Global Scientific JOURNALS

GSJ: Volume 11, Issue 7, July 2023, Online: ISSN 2320-9186
www.globalscientificjournal.com

AUTOMATED INJECTION VULNERABILITIES DETECTION IN WEB SERVICES USING GENETIC ALGORITHM

Muhammad Saddam Hussain Shah

Department of Computer Science & Information Technology, University of Engineering & Technology Peshawar, Pakistan
E-mail: saddamshah4u@gmail.com

ABSTRACT

Web services are widely used in online applications today, offering a means for applications to interact with online services provided by servers. However, their popularity also makes them vulnerable to attacks, with injection attacks being the most common and concerning threat. Injected malicious code can alter the expected behavior of web services, potentially exposing sensitive information. Existing tools for testing web service security have limitations, either testing known vulnerabilities or requiring manual testing, which is time-consuming and resource-intensive. In this research, we conduct a comprehensive literature review of testing methodologies, analyze existing tools, and propose a proof of concept for testing web services using Genetic Algorithms (GA) for test data generation. The proposed automated testing tool is evaluated on different web services and test case scenarios to determine its success rate.

Keywords: Automated Testing, Web Services, Injection Vulnerabilities, Genetic Algorithm, SQL Injection, Security Testing, Test Data Generation

INTRODUCTION

Over the past decade, internet services in various domains have gained immense popularity, with people worldwide relying on web services for diverse purposes in sectors like health, banking, shopping, education, social networking, and government. These services, such as social networks, online shopping platforms, and online banking applications, offer convenience and accessibility to users. However, the increasing popularity of web services also attracts malicious intent, leading to a growing threat of attacks on their security.

As web services are extensively data-driven and rely on databases for storing information, they become vulnerable to exploitation through injection attacks. Attackers exploit input fields to manipulate the underlying database queries, posing risks such as exposing sensitive data, data modification, service unavailability, privilege escalation, authentication bypass, and data destruction. Therefore, it becomes crucial to implement robust security measures to safeguard web services from these threats.

Despite researchers and security engineers adopting updated security mechanisms, attackers continue to evolve their strategies to exploit vulnerabilities in web services. One prevalent vulnerability is SQL injection, consistently ranking among the top vulnerabilities according to the Open Web Application Security Project (OWASP) over the years [1].

In injection attacks, malicious code is injected through user interface input fields, leading the web service to behave abnormally or unexpectedly. The injected code aims to cause damage to data or expose sensitive information, making web services susceptible to exploitation.

BACKGROUND

In recent years, internet services across various domains have witnessed an unprecedented surge in popularity, becoming an integral part of people's lives globally. Web services have transformed the way individuals engage with online platforms, encompassing diverse sectors such as health, finance, e-commerce, education, social networking, and government services. These web services offer convenience, efficiency, and accessibility, making them an indispensable component of the modern digital landscape.

As the reliance on web services continues to grow, so does the concern over their security. The widespread adoption of these services has made them an attractive target for malicious actors seeking to exploit vulnerabilities for nefarious purposes. One of the most common and widely-used attack methods against web services is injection attacks.

Injection Vulnerabilities

Injection vulnerabilities pose a significant threat to web services, particularly those that rely on data-driven operations with underlying databases. Attackers exploit these vulnerabilities by injecting malicious code into input fields provided by the user interface. By manipulating the queries executed on the database, the attackers can potentially gain unauthorized access, modify data, disrupt services, escalate privileges, bypass authentication mechanisms, or even cause irreversible data loss.

Injection Vulnerabilities Types

Among various injection vulnerability types, SQL injection remains a major concern. Ranked among the top vulnerabilities by the Open Web Application Security Project (OWASP) [1] for an extended period, SQL injection attacks target database-driven systems. These attacks can have severe consequences and are frequently

exploited by adversaries aiming to compromise web services.

Genetic Algorithm

Genetic Algorithms (GAs) offer a powerful approach to address security testing challenges, particularly concerning the detection and mitigation of injection vulnerabilities in web services. GAs are a subset of evolutionary algorithms that draw inspiration from the process of natural selection. They iteratively evolve and refine solutions to problems through selection, crossover, and mutation, mimicking the principles of genetic inheritance and survival of the fittest.

The application of Genetic Algorithms in security testing involves the automated generation and evaluation of test cases, enabling the systematic exploration of the vast input space of web services. By optimizing the testing process, GAs can efficiently identify potential vulnerabilities, including undiscovered ones, and enhance the overall security posture of web services.

Genetic Algorithm Steps

Genetic Algorithms (GAs) employ an evolutionary approach to problem-solving, inspired by the principles of natural selection and genetics. The process involves iteratively evolving and improving potential solutions to a given problem. Here are the key steps involved in a Genetic Algorithm:

1. Initialization: The process begins by creating an initial population of potential solutions, known as individuals or chromosomes. Each individual represents a possible solution to the problem at hand. The population is typically generated randomly or through some heuristic method.

2. Fitness Evaluation: Each individual in the population is evaluated based on a fitness function, which quantifies how well the individual solves the problem. The fitness function provides a measure of the solution's performance or suitability for the given task.

3. Selection: In this step, individuals are selected from the population to create a new generation of solutions. The probability of selection is usually proportional to an individual's fitness; better-performing individuals have a higher chance of being selected.

4. Recombination (Crossover): Selected individuals undergo recombination or crossover, which involves combining genetic information from two or more parents to create new offspring. The process mimics genetic recombination found in nature, promoting exploration of different solution spaces.

5. Mutation: After crossover, some individuals in the new generation may undergo mutation. Mutation introduces small random changes to an individual's genetic information. This step helps introduce new variations and prevents premature convergence to suboptimal solutions.

6. Replacement: The new generation, comprising offspring from recombination and potentially mutated individuals, replaces the previous population. The replacement strategy may vary, such as elitism (preserving the best individuals) or generational replacement (replacing the entire population).

7. Termination: The algorithm continues to iterate through the selection, crossover, and mutation steps for a predefined number of generations or until a stopping criterion is met. The termination condition could be a satisfactory solution, a maximum number of iterations, or a time limit.

8. Solution Extraction: Once the algorithm terminates, the best individual or solution, determined by the fitness

function, is extracted from the final population. This solution represents the optimal or near-optimal solution to the problem.

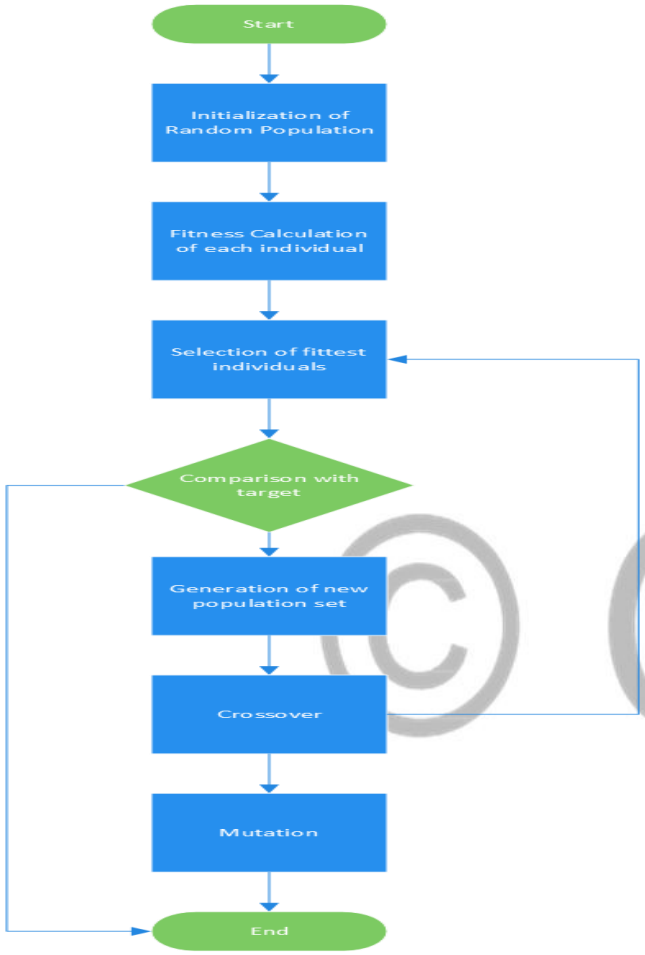

Figure 1: Genetic Algorithm Flow Chart

LITERATURE REVIEW

Web service security literature extensively addresses injection vulnerabilities, with a particular focus on SQL injection attacks, exploring diverse detection and prevention approaches. Web services' widespread adoption across domains like health, finance, e-commerce, and social networking has made them a prime target for malicious actors seeking to exploit vulnerabilities.

One study [2] assesses the impact of firewalls and database proxies on SQL injection testing. Though these mechanisms offer value in a comprehensive defense strategy, they may pose challenges during thorough testing. Researchers conducted experiments in a controlled testbed to analyze their effectiveness in detecting and preventing SQL injection vulnerabilities.

In [3], an innovative automated testing method for SQL injection vulnerabilities is introduced, using input mutation techniques with mutation operators. The approach covers a wide range of test scenarios, including legitimate and malicious inputs, enhancing testing comprehensiveness and efficiency.

SOFIA, an automated security oracle for black-box SQL injection testing [4], utilizes dynamic analysis and machine learning to detect injection points and vulnerabilities in responses. The architecture includes an injection detector and a security oracle, trained using machine learning techniques.

Another study [5] evaluates web security mechanisms through vulnerability and attack injection, assessing their ability to withstand real-world attacks. Experiments inject various vulnerabilities and attacks into web applications, analyzing detection, prevention, and intrusion detection system accuracy.

In [6], researchers propose a novel approach using Gene Expression Programming (GEP) for detecting web application attacks. GEP evolves detection rules using training datasets, demonstrating the effectiveness of evolutionary computation techniques.

To counter SQL injection vulnerabilities, multi-layered defense systems are proposed [7], combining detection and prevention techniques. The approach analyzes input parameters and query structures while employing input

sanitization and prepared statements to prevent SQL injection.

Researchers propose cross-platform intrusion detection systems with inter-server communication [8], enhancing detection capabilities by sharing real-time information among interconnected servers.

The combination of k-means, fuzzy neural networks, and SVM classifiers [9] improves intrusion detection accuracy. The approach clusters network traffic data and classifies activities in real-time, enhancing detection effectiveness.

A poster presentation [10] showcases a multi-source data analysis approach for SQL injection detection, combining machine learning techniques with HTTP, application, and database logs to achieve improved accuracy.

Studies [11][12] introduce runtime monitors for detecting tautology-based and union query-based SQL injection attacks, enhancing real-time web application security.

Additionally, the literature provides an overview of genetic algorithms (GA) [13], showcasing their efficacy in solving complex optimization problems across various applications.

A heuristic-based approach [14] complements existing techniques, identifying suspicious patterns and behaviors indicative of SQL injection attacks to enhance vulnerability detection.

Furthermore, a comparative study [15] analyzes white box, black box, and grey box testing techniques, offering insights into their strengths and limitations for different scenarios.

RESEARCH METHODOLOGY

The research methodology employed in this study is designed to comprehensively address the security testing of web services for injection vulnerabilities using a genetic algorithm. The aim is to evaluate existing tools, design a novel objective tool, and conduct experiments to perform a comparative analysis. This systematic approach ensures the credibility of the research and provides valuable insights into the effectiveness of various vulnerability detection techniques.

1. Literature Review: In the initial phase, a comprehensive literature review was conducted to gather existing research knowledge. This step provided a solid foundation for identifying relevant tools and designing the required tool to effectively address the research problem.

2. Evaluating Existing Tools: Following the literature review, a practical analysis was conducted to assess the performance and effectiveness of existing tools in detecting vulnerabilities. User experience data was collected through experiments to establish benchmarks for the development of the objective tool.

3. Curating and Choosing Test Subjects for Assessment: To ensure a comprehensive evaluation, a selection of prototype web services was created as test subjects. Additionally, widely-used web services and applications were included to enhance the representativeness of the evaluation.

4. Developing, and Testing the Proposed Tool: During this stage, the objective tool was designed, developed, and thoroughly tested to address the research problem. Rigorous testing ensured its functionality and efficacy.

5. Performing Experiments and Comparative Evaluation: In this step, experiments were executed on the selected test subjects using each tool, and the results were analyzed comparatively. This process validated the proposed approach and the objective tool while assessing the efficiency of existing vulnerability detection tools.

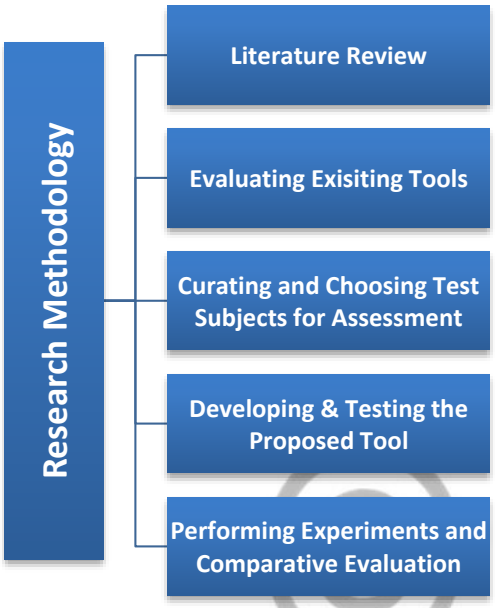

Figure 2: Research Methodology Steps

RESULTS

In this study, we applied the genetic algorithm (GA) for generating malicious test data to evaluate web security. The prototype tool outperformed existing tools, detecting 76 unique pattern payloads not flagged by others. We achieved successful test objectives for single and multi-variable targets with and without validation mechanisms. Real-world application testing showed the GA's efficiency in detecting vulnerabilities. It's important to use GA alongside best practices like input validation and secure coding for comprehensive web security.

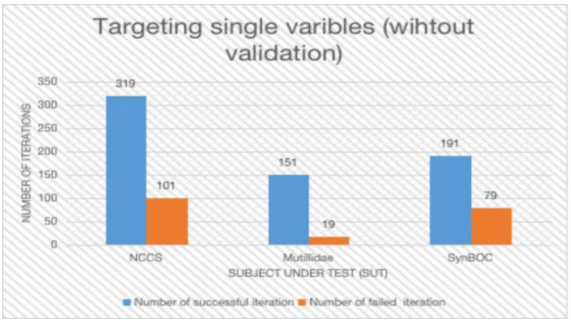

Figure 3: Single variable targeting without validation

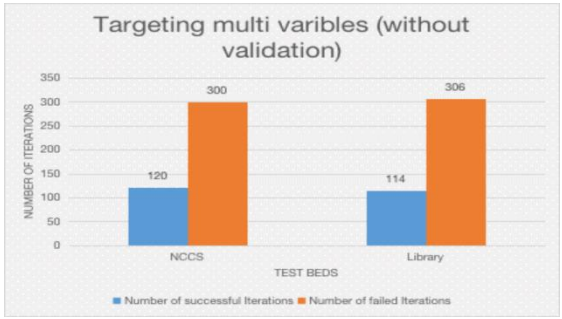

Figure 4: Multi-variable targeting without validation

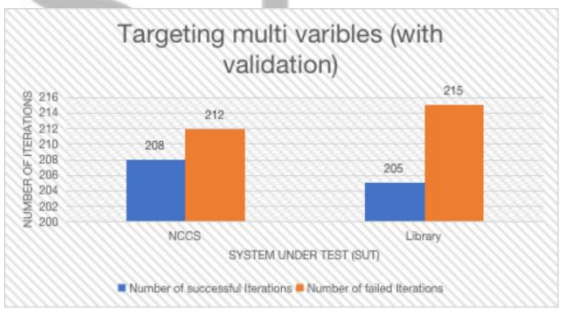

Figure 5: Multi-variable targeting with validation

CONCLUSION

In this research, we investigated the use of genetic algorithm (GA) for vulnerability detection and web security testing, which offers significant advantages over traditional brute force or random test generation techniques. Inspired by biological evolution, GA efficiently targets vulnerabilities, making it a valuable tool for security researchers and penetration testers. The study successfully demonstrated the proof of concept of employing genetic

algorithms in automated web security testing through a comparative analysis of the results generated by our proposed approach.

However, it is essential to acknowledge that while genetic algorithms can aid in detecting SQL injection vulnerabilities, they should not be relied upon as the sole defense mechanism. Best practices such as input validation, parameterized queries, and secure coding practices must also be implemented to effectively mitigate the risk of SQL injection attacks. The combination of these approaches ensures a robust and comprehensive web security strategy.

REFERENCES

[1] OWASP Top Ten Web Application Security Risks | OWASP, 2020. OWASP Top Ten page available online at https://owasp.org/www-project-top-ten/ (Access Date 17-05-2020).

[2] Appelt, D., Alshahwan, N. and Briand, L., 2013. Assessing the impact of firewalls and database proxies on sql injection testing, International Workshop on Future Internet Testing, pp. 32–47.

[3] Appelt, D., Nguyen, C.D., Briand, L.C. and Alshahwan, N., 2014. Automated testing for SQL injection vulnerabilities: an input mutation approach, Proceedings of the 2014 International Symposium on Software Testing and Analysis, pp. 259–269.

[4] Ceccato, M., Nguyen, C.D., Appelt, D. and Briand, L.C., 2016. SOFIA: An automated security oracle for black-box testing of SQL-injection vulnerabilities, 2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 167–177.

[5] Fonseca, J., Vieira, M. and Madeira, H., 2013. Evaluation of web security mechanisms using vulnerability & attack injection. IEEE Transactions on dependable and secure computing, 11: 440–453.

[6] Skaruz, J. and Seredynski, F., 2009. Detecting web application attacks with use of Gene Expression Programming, 2009 IEEE Congress on Evolutionary Computation, pp. 2029–2035.

[7] Hlaing, Z.C.S.S. and Khaing, M., 2020. A Detection and Prevention Technique on SQL Injection Attacks, 2020 IEEE Conference on Computer Applications (ICCA), pp. 1–6.

[8] Priyadarshini, R., Jagadiswaree, D., Fareedha, A. and Janarthanan, M., 2011. A cross platform intrusion detection system using inter server communication technique, 2011 International Conference on Recent Trends in Information Technology (ICRTIT), pp. 1259–1264.

[9] Chandrasekhar, A.M. and Raghuveer, K., 2013. Intrusion detection technique by using k-means, fuzzy neural network and SVM classifiers, 2013 International Conference on Computer Communication and Informatics, pp. 1–7.

[10] Ross, K., Moh, M., Moh, T.-S. and Yao, J., 2017. Poster: Multi-source data analysis for SQL injection detection, 38th IEEE Symposium on Security and Privacy (IEEE S&P), San Jo-se, CA.

[11] Dharam, R. and Shiva, S.G., 2012. Runtime monitors for tautology based SQL injection attacks, Proceedings Title: 2012 International Conference on Cyber Security, Cyber Warfare and Digital Forensic (CyberSec), pp. 253–258.

[12] Dharam, R. and Shiva, S.G., 2013. Runtime monitors to detect and prevent union query based SQL injection attacks, 2013 10th International Conference on Information Technology: New Generations, pp. 357–362.

[13] Deb, K., 1998. Genetic algorithm in search and optimization: the technique and applications, Proceedings of international workshop on soft computing and intelligent systems, pp. 58–87.

[14] Angelo Ciampa, C. A. (2010). A heuristic-based approach for detecting SQL-injection vulnerabilities in web applications. SESS '10: Proceedings of the 2010 ICSE Workshop on Software Engineering for Secure Systems, 43-49.

[15] Mohd Ehmer Khan, F. K. (2012). A Comparative Study of White Box, Black Box and Grey Box Testing Techniques. International Journal of Advanced Computer Science and Applications(IJACSA), 3(6).