attacks persists for any organization.

2. Literature Review

Hackers often target weaknesses in how web applications communicate (application layer vulnerabilities). These can be tricky to defend against. Luckily, there are tools and techniques to scan websites for these weaknesses. Regularly checking your website for vulnerabilities (like a daily scan) is a great way to stay ahead of attackers. This proactive approach lets you fix any problems before they can be exploited. For completely new vulnerabilities (zero-day attacks), there's no pre-made fix (patch). However, even in these cases, it's possible to identify where the attack originates or where compromised users are being sent. This information can help stop the attack from spreading further [6].

Developing a tool named SecuBat, utilizing the black box test approach. This vulnerability scanning tool automatically examines web applications through both generic and specific analyses to uncover SQL injection and XSS vulnerabilities. SecuBat employs multi-threaded crawling, attack, and analysis components, supported by a graphical user interface, to scan web pages for exploitable security flaws. Therefore, it is crucial to design and evaluate vulnerability scanners meticulously to prevent false positives [7].

A key challenge in security testing is the difficulty of definitively proving an application is secure. While finding vulnerabilities can be straightforward, guaranteeing their absence is nearly impossible. Both penetration testing and static code analysis have limitations. Penetration testing relies on successfully running code, but often only examines the application's final output. This limited visibility into internal workings hinders its effectiveness. On the other hand, comprehensively analyzing source code can be difficult due to complexity and the lack of a dynamic view during execution. Additionally, static code analysis requires access to the source code, unlike penetration testing. Choosing the wrong security tool can lead to deploying applications with undiscovered vulnerabilities [8, 9].

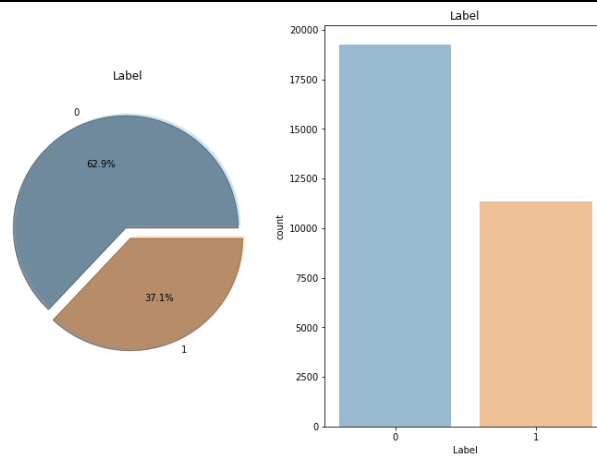


Figure 1. Evaluation on SQL detection and prevention system based on genetic algorithm

2.1. Theoretical Framework

Burn and Gove (2017) define research design as the blueprint for a study, guiding researchers to make sound decisions, we adopted an incremental development approach for this project due to its flexibility in development, debugging, testing, and validation. As described by [10], this method involves building an initial version and then iteratively refining it through multiple releases until an optimal version is achieved. The incremental model (framework) that was followed in this work is presented diagrammatically. The GA acts as a machine learning component, improving the tool's performance during vulnerability scans. The scanner prioritizes ease of use with minimal dependencies, making it suitable for single-machine deployments [11]. Our tool utilizes GA as its core machine learning technique. Initiating a scan triggers the GA module, responsible for intelligently grouping data based on similarities. Subsequently, a random sample from each group is chosen and fed to the analysis module for vulnerability detection (focusing on SQL injection and XSS). [12] The results obtained from this sample are then generalized to represent the entire group, assuming a degree of similarity among the grouped objects/scripts. This approach significantly reduces scan time compared to brute-force methods, where every script is individually analyzed. We target web-based applications and report vulnerabilities based on HTTP status code analysis [4, 5, 9].

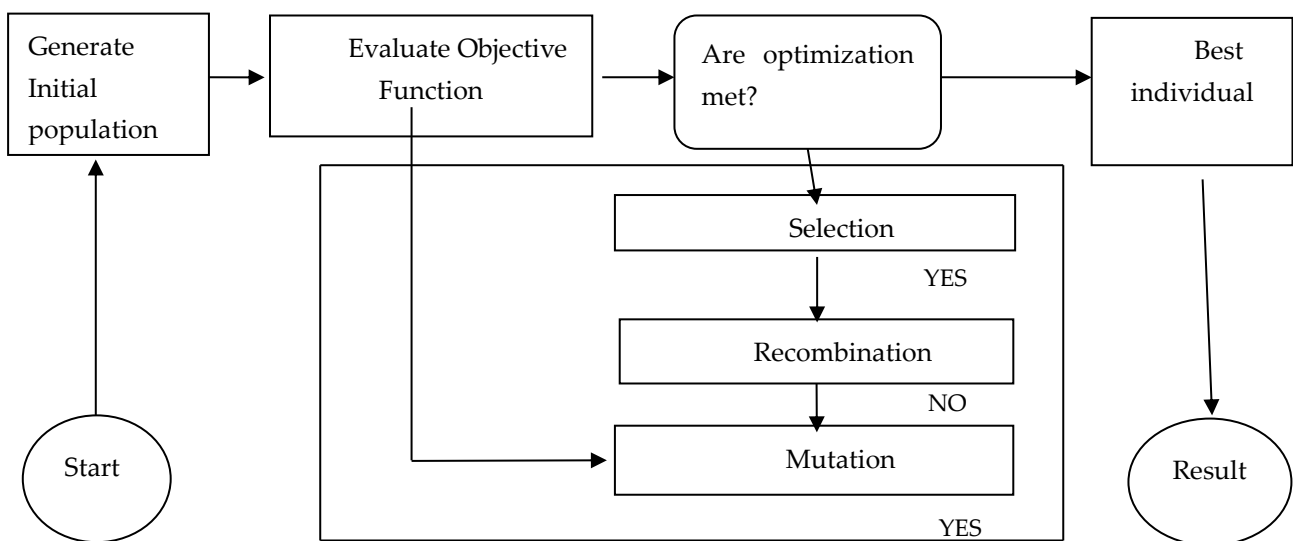


Figure 2. Structure of a simple genetic algorithm. Source: (Pohlheim, 2021)

3. Research Methodology

Burn and Gove (2017) define research design as the blueprint for a study, guiding researchers to make sound decisions and avoid issues that could invalidate results. Following this approach, this research employed systematic sub-headings.

We adopted an incremental development approach for this project due to its flexibility in development, debugging, testing, and validation. As described by Sommerville (2019), this method involves building an initial version and then iteratively refining it through multiple releases until an optimal version is achieved to enable detect and prevent attack from hackers accurately.

3.1. Research Design

The design of our vulnerability scanner considers real-world use cases, desired features, and the technology needed to bring it to life. We leverage Unified Modeling Language (UML) models to visually represent the system's components and interactions (discussed in following sections). For complex systems like ours, traditional optimization methods can fall short.

3.2. Software Development Model

The design of our vulnerability scanner considers real-world use cases, desired features, and the technology needed to bring it to life. We leverage Unified Modeling Language (UML) models to visually represent the system's components and interactions. For SQL complex systems like ours, traditional optimization methods can fall short. Here's where genetic algorithms (GA) comes in. Inspired by natural selection, GA mimics evolution to find the best solution. It starts with a pool of potential designs (individuals) and applies genetic operators like crossover, mutation, and selection to create new generations; built-in network libraries streamline network connection management.

One Point Crossover (Figure 3)

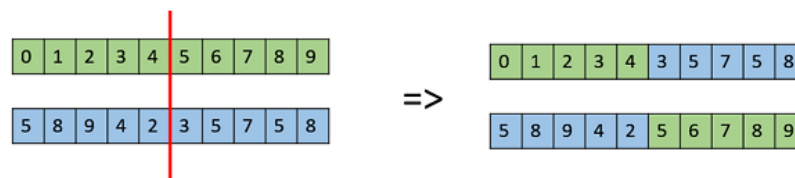


Figure 3. Structure of a One Point Crossover

Source: (Pohlheim, 2021)

Multi Point Crossover (Figure 4)

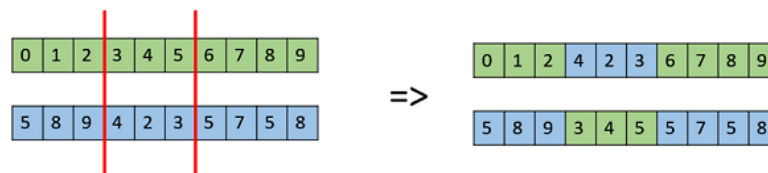


Figure 4. A Multi Point Crossover

Uniform Crossover (Figure 5)

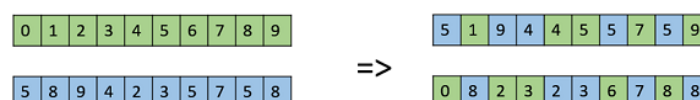


Figure 5. Structure of a Uniform Crossover

3.2.1. System Development requirement

Operating System: Windows 11 or later, Software: Java Runtime Environment (JRE) 1.8 or higher, Memory: At least 500MB RAM, Storage: 120MB free hard disk space, Connectivity: Network card or Wi-Fi adapter, internet availability

The software focuses on detecting two common vulnerabilities: SQL Injection: This exploit tricks the web application's database into revealing sensitive information or even modifying data. Cross-Site Scripting (XSS): This vulnerability allows attackers to inject malicious scripts into web pages, potentially harming unsuspecting users. SQL injection, the software scans for SQL injections where SQL commands are injected into the form component to check if an attacker can also effectively insert code and modify SQL commands.

4. Conclusion

Our exploration of genetic algorithms for SQL injection detection shows promising results. By mimicking natural selection, the system continuously refines its detection rules, making it more adept at identifying and blocking these attacks. This approach offers two key advantages:

4.1. Adaptability

The system can learn and adjust to new threats, staying ahead of evolving attack methods.

4.2. Robustness

It becomes increasingly resilient against ever-changing threat landscapes.

Genetic algorithms provide a dynamic and efficient way to improve the system's accuracy over time. This translates to a significant boost in security for database-driven applications, safeguarding them against the constant threat of SQL injection vulnerabilities.

4.3. Recommendations

Continuous Monitoring and Updates: Regularly review emerging attack patterns and update the system accordingly. This proactive approach keeps your defenses effective against evolving threats.

Adaptive Learning: Implement an adaptive learning mechanism. By analyzing real-time data, the system can dynamically adjust its detection rules to respond to new and sophisticated SQL injection techniques.

4.4. Collaborative Defense

Explore incorporating threat intelligence sharing with other security systems. This is collaborative approach strengthens your overall security posture by leveraging insights from a wider network.

4.5. User-Friendly Configuration

Provide a user-friendly interface for administrators to adjust settings, thresholds, and rules. This allows for easy customization based on specific application needs and environments.

Author contributions: All authors have equal contributions.

Funding Support: This work is sponsored among the authors.

Acknowledgement: Grateful to supervisor Prof. Edim Azom Emmanuel for your unfailing attention and guidance

for the assistance in data analysis.

Data Availability Statement: The data that support the findings of this study are openly available in kaggle.com “<https://www.kaggle.com/datasets/sajid576/sql-injection-dataset>”.

Ethical Statement: This study does not contain any studies with human or animal subjects performed by any of the authors.

Conflicts of Interest: The authors declare that they have no conflicts of interest to this work.

References

- [1] Delwar, C. & Jain, S.C., (2019). "Analysis and classification of SQL injection vulnerabilities and attacks on web applications," *International Conference on Advances in Engineering and Technology Research (ICAETR)*, pp.1- 6.
- [2] Han, S.; Xie, M.; & S. Kumar (2021). SQL injection:Types, methodology, attack queries and prevention, *3rd Int.Conf. Computer. Sustainable Global Dev.(INDIACom)* pp. 2872–2876.
- [3] Slatalla. D. & Himanshu. G. (2020). SQL Filtering: An Effective Technique to prevent SQL Injection Attack, in *International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*, pp. 312 – 317.
- [4] Bing .Z., & Chitsutha. S. (2021). "Machine Learning for SQL Injection Prevention on Server- Side Scripting", in *International Computer Science and Engineering Conference (ICSEC)*,pp. 1-6.
- [5] Son, S.; McKinley, K.S.; Shmatikov, V. Diglossia: Detecting code injection attacks with precision and efficiency. *Proc. ACM Conf. Comput. Commun. Secur.* 2018, 2, 1181–1191. [Google Scholar] [CrossRef]
- [6] Devakunchari R. & Valliyammai C. (2022). A top web security vulnerability SQL injection attack", *Seventh International Conference on Advanced Computing (ICoAC)*.
- [7] Delwar.D, (2018). "Advanced Automated SQL Injection Attacks and Defensive Mechanisms", in *Annual Connecticut Conference on Industrial Electronics,Technology & Automation (CT-IETA)*,pp. 1-6.
- [8] Fonseca.F, & Tarique M., (2019). Detection of SQL injection attacks: A machine learning approach, *Int. Conf. Electr. Comput. Technol. Appl. (ICECTA)* pp. 1- 6.
- [9] Schneider. K, (2021). Based approach for detection of injection attacks, *Proc. 2nd IEEE Int. Conf. . Intelligent Knowledge. Econ. ICCIKE*, pp. 378-383.
- [10] Jovanovic J., & Yukovetskyi O. S. (2021). "SQL Injection Prevention System", *IEEE International Conference Radio Electronics & Communications*.
- [11] Halfond H., & ThosarS., (2016). "Detection of SQL injection and XSS attacks in three tier web applications",*International Conference on Computing Communication Control and automation(IC-CUBEA)*.
- [12] Huang H,& Srinivas A. (2023). "An Application Specific Randomized Encryption Algorithm to Prevent SQL Injection", *International Conference on Trust, Security and Privacy in Computing*



Disclaimer/Publisher's Note: The views, opinions, and content expressed in all articles are solely those of the respective author(s) and contributor(s) and do not necessarily reflect those of the JSSCI, its editors, or the publisher. JSSCI and its editorial team assume no responsibility for any harm or damage resulting from the use of information, methods, or products mentioned in the publication