Detecting SQL Injection vulnerabilities using nature-inspired algorithms

Kevin Baptista[1], Anabela Moreira Bernardino[1[0000-0002-6561-5730]] and Eugénia Moreira Bernardino[1[0000-0001-5301-5853]]

[1] Computer Science and Communication Research Center (CIIC), School of Technology and Management, Polytechnic of Leiria, 2411-901, Leiria, Portugal
2190371@my.ipleiria.pt, anabela.bernardino@ipleiria.pt, eugenia.bernardino@ipleiria.pt

Abstract. In the past years, the number of users of web applications has increased, and also the number of critical vulnerabilities in these web applications. Web application security implies building websites to function as expected, even when they are under attack. SQL Injection is a web vulnerability caused by mistakes made by programmers, that allows an attacker to interfere with the queries that an application makes to its database. In many cases, an attacker can see, modify or delete data without proper authorization. In this paper, we propose an approach to detect SQL injection vulnerabilities in the source code, using nature-based algorithms: Genetic Algorithms (GA), Artificial Bee Colony (ABC), and Ant Colony Optimization (ACO). To test this approach empirically, we used web applications purposefully vulnerable as Bricks, bWAPP, and Twitterlike. We also perform comparisons with other tools from the literature. The simulation results verify the effectiveness and robustness of the proposed approach.

Keywords: SQL Injection, Nature-inspired Algorithms, Genetic Algorithms, Swarm Optimization algorithms.

1 Introduction

The Open Web Application Security (OWASP) Top 10 is a standard awareness document for developers and web application security that lists the top 10 web application security risks for 2021 [1]. The focus of this paper is on SQL Injection.

OWASP defines SQL Injection as a type of injection attack that occurs when untrusted data is sent to an application as part of a query [1]. The main goal for the attacker is to trick the interpreter into executing unintended queries to execute unauthorized actions like obtaining unauthorized data.

In the last years, several combinatorial optimization problems have arisen in the communication networks field. In many cases, to solve these problems it is necessary the use of emergent optimization algorithms [2]. We developed an automated tool, based on the application of nature-based algorithms that tries to find the greatest number of vulnerabilities in the shortest time possible. We implemented Genetic Algorithm (GA), Artificial Bee Colony (ABC), and Ant Colony Optimization (ACO). The main

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08760-8_38

purpose of this tool is to be used in a white box scenario, having access to the code base. Thus, it could help developers to find potential vulnerabilities in their codebase. To empirically evaluate our approach, we used several open-source PHP projects that are known to have certain vulnerabilities, such as Bricks, bWAPP, and Twitterlike.

We compare the results obtained by our approach with manual analysis, the results obtained in previous works by the same authors [3, 4], and with the tools Web Application Protection (WAP) and SonarPHP that use a static analysis approach to detect vulnerabilities in PHP web applications.

The paper is organized as follows. Section 2 presents some related works. Section 3 describes the problem representation. Section 4 describes the proposed algorithms and, Section 5 discusses the experiments conducted and the results obtained. Section 6 presents the conclusions and some directions for future work.

2 Related Work

An intensive study was done before developing the approach presented in this document. Since this article has a limited number of pages, we only highlight the most important works in the area.

Mckinnel et al. [5] made a comparative study of several Artificial Intelligence algorithms in exploiting vulnerabilities. They compiled several works in the area to compare several algorithms, including unsupervised algorithms, reinforcement Learning, GA, among others. The authors conclude that GA performs better over time due to the evolutionary nature of generations. They suggest that its applicability needs to take into account a good definition of the fitness function in order to obtain better results.

Manual penetration tests, although effective, can hardly meet all security requirements that are constantly changing and evolving [6]. Furthermore, they require specialized knowledge which, in addition to presenting a high cost, is typically slower. The alternative is to use automated tools that, although faster, often do not adapt to the context and uniqueness of each application. In [6], the author developed a reinforcement learning strategy capable of compromising a system faster than a brute force and random approach. He concluded that it is possible to build an agent capable of learning and evolving over time so that it can penetrate a network. Its effectiveness was equal to human capacity. Finally, he concludes that although the initial objective was long, there are still several directions to be explored, from the use of different algorithms for both an offensive (red team) and a defensive (blue team) security perspective. It suggests the application of game theory concepts [7], especially treating a problem like a Stackelber Security Game. These techniques have been successfully applied in various security domains such as finding the optimal allocation for airport security given the attackers' knowledge.

Alenezi and Javed [8] analyzed several open-source projects in order to identify vulnerabilities. They concluded that most of these errors are due to negligence on the part of developers as well as the use of bad practices. The authors suggest the development of a framework that encourages programmers to follow good practices and detect possible flaws in the code.

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08760-8_38

In [9], the authors propose a solution to detect XSS (cross-site scripting) vulnerabilities based on the use of GAs as well as a proposal to remove the vulnerabilities found during the detection phase. The aim, therefore, was to find as many vulnerabilities as possible with as few tests as possible. The results obtained with GAs were compared with other static analysis tools.

In [3] and [4] the same authors of this paper proposed an approach to detect SQL injection vulnerabilities in the source code, using GA [3] and swarm-based algorithms [4]. In these works were used different representations of the individuals. We also compare our results with these results.

3 Problem Representation

In order to use a nature-inspired algorithm as an optimization algorithm to find SQL Injection vulnerabilities, the process was divided into two steps: (1) identification of all SQL queries; and (2) use of GA, ABC, or ACO to generate attack vectors to be injected in the queries.

In order to obtain all non-parametrized queries, first, it is necessary to perform a search to list all PHP files recursively in a given folder. Afterward, all variables in the code are indexed and their history is kept. This step is crucial to capture SQL queries that are parametrized, but still vulnerable because the vulnerabilities occurred before in the code. These queries and all non-parametrized queries are kept in a list to be used in the second phase by the algorithm.

The main goal of the second phase is to find a vector that could compromise one of the queries listed in the previous step. So, the algorithm starts by initializing all the needed parameters, then it either goes through the GA, ABC, or ACO. Each one has its specific steps and parameters, but the fitness calculation is common to all of them. More information about these steps can be found in [3, 4].

Here, each individual is made up of a set of five genes. Each gene is a String (values are derived from SQL injection database which was constructed based on resources [10] and [11]). In Fig. 1, there is a possible representation for the individual, where each gene is an attack vector.

Fig. 1. Example of an individual.

Following this approach, an individual can generate up to *N* different attack vectors, where *N* is the number of genes in the individual. individuals (bees in ABC and ants in ACO). As represented in Fig. 1, each gene in the individual is going to be tested as an attack vector.

The queries executed for this scenario are illustrated in Table 1 (an individual with 5 genes executes 5 queries). In this example, only two are vulnerable.

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08760-8_38

Table 1. Executed queries.

Query	Vulnerable
SELECT * **FROM** users **WHERE** ua='--1=1'	No
SELECT * **FROM** users **WHERE** ua='x' or 1=1; --'	Yes
SELECT * **FROM** users **WHERE** ua='') OR 1=1--'	No
SELECT * **FROM** users **WHERE** ua='\";DROP TABLE users; --'	Yes
SELECT * **FROM** users **WHERE** ua='or 1=1--'	No

A fitness value is used to evaluate the performance of an individual. An individual with bigger fitness means that it is able to crack successfully more queries. The fitness function used for this problem is as follows:

$$fitness(i) = \frac{U_{vul}*5+G_{vul}*2}{totalGenes} \quad (1)$$

where i is the individual being tested, $totalGenes$ is the total number of genes in an individual, U_{vul} is the number of unique vulnerabilities detected by the individual, and G_{vul} is the number of genes that detected at least one vulnerability.

4 Algorithms

The nature-inspired algorithms are used to generate attack vectors to be injected into the queries. Evolutionary Algorithms (EAs) are randomized search heuristics, inspired by the natural evolution of species [2, 12]. The main idea is to simulate the evolution of candidate solutions for an optimization problem. GA is an example of an EA [2].

The basic concept of GA is designed to simulate processes in a natural system necessary for evolution, specifically those that follow the principles first laid down by Charles Darwin - the survival of the fittest [12]. GAs are EAs that use techniques inspired by evolutionary biology, such as inheritance, mutation, selection, and crossover. More information about the steps of this algorithm can be found in [2, 12].

The area of Swarm Intelligence (SI) relies on the collective intelligence of agents that interactively explore the search space. Some of the best-known areas of swarm intelligence are ACO, Particle Swarm Optimization, and bees-inspired algorithms [13, 14].

The ACO algorithm is essentially a system based on agents which simulate the natural behavior of ants, including mechanisms of cooperation and adaptation [16, 17, 18]. In real life, ants indirectly communicate among themselves by depositing pheromone trails on the ground, influencing the decision processes of other ants. This simple communication form among individual ants causes complex behaviors and capabilities in the colony. The real ant behavior turns into an algorithm establishing a mapping between (1) the real ant search and the set of feasible solutions to the problem; (2) the amount of food in a source and the fitness function; and (3) the pheromone trail and an adaptive memory [18]. The pheromone trails serve as distributed, numerical information which ants use to probabilistically build solutions to the problem to be solved

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08760-8_38

and which they adapt during the execution of the algorithm to reflect their search experience. More information about the steps and the formulas used to initialize/update the pheromone trails of this algorithm can be found in [4, 13, 14, 17].

The minimal model of swarm-intelligent forage selection in a honey-bee colony, that ABC algorithm adopts, consists of three kinds of bees: employed bees, onlooker bees, and scout bees [19, 20]. In ABC each iteration of the search consists of four steps: (1) sending the employed bees onto their food sources and evaluating their nectar amounts; (2) after sharing the nectar information of food sources, selecting food source regions by the onlookers and evaluating the nectar amount of these food sources; (3) determining the scout bees and then sending them randomly onto possible new food sources; and (4) memorizing the best food source [19, 20]. These four steps are repeated through a predetermined number of iterations defined by the user. In a robust search process, the exploration and exploitation processes must be carried out together. In the ABC algorithm, while onlookers and employed bees carry out the exploitation process in the search space, the scouts control the exploration process. More information about the steps of this algorithm can be found in [2, 4, 19, 20].

5 Experimental Results

In order to implement the approach described, we develop a tool in Java. To conduct the experiments, we collected different open-source web applications purposefully vulnerable as Bricks, bWAPP, and Twitterlike. All experiments were performed on a Raspberry PI 4 Model B with 8GB of RAM and a quad-core 64-bits of 1.5Ghz.

In order to obtain the best combination of parameters, several smoke tests were performed. Table 2 shows the best combination of parameters obtained for the algorithms GA, ACO, and ABC.

Table 2. Best parameters.

Alg.	Parameters
GA	Max generations: 50; population size: 20; elitism: no; selection method: tournament (size 6); crossover method: one cut (probability: 0.5); mutation probability: 0.05
ACO	Max iterations (*mi*): 30; number of ants: 80; *mi* without improvement: 2; number of modifications: 5; Q: 100; q probability: 0.1; pheromone evaporation rate: 0.3; pheromone influence rate: 0.3
ABC	Max iterations: 100; number of employed bees: 75; number of onlooker bees: 100; number of scout bees: 10% of employed bees; number of modifications: 11

We have first identified SQL Injection vulnerabilities manually since there is not an official number of these vulnerabilities. The biggest problem with a manual approach is that it takes a long time to detect all the vulnerabilities.

The results produced by our approach were compared with manual analysis and with the tools WAP and SonarPHP that use a static analysis approach. We also compare our results with the results obtained in previous works of the same authors that also use GA

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08760-8_38

[3], ACO [4], and ABC [4]. The solutions in these works are represented differently. These results are shown in Table 3.

Table 3. Best results.

Project	Manual	GA	ABC	ACO	GA [3]	ABC [4]	ACO [4]	WAP	SonarPHP
Bricks	12	**12**	**12**	**12**	11	11	11	11	**12**
bWAPP	56	**52**	33	33	30	47	51	15	14
Twitterlike	17	**13**	**13**	**13**	12	10	10	5	9

As we were able to see, when comparing with static analysis, our approach was able to identify more vulnerabilities. All the algorithms implemented present better results in comparison with static analysis. GA, using the representation of the individuals shown in this paper, proves to be more efficient in terms of the number of vulnerabilities detected.

As we can see most vulnerabilities are detected with GA, however, for the case of bWAPP and Twitterlike GA did not find all vulnerabilities. This is probably due to the fact that an individual has a fixed genome in terms of size, which sometimes leads to invalid queries. An approach with a dynamic genome size could potentially bring better results in terms of total SQL injection vulnerabilities found.

6 Conclusion

SQL injection can cause serious problems in web applications. In this paper we presented an approach to detect SQL Injection vulnerabilities in the code base, using a white-box approach. To optimize the search of potential vulnerabilities in the code we use nature-inspired algorithms: GA, ACO, and ABC. The optimization problem was formulated to find the best set of attack vectors. The proposed approach was divided into two steps, the first being a pre-analysis of queries in the source code that were used for the next phase, in which the various algorithms were used. This abstraction of concepts is quite convenient, allowing scalability of functionalities and the opportunity to implement several algorithms.

With more optimizations, better adaptations to the code of the algorithms, we think that it will be possible to improve the results obtained. The tool should be expanded to support multiple languages. At this phase, our approach only supports PHP applications. The scope of this article was constrained to SQL injection, but other vulnerabilities could potentially benefit from this approach.

Acknowledgements

This work was supported by national funds through the Portuguese Foundation for Science and Technology (FCT) under the project UIDB/04524/2020.

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08760-8_38

References

1. Stock, A., Glas, B., Smithline, N., Gigler, T.: OWASP Top Ten. OWASP Homepage. https://owasp.org/www-project-top-ten/, last accessed 2022/02/18.
2. Yang, X.-S.: Nature-Inspired Optimization Algorithms. 1st edn. Elsevier (2014).
3. Batista, K., Bernardino A.M, Bernardino E.M: Exploring SQL Injection Vulnerabilities Using Genetic Algorithms. Proceedings of the XV. international research conference, Lisboa (2021).
4. Batista, K., Bernardino E.M, Bernardino A.M: Detecting SQL injection vulnerabilities using Artificial Bee Colony and Ant Colony Optimization. Lecture Notes in Networks and Systems, Springer (2022).
5. McKinnel, D.R., Dargahi, T., Dehghantanha, A., Choo, K.-K.R.: A systematic literature review and meta-analysis on artificial intelligence in penetration testing and vulnerability assessment. Computers & Electrical Engineering, 75, pp. 175-188 (2019).
6. Niculae, S.: Applying Reinforcement Learning and Genetic Algorithms in Game-Theoretic Cyber-Security. Master Thesis (2018).
7. Nguyen, T.H., Kar, D., Brown, M., Sinha, A., Xin Jiang, A., Tambe, M.: Towards a Science of Security Games. In: Toni B. (eds) Mathematical Sciences with Multidisciplinary Applications. Springer Proceedings in Mathematics & Statistics, 157. Springer, Cham (2016).
8. Alenezi, M., Javed, Y.: Open source web application security: A static analysis approach. In: 2016 International Conference on Engineering and MIS (2016).
9. Tripathi, J., Gautam, B., Singh, S.: Detection and Removal of XSS Vulnerabilities with the Help of Genetic Algorithm. International Journal of Applied Engineering Research, 13(11), pp. 9835-9842 (2018).
10. Friedl, S.: SQL Injection Attacks by Example. http://www.unixwiz.net/techtips/sql-injection.html (2017), last accessed 2022/02/18.
11. Mishra, D.: SQL Injection Bypassing WAF. https://www.owasp.org/index.php/SQL_Injection_Bypassing_WAF, last accessed 2022/02/18.
12. Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computing. Springer, Heidelberg (2015).
13. Kennedy, J., Eberhart, R.C., Shi, Y.: Swarm intelligence. Morgan Kaufmann, San Francisco (2001)
14. Wahab, M.N.A., Nefti-Meziani, S., Atyabi, A.: A comprehensive review of swarm optimization algorithms. PLoS One, 10(5), e0122827 (2015).
15. Karaboga, D., Akay, B. A survey: algorithms simulating bee swarm intelligence. Artif Intell Rev 31, 61 (2009).
16. Dorigo, M.: Ottimizzazione, apprendimento automatico, ed algoritmi basati su metafora naturale (Optimisation, learning and natural algorithms). Doctoral dissertation. Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy (1991).
17. Dorigo, M., Maniezzo, V., Colorni, A.: The ant system: Optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics 26, pp. 29–41 (1996).
18. Gambardella, L.M., Taillard, E.D., Dorigo, M.: Ant colonies for the quadratic assignment problem. J. Operational Research Society 50(2), pp. 167–176 (1999).
19. Karaboga, D.: An idea based on honey bee swarm for numerical optimization, Technical report TR06. Erciyes University, Engineering Faculty, Computer Engineering Department (2005).
20. Karaboga, D., Akay, B.: A comparative study of Artificial Bee Colony algorithm. Applied Mathematics and Computation, 214:108–32 (2009).

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08760-8_38