Mutation Based SQL Injection Test Cases Generation for the Web Based Application Vulnerability Testing

Benikhlef Ilies[1, a], Wang Chenghong [2, b], Gulomjon Sangirov[3, c]

[1]Department of Computer Science and Technology, Harbin Engineering University, 150001, China

[2]Department of Information Technology and Services, Syracuse University 471023, USA

[3]Department of Information and Communication Engineering, Harbin Engineering University, China

[a]email: b_ilyes@hrbeu.edu.cn, [b]email: skiphoopwow@gmail.com

Keywords: Mutation testing, Test Cases Generation, SQL Injections, mutation Analysis, fuzz testing, vulnerability detection

Abstract. Security testing is the process of detecting the exploited defects which conduct attacks. Since SQL Injection vulnerabilities are one of the most common threats of a web-based application, testing still the most important technique in order to gain confidence that an articraft behaves as expected. This scenario occurs when untrusted inline simple inputs are accepted as a database input which can lead to some security breaches such as altering the intent of the original query and getting some privileges, leaking of private information, authentication bypassing…etc. Although the awareness of SQL Injection attacks, the risk is increasing and the consequences are very severe, still many people do not have very concrete ideas on how to prevent against them. It becomes not easy to check and test the application data flaws, but since the manually testing is hard and time-consuming security testing and fuzzing test remain the tools where almost all worldwide companies focus are concentrated rather than web application scanners. In practice Software's Vulnerabilities detections mean the obtaining of adequate test cases set that contain effective queries or attacks that reveal new data flaws and define the risk, identifying the unexpected behavior by performing test cases generation based on the mutation to mitigate that risk with new attack scenarios. In this paper we applied the idea of mutation-based test cases generation to get a new set of test cases to test against SQL Injections attacks. The results can be used for web-applications penetration testing, fuzz testing, SQL injection detection and prevention, it can also be used to compare between brute force tools, web-application scanners effectiveness, enlarge the space of test cases what can reduce the time costs of testing process and finally software's quality assurance.

Introduction

Since the hackers first target is web-based applications that one have been developed continuously, fundamentally and radically, in our daily life where the common thoughts say that this web applications might be secure and must do much better job than other applications. For us as society and individuals expect it to behave correctly, but unfortunately the majority of thus applications are not secure and may also contain some kinds of data flaws or bugs that can be exploited by attackers. SQL Injection still remain one of the most exploited web applications vulnerabilities, where the threat comes from the obtaining of unrestricted access to databases by inserting malicious strings to SQL queries via the web application to gain access, control and privileges of its administrator.

Since web applications are complex and face a massive amount of threats and since the manual testing for security issues are hard and time-consuming, it becomes not easy to verify if they are secure against this kind of threats or not. In practice the appropriate way is to identify and remove software's defects, by test cases generation automation which can save time and cost. In Software testing it's prominent to call for black-box testing technique however the tester will interact with system's user interface by providing inputs and inspect outputs without knowing how and where the inputs are worked upon. Software Testing means the inspection of the application behavior on a finite set of test cases. In this paper a new optimal design of soccer robot control system which is based on mechanical analyses and calculations on the pressure and transmutation states of chip kick mechanics, this new control system with high precision for speed control and high dynamic quality.

SQL Injection Scenario:

The concept of injection attacks is to inject malicious code so as to alter the intent of original query [1-6]. Injection flaws occur when untrusted data are sent to an interpreter as a part of a command or query, then attacker's hostile data can trick the interpreter and execute unintended commands or accessing data without proper permission.

SQL Injection queries are usually tested for correctness by executing them on datasets, to see if they give the desired results on each dataset. The invalid ones are often the result of small changes made by mutation. The idea behind this method is to mutate attack model either by changing parts of SQL injection statements or by changing the order of the execution.

The test cases execution may be very costly and time-consuming and the analysis of test cases execution result can be more complex. Mutation testing propose automatic mechanism to get more test cases and discover more bugs at once, it sounds good because some professionals testers need to complete the check and obfuscation of test cases manually to reveal more bugs on the AUT(Application Under Test). The best way to improve the effectiveness of SQLi testing is to get more coverage, more bugs with fewer test cases in a short time [16]. In our point of view regardless of the similarity of the successful test cases between each other (e.g., encoded by the same encoding method or followed with similar prefix or suffix), but almost certainly it all disclose a significant amount of bugs. More specifically good test cases may have all combinations that help to discover flaws, so more we cover all combinations or probabilities where SQLi can take place.

Mutation Operators and mutation Approach:

The mutation approach describes mainly the process of generating test cases than comparing the newly generated mutants with original ones in question of quantity and quality. To achieve the requirements of best generation it should be rather built strong mutation operators using specific techniques to expand them.

In this section we first propose the mutation operators to assess the quality of test cases generation. This mutants modify different features on the SQL queries. SQL attack vector is divided into the three following types of characteristics. The behavior conversion (behavior change), syntax correction (syntax repairing) and overall confusion (obfuscation). We implemented some new faults which are given bellow:

- → Replace apostrophe character with UTF-8

  ’ = %EF%BC%87)
- → Replace apostrophe with its illegal double Unicode counterpart apostrophe

  ' = %00%27)
- → Replace UNION ALL SELECT with UNION SELECT

✧ According to the techniques used for mutation we can define three types of mutation:

1) Value mutations:

This kind of mutations tends to change the values to detect errors in the tested software. We usually change this values of constants to bigger or smaller values.

2) Decision mutation:

The decisions/conditions are changed to check for design errors. Typically thus changes could be an arithmetic operator like (<, >, =, +, -, *, /) or also logic operators (AND, OR, NOT).

3) Statement mutation:

Changes the statements by deleting or duplicating the line using evasion, stitching, obfuscation and replacement of the statement.

✧ According to the output analysis and comparison we define also 3three kind of mutation:

4) Strong mutation:

To consider the mutation strong, the results of the test cases mutation should be different from the original one. The output length, variable assignments and the number of data retrieval are items which distinguish strong mutation. So that if the mutation results satisfy the reachability and necessity criteria (requirement). This can observe by tracing and monitoring the output state changes [15, 16].

5) Weak mutation:

A mutation is said to be weak if there is a less probability of generating any solution from any mutant is efficient. Therefore, if the mutation results satisfy the reachability and requirements criteria [17].

6) Firm mutation:

Firm mutation is the manner that the number variations that may mutate depend on the framework characteristics which affect the evaluation of mutation effectiveness and coverage [17].

We proposed mutation technique to force the generation of adequate test cases samples for the aim of vulnerability testing. The operators mutate source code to inject the vulnerabilities in the library function calls and unsafe implementations elements. The mutant generated by the operators are killed by test cases that expose these vulnerabilities. The selecting test cases able to effectively kill mutants when performing weak mutation testing are collected as original test cases and involve them on test cases generation algorithm in order to generate suitable test cases for SQL Injection Vulnerabilities testing.

We perform test cases generation algorithm to be used to test vulnerable applications using the HTTP fuzzer (fuzz testing).

The mutation algorithm consists of making changes on a valid test case extracted from known Vulnerability Scanners like: FuzzDB, Wfuzz, Jbrofuzz...etc.

Related Work

Mutation is a fault-based testing strategy that measures the quality of testing by probing whether the test set (test input data) used in testing can reveal certain types of faults. Mutation is not to reveal new faults, but also establish a level of reliability and confidence accordingly to software quality assurance. Since SQL Injection vulnerabilities get more sophisticated and complex, simple test cases cannot cover them, and since manual testing is getting more time consuming and high cost, we need more sophisticated test cases too. Mutation testing is powerful [11], and effective in detecting bugs [12, 13].

Automatic test cases generation through mutation is a new method which has been researched with less practical employment and guidance [14]. Generate good test cases with high mutation score is a big challenge [15] due to the lack of automated tools to generate tests [1]. This paper aims to perform test cases generation automation based on the mutation to see how this technique can help to test web-based application SQLI vulnerabilities and to solve the problem of leak off test cases set. The challenging part of generation test cases is to success the test purposes because it is impossible to achieve an abundantly tested application, assumed that the number of test cases needed for the fully testing software application is infinite; subsequently a suitable set of test cases will be able to detect a prodigious number of faults. Good test cases should have the quality to cover more features of test objectives

Proposed Generation Method

The technique used in the elaboration of this algorithm is an evasion technique that is employed against attack to avoid the signature-based detection systems. A signature is a pattern of known attack payload, it usually consists of one or more SQL keywords, delimiters, and expressions. Evasion technique obscures input strings when automated generating of test cases.

Evasion is depending on how the information has been introduced in the query, we need many techniques to break the syntax. This obfuscation and ambiguities might lead to SQL injection in the tested software, if not fixed it can easily pass through testing phase undetected while the number of test cases is not enough. Sophisticated matches, Hex Encoding, Char Encoding, Shunning keywords are the main techniques used for evasion

Table 1 Number of generated Test Cases Based on Mutation

Evasion Mode	*Original Query (SELECT user_name FROM information_schema.tables;)*
URL Encoding	SELECT %75ser_%6Eame FROM information_schema.tables;
Double Encoding	SELECT %2575ser_%256Eame FROM information_schema.tables;
Unicode Encoding	SELECT %u0075ser_%u6Eame FROM information_schema.tables;
Invalid Hex Encoding	SELECT %us%er_%na%me FROM information_schema.tables;
Space	information_schema.tables
Backticks	\`information_schema\`.\`tables\`
Specific Code	/*!information_schema.tables*/
Alternative names	information_schema.partitions information_schema.statistics information_schema.key_column_usage information_schema.table_constraints

The main goal of this research is to automatically generate test cases based on mutation approach. To evaluate the performance of the mutation algorithm we reinforce our study with coverage and effectiveness assessment results analysis. Mutation testing allows us to compare the test cases effectiveness and the vulnerability scanners reliability.

This experiment was mainly performed to answer the ensuing questions:

1. Do the mutation operators help to generate adequate test cases?
2. Generate a large amount of test cases is likely to detect SQLI vulnerabilities?

We use benchmark test data set containing effective and ineffective attack test cases from 3 common discovery and attacks test cases database which are used by most of the vulnerability scanners tools.

After each successful test, the scanner result was evaluated to determine which test cases was effective and which kind of vulnerabilities were discovered.

We implement an algorithm for mutation-based test cases generation aimed at SQL Injection vulnerabilities testing. The approach consists of generating mutants based on the proposed operators according to evasion technique. The initial and mutated statements were injected to vulnerable web applications using fuzz testing to monitor and evaluate the test cases set after query execution. To perform mutation analysis we used 5 web application available from the open source web application repository to evaluate the resulted test cases validity and correctness the same test is repeated many times using many initial and mutated test cases databases samples. The set of test cases which includes a number of queries that makes an application vulnerable to SQL injection. For each of the 5 web applications we inject an initial test cases set which includes mimic parameters, commands, and characters than assess the coverage against 20 vulnerable application.

Test results

The table below describes the effectiveness of second order mutation which divulges that when the rate of initial test cases set effectiveness is greater, the second order mutation will increase the mutation score. The second order initial test cases space we are talking about is export the first order mutation successful test cases to generate more effective test cases, which we perfectly improve the effectiveness of the test cases.

The test cases effectiveness and performance analysis was elaborated making a comparison between the analysis result using Burp suite. Therefore test cases generation using mutation approach is told to be a weak mutation, but the investigation showed that if the initial test cases space is effective and has good coverage it lead to good testing results (has a good number of detected vulnerabilities or a great number of data retrieval). Since we are using a big scale of test cases space we took interest on the entire number of strong test cases. For example the number of strong mutants varies from 29 to 368 in the first benchmark using the test cases set 1 (Fuzz DB) as defined in table 1.1

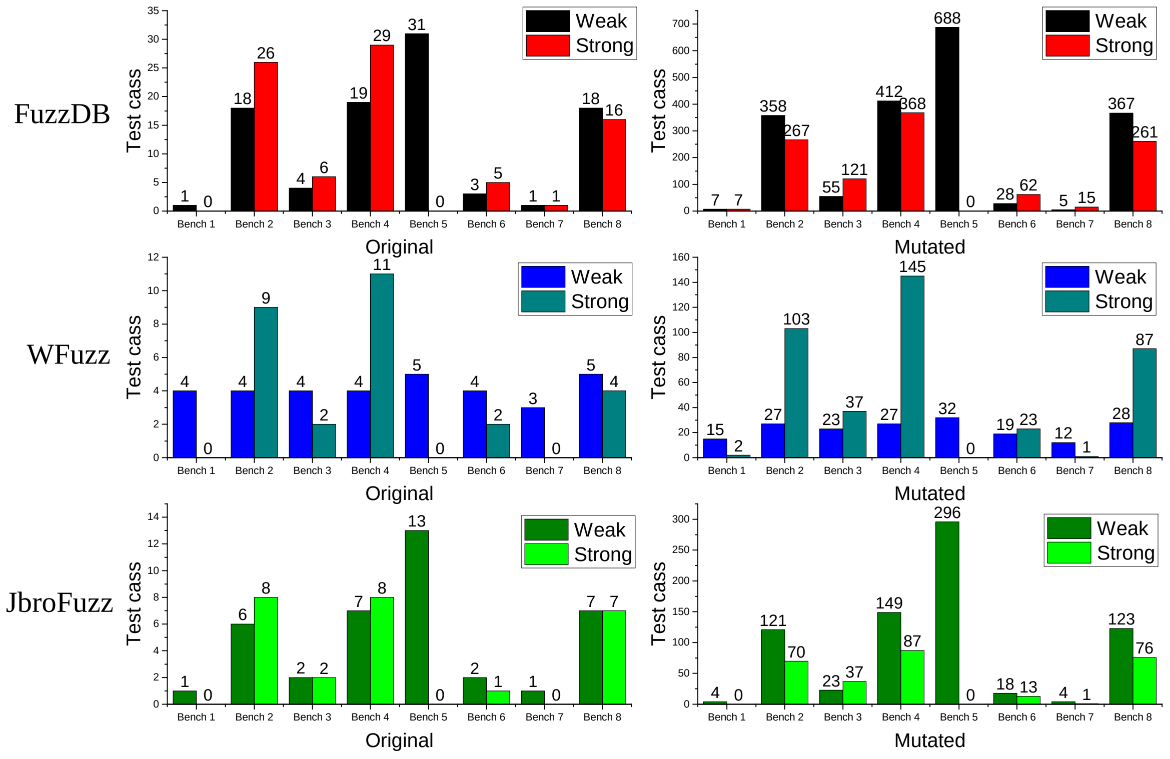

Fig.1. Comparison between the test cases database performance analysis

Conclusion

Mutation testing approach is fault-based testing to firstly generate test cases and Secondly measure the effectiveness of test cases which have the potential to detect real faults in the program. It suffers from expensive computation, beside the original program every mutant has to be executed against all the test cases where an equivalent mutant always produces the same output on the original program. Experiments with some benchmark confirmed that our technique detects more bugs than original ones. From the experimental results, it can be concluded that the proposed test cases generation method works better than black-box test cases generation giving better-quality test cases in terms of coverage and efficiency which helps to detect SQL injection vulnerabilities

Acknowledgement

This paper, is funded by the international Exchange Program of Harbin Engineering University for Innovation-Oriented Talents Cultivation.

References

[1] DEREZIŃSKA Anna. An experimental case study to applying mutation analysis for SQL

queries. Computer Science and Information Technology, 2009. IMCSIT'09. International Multiconference on. IEEE, 2009; 559-566.

[2] M Dave and R Agrawal. Search-based techniques and mutation analysis in automatic test case generation: A survey. Advance Computing Conference (IACC), 2015 IEEE International. Banglore, 2015; 795-799.

[3] APPELT Dennis, NGUYEN Cu Duy, BRIAND Lionel C., et al. Automated testing for SQL injection vulnerabilities: an input mutation approach. Proceedings of the 2014 International Symposium on Software Testing and Analysis. ACM. San Jose, CA, USA, 2014; 259-269.

[4] SHAHRIAR Hossain et ZULKERNINE Mohammad. MUSIC: Mutation-based SQL injection vulnerability checking. Quality Software, 2008. QSIC'08. 8th International Conference on. IEEE. 2008; 77-86.

[5] J Tuya, M J Suarez-Cabal and C de la Riva. SQLMutation: A tool to generate mutants of SQL database queries. Mutation Analysis. Second Workshop on, Raleigh, NC. 2006: 1-1.

[6] TUYA Javier, SUÁREZ-CABAL, Ma José, et DE LA RIVA Claudio. Mutating database queries. Information and Software Technology. Spain. 2007; 49(4): 398-417.

[7] PAPADAKIS Mike et MALEVRIS Nicos. Mutation-based test case generation via a path selection strategy. Information and Software Technology. 2012; 54(9): 915-932.

[8] GUTIERREZ-MADRONAL Lorena, SHAHRIAR Hossain, ZULKERNINE Mohammad et al. Mutation Testing of Event Processing Queries. Software Reliability Engineering (ISSRE). 23rd International Symposium on. IEEE. 2012; 21-30.

[9] MOUELHI Tejeddine. *Mutation analysis applied to security tests*. *ENST* Bretagne. 2007.

[10] Offutt, A.Jefferson. A Practical system for mutation testing: help for the common programmer. Test Conference. 1994 Proceedings. International. IEEE. 1994.

[11] Mateo, Pedro Reales, Macario Polo Usalo and Jeff Offutt. Mutation at system and functional levels. Software testing, Verification, and validation Workshop (ICSTW). 2010 Third International Conference on IEEE. 2010.

[12] Assis Lobo de Oliveria, Andre, Celso Gonyalves Camilo-Junior, and Auri MR Vincenzi. Acoevolutionary algorithm to automatic test case selection and mutant in mutation testing. Evulutionary Computation (CEC), 2013 IEEE Congress on. IEEE. 2013.

[13] Papadakis Mike, and Nicos Malevris. Mutation bases test case generation via path selection strategy. Information and Software Technology. 2012; 54(9): 915-932.

[14] Souza Francisco, Carols F, Papadakis M, Delamaro EM et al. Test Data Generation Technique for Mutation Testing: A Systematic Mapping. Workshop on Experimental Software Engineering (ESELAW'14). 2014.

[15] S. Anderson, "Mutation Testing," School of Informatics, 2011.

[16] W. E. Howden, "Weak Mutation Testing and Completeness of Test Sets," *IEEE Transaction on Software* Engineering, vol. 8, pp. 371-379, 1982.

[17] M.R.Woodward and K.Halewood, "From Weak to Strong, Dead or Alive? An Analysis of some Mutation Testing Issues," *In Software Testing, Verification, and Analysis. Proceedings of the Second Workshop on IEEE,* pp. 152-158, 1988.