Automated Security Testing Framework for Detecting SQL Injection Vulnerability in Web Application

Nor Fatimah Awang[1(✉)] and Azizah Abd Manaf[2]

[1] Faculty of Defence Science and Technology,
National Defence University of Malaysia, Kuala Lumpur, Malaysia
norfatimah@upnm.edu.my
[2] Advanced Informatics School (UTM AIS), UTM International Campus,
Kuala Lumpur, Malaysia
azizaham.kl@utm.my

Abstract. Today almost all organizations have changed their traditional systems and have improved their performance using web-based applications. This process will make more profit and at the same time will increase the efficiency of their activities through customer support services and data transactions. Usually, web application take inputs from users through web form and send this input to get the response from database. Modern web-based application use web database to store all critical information such as user credentials, financial and payment information, company statistics etc. However error in validation of user input can cause database vulnerable to Structured Query Language Injection (SQLI) attack. By using SQLI attack, the attackers might insert malicious code in the user input and trying to gain access to the confidential and sensitive data from database. Security tester need to identify the appropriate test cases before starting exploiting SQL vulnerability in web-based application during testing phase. Identifying the test cases of a web application and analyzing the test results of an attack are important parts and consider as critical issues that affects the effectiveness of security testing. Thus, this research focused on the developing a framework for testing and detecting SQL injection vulnerability in web application. In this research, test cases will be generated automatically based on SQLI attack pattern and then the results will be executed automatically based on generated test cases. The primary focus in this paper is to develop a framework to automate security testing based on input injection attack pattern. To test our framework, we install a vulnerable web application and test result shows that the proposed framework can detect SQLI vulnerability successfully.

Keywords: Security testing · Penetration testing · Test case generation

1 Introduction

Web application systems are one of the most ubiquitous software systems in use today. Since they appeared they have grown quickly and have evolved faster than other software systems. More than one billion people worldwide using the Internet and web

H. Jahankhani et al. (Eds.): ICGS3 2015, CCIS 534, pp. 160–171, 2015.
DOI: 10.1007/978-3-319-23276-8_14

applications as their daily routine activities for a variety of reasons, such as communicating with others, conduct research, shopping, banking and electronic commerce [1]. The web application is consider both, as a communication and a source of information [2]. It offers a collection of various services and resources such as customer support services, online banking and data transactions [2]. Most of the organizations use the web applications to make more profit and at the same time to increase the efficiency of their activities. The growth of these applications gave a high impact and business opportunity to the organization. As web applications become adopted by more and more organizations, they have become more complex and sophisticated. In many cases their success is crucial for the success of the organizations. Thus ensuring security of the Web application systems is a big concern for organizations.

Web application vulnerabilities represent huge problems for companies and organizations. These vulnerabilities leave organizations' web applications exposed to attack and majority of all security problems in web application is caused by string based injection through web form such as SQL injection and cross site scripting. SQL injection vulnerabilities have been described as one of the most serious attacks for Web applications [3, 4]. SQL injection vulnerabilities are based on injection strings input into database to construct SQL queries and may allow an attacker to extract data from database. A web application is consider vulnerable to an SQL injection attack if an attacker is able to insert SQL statements into an existing SQL query of the application and extract sensitive and confidential data from database. This SQL injection vulnerability usually occurs when web application does not properly sanitize the user input.

With the majority of vulnerability exists in web applications today, it is important to evaluate and detect the vulnerability of web application before it is sent to production [5]. Detecting and preventing vulnerabilities in web application has become an important concern for organizations. Many organizations are starting to take initiatives to prevent these types of attacks. To minimize the probability of vulnerabilities exist in web applications, organizations need some methodologies or approaches to increase efforts to protect against web-based application attack [5, 6]. Therefore, organizations have a big task to implement security testing methodologies into the software development life cycle. The purpose of security testing is to find any security weaknesses or vulnerabilities within an application and document all the vulnerabilities to help developer to fix them. In this paper we develop a framework for testing and detection SQL injection vulnerabilities in web applications. We automate the process of security testing based on input injection attack pattern. We separated the framework into three stages. (i) Develop Test Cases Generator which is used to generate test cases. In the first stage, we apply permutation technique in order to generate test cases automatically; (ii). Develop Attack Generator. This generator will be used to automate the injection attack process based on input generated in the first stage. (iii). Response Generator. The goal of response generator is to analyze and determine whether an attempted attack has been successful or not. Some manual work is still required before automating the testing process. The process of modeling test case generation may not be very time consuming if the tester has knowledge about the internal structure of the application and type of SQL injection vulnerabilities.

This idea is using penetration testing framework adapted from Open Web Application Security Project (OWASP) and security testing lifecycle as shown in Fig. 1 [6–8].

On the middle of Fig. 1, by selecting and combining some components in penetration testing framework and security testing lifecycle, we proposed the new framework to automate the security testing in web application. In this paper, we are focusing on SQLIA due to the common vulnerabilities that have evolved in the last decade. The rest of the paper, we follow the idea of developing our framework. Our framework closely related to [7, 9, 10].

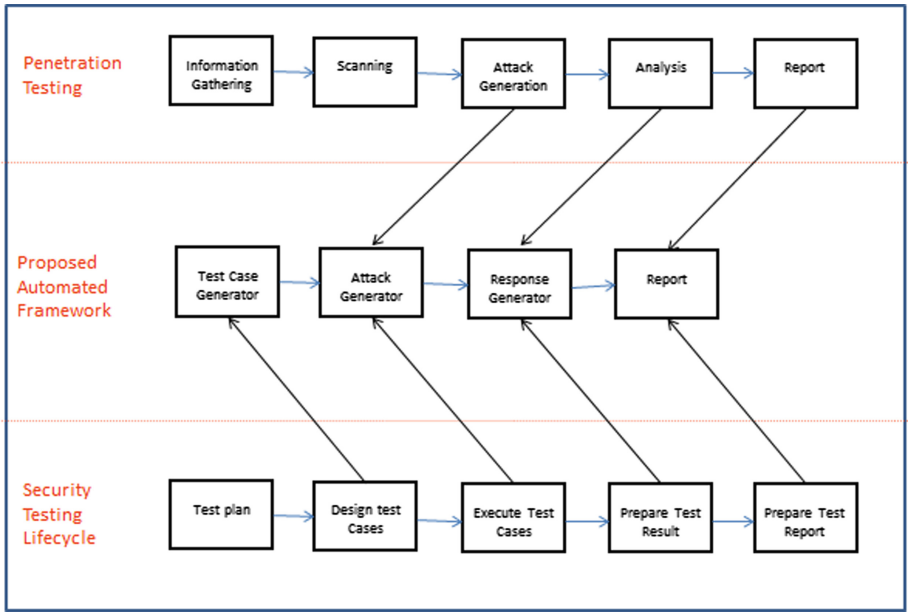

Fig. 1. Transition to new framework

The structure of this paper is as follows. Section 2 briefly describes the background of SQL injection attack. Section 3 discusses the proposed framework, Sect. 4 presents test result and Sect. 5 presents the conclusion.

2 Background of SQL Injection Attack (SQLIA)

SQL injection attacks are one of the topmost threats for applications written for the Web. In OWASP listing, SQLIA is always appear in top ten list. These attacks are launched through specially crafted user input on web applications that use input string operations to construct SQL queries [11]. These attacks used by attackers to steal data from back-end database of web application. It takes the advantage of dynamic input of web applications that allow users to insert data such as login form and registration page. This type of attack usually takes place when attackers insert some special code into web form. Online forms such as login prompts, search enquiries, guest book and registration forms are always the main target of SQL injection vulnerabilities.

The simple test to check for SQLIA is to append 'OR 1 = 1' [12] or just single quote (') to any input of any form and wait for the data response returned. The response could be error message or any information that may give some clue to the attackers to conducting next step of attack. In the rest of this section, we discuss three important characteristics of SQLIAs that we use for describing SQLIA: how SQL works, injection based on attack input, injection based on attack intend and injection based on SQL types.

2.1 How SQL Works

SQL is stand for Structure Query Language and originally developed in the early 1970's by Edgar. SQL is used for accessing database servers including MySQL, Oracle and SQL server. Web programming language such as Java, ASP.NET and PHP provide various methods for constructing and executing SQL statements. In addition, SQL language is a communication way between users and database in order to allow the user to interact with database. SQL statement can modify the structure of databases and manipulate the contents of databases [16].

2.2 SQL Injection Attack Based on Input Source

Malicious SQL statements can be introduced into a vulnerable application using many different input mechanisms. In this section, we explain the most common mechanisms. Table 1 explains the most common input that usually used by attackers [12–15].

Table 1. SQLIA based on input source

Classification	Description
Injection through user input	Injects malicious SQL commands into user input query in web form based on GET and POST parameter.
Injection through cookies	Injects malicious command and modified cookies variables containing SQLIA. Cookies are files used for an origin website and generated by website and send this state information to a user's browser or client machine. This state information can be used for authentication or identification of a user session. Cookie variables sometimes are not properly sanitized and can be used to bypass authentication or make any SQL query by injecting arbitrary SQL code.
Injection through server variables	Server variables are components of the message header that contain HTTP header, network header and environmental variable. They define the operating parameters for HTTP transaction such as request and response information. By using the request and response field, attackers can submit arbitrary input and exploit through HTTP header.
Second order injection	This injection occurs when data input stored in a place and then used in a different SQL query without correct filtering or without using parameterized queries

2.3 SQL Injection Attack Based on Attack Intend

SQLIA can also be characterized based on the goal, or intent, of the attacker. Attacks have been characterized by their intent are summarized in Table 2 [9, 12–14].

Table 2. SQLIA based on attack intend

Classification	Description
Identifying injectable parameter	The first step to identify SQLI vulnerabilities is to identify which input parameter values are likely to have vulnerabilities into SQL query statements. Attackers will use their knowledge to discover which parameters and user input fields are vulnerable to SQLIA
Extracting data	The goal of these types of attack is to extract data values from database. Attackers usually use GET and POST method to extract response from web server.
Adding or modifying	To add or change information in database
Performing denial of Service	To shutdown, locking or dropping database. Attackers can extract or dump the complete database by using "UNION" and "SELECT" commands.
Bypassing authentication	Attackers occurs use these types of attacks to inject some malicious code such as 'OR 1 = 1' into login form and try to bypass database and application authentication mechanism
Executing remote commands	To execute arbitrary commands on the database including stored procedure or functions
Determine database schema	Attackers use these types of attacks to obtain information about table names, column names, column type, etc.

2.4 SQL Injection Attack Based on Attack Techniques

There are a variety of techniques that attackers can use to perform these attacks [9, 12–14]. The most common SQLIA based on the attack type are summarized in Table 3. The different techniques of attacks are generally not performed in isolation. Attackers may use and combine these techniques in order to achieve several attack types based on attackers' goal. In this paper, we just summarize all possible attack without writing detail. These types of attacks will be used in the next section to generate test case generation.

3 Our Proposed Framework

As shown in Fig. 1, our proposed framework adapted from penetration model and security testing lifecycle model. In our framework, we develop three main components and Fig. 2 shows our proposed framework.

1. Input Generator - input generator component is used to generate test cases based on injection attack pattern. We apply permutation technique in order to generate test cases automatically [17].

Table 3. SQLIA based on attack technique

Classification	Description
Tautologies	Inject code to one or more SQL query and these query will always evaluates to true. For example: query = "SELECT * FROM users WHERE username = 'name' AND password = 'pwd'"; Attackers can use tautologies to exploit this peace of code by inserting this value 'OR 1 = 1' to the text box of login page in WHERE username clause. This make the system will always evaluates the result to be true and bypass the authentication system. In addition, tautology method can inject string type, numerical type and comment type. Once the user has got the access, he can modify the data in the database and this can cause a major loss for the organization. Injection Attack = SELECT * FROM users WHERE username =″ OR 1 = 1– AND password = ″
Illegal/Logical	When faced with invalid queries, databases will provide an error message that can give detailed information about the type of database that is running and further information about the query. Using error messages rejected by the database, the attacker will use to find useful data. A number of different approaches can be used to generate invalid queries. For example single quote string, double quote or other logical errors can all be used to help identify information about the database. Injection Attack = SELECT * FROM users WHERE name = '/*! – */' AND password = ″
Piggy-backed	Piggy-Backed queries are used when the attacker would like to alter the developer statement and potentially run an entirely separate statement of their own. Insert additional queries to be executed. The secondary queries will be used to alter, delete or disable the application. Using the same example as before, the syntax of SQL defines the semicolon as a delimiter and executes the two separate statements. As a result, DROP syntax will remove the logs table from database. Injection Attack = SELECT * FROM users WHERE username = 'name' and password = ″;DROP TABLE logs –
Union	Injected query is joined with a safe query using the keyword UNION. UNION statement in SQL allows an attacker to combine two separate SELECT statements into one result. By Unioning on extra data from other tables, or the same table in the system the attacker is able to recover additional information. Assuming the number and types of the columns in both the users and Email Addresses tables match, and there is no user with the username ``, the database will union together the two sets. One containing all the Email Addresses in the database, the other containing all zero users with the username ″. The ending result is the display of all Email Addresses in the system to the user. Injection Attack = SELECT * FROM users WHERE username ="UNION SELECT * from EmailAddresses – AND password = ″

2. Attack Generator - this generator will be used to automate the injection attack process based on input or test cases generated in the first phase.

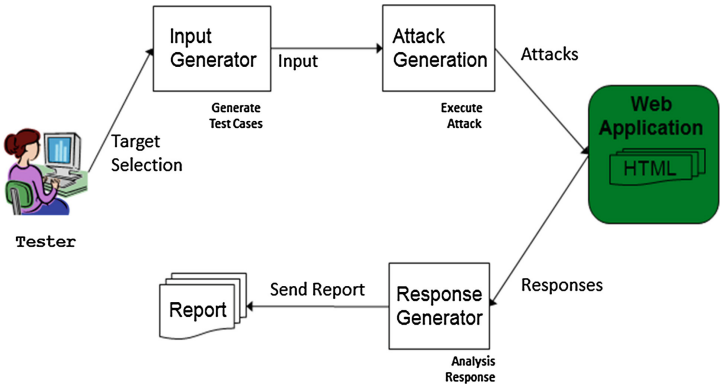

Fig. 2. Overview of proposed framework

3. Response Generator - The goal of response generator is to analyze and determine the response and output send by web server in order to detect vulnerabilities.

3.1 Input Generator

In this section, we discuss our approach in generating test cases automatically based on attack techniques. Based on input string in attack technique as shown in Fig. 3, we formulate and design the attack grammar and divide into six different patterns templates as listed below:

- #numeric = all numeric characters [0-9]
- #alphabet = all alphabet characters [A-z]

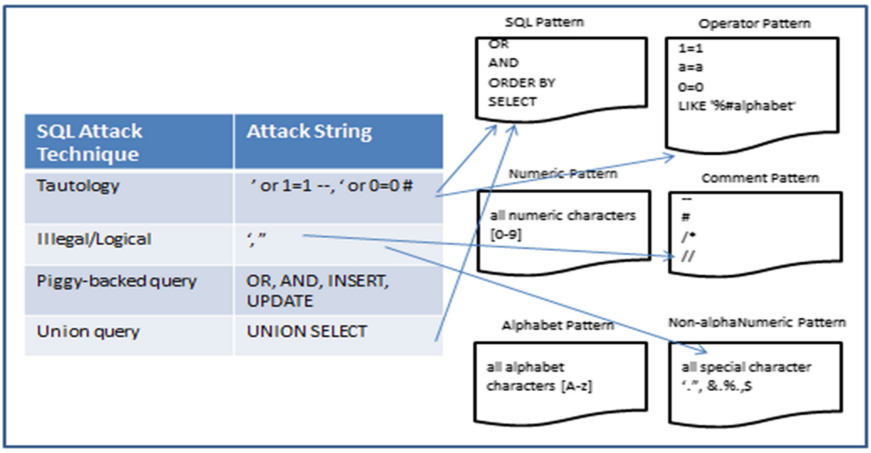

Fig. 3. Formulation of attack file to generate test cases

- #non_alphanumeric = not alphanumeric but printable (e.g.: punctuation)
- #sql = sql syntax (sql.txt)
- #operator = the operators used in various programming language (operators.txt)
- #inline_comment = various type of comments (inline_comments.txt)

Using permutation algorithm [17] this pattern generates different variations of test cases based on SQL injection attack as shown in Fig. 4. For example, from the tautology technique, the simple input string to inject to input field is 'OR 1 = 1. There are a lot of combination of tautology's input string in order to detect vulnerability such as 'OR 1 = 1–, 'OR 1 = 1#, etc. Testers are allowed to modify and insert new data in pattern template.

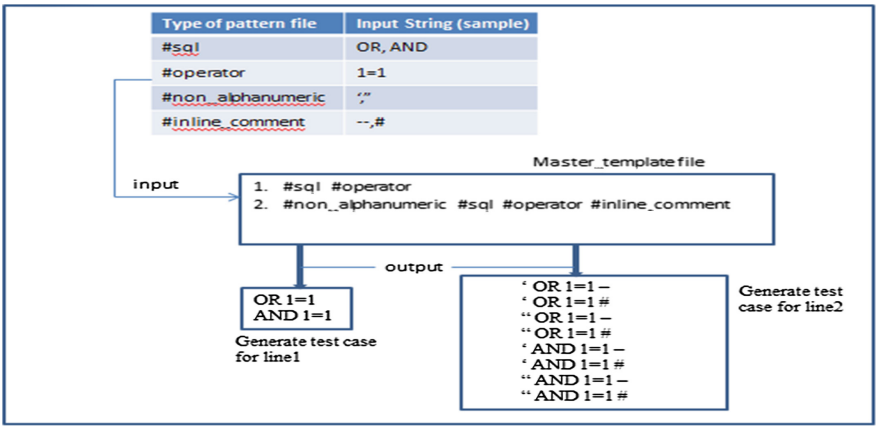

Fig. 4. Input generator component

In this section, we follow closely related paper for test case generation. [18–20]. The main important in this phase is master_template file. Master_template file will be executed after tester run the generator. The master template contains list of template for the application to perform permutation to generate more test cases. To form the template we introduce a syntax which started with '#' symbol to represent the data that needs to be replaced. The generator then generate the dictionary files that contains all pattern files (sql.txt, operators.txt, inline_comments.txt, etc.) to form list of string tokens for test cases. Each test case is generated automatically by using permutation algorithm based on the input line in the master_template file. Refer to Fig. 4.

Based on a sample in Fig. 4, we can see in master_template file, there are two lines listed and attack generator will generate the test cases separated and saved in a different file. By applying the permutation technique algorithm, we shall get the following results as stated in Table 4.

Table 4. Test Case Generation

Test cases for line 1:	Test cases for line 2:
OR 1 = 1	‘OR 1 = 1 –
	‘OR 1 = 1 #
	“OR 1 = 1 –
	“OR 1 = 1 #
AND 1 = 1	‘AND 1 = 1 –
	‘AND 1 = 1 #
	“AND 1 = 1 –
	“AND 1 = 1 #

3.2 Attack Generator

The fundamental objective of this section is the design of the framework that covers all steps to automate the injection attack process. After the generation of test cases have completed, Attack generator starts processing a set of target URL and target parameter. As mentioned before, some manual work is still required before automating the attack generator process. Tester is needed to identify target URL and target parameter. The test cases that have been generated in phase1 will be used as an input in this phase. In order to extract HTTP response and injecting to the target system automatically, we develop our framework with Apache HTTP Client API. We present an efficient algorithm to send many attacks and handle many HTTP response page as shown in Fig. 5. Attack generator component will use input.xml file to attack to target system by using POST or GET method and also identify which parameter is chosen to inject the input generated test cases from previous phase [21–24].

Sample of Input.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<client>
  <url>http://WackoPicko/users/login.php</url>
  <method>post</method>
  <parameters-group>
      <parameters>
      <param>username</param>
      <file>D:\\input.txt</file>
      </parameters>
         <parameters>
         <param>password</param>
         </parameters>
  </parameters-group>
</client>
```

3.3 Response Generator

After an attack has been launched, the analysis of response page will be send to the response generator. The response generator component uses attack specific response

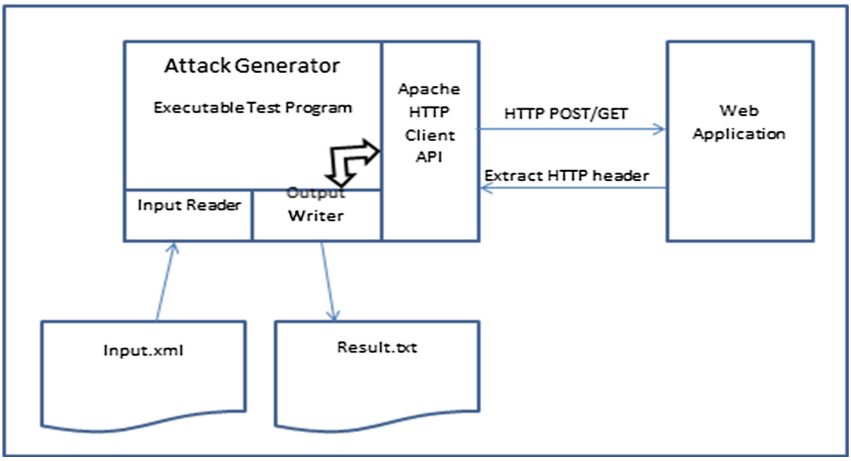

Fig. 5. Attack generator component

criteria to decide if the attack was successful. If the web application does not handle exceptions or server errors, the result for SQL error description will be included in the response page.

4 Result

This section presents the testing results that we have carried out to assess our framework. For the testing of our framework, two different vulnerable web applications were deployed locally and tested against SQLIA. This testing applications are namely as Mutillidae and WackoPicko website. Mutillidae website has an authentication bypass vulnerability, which allows the attacker to directly access the administrative functionalities. WackoPicko is an online photo sharing website that allows users to upload, comment and purchase pictures It is designed with a number of vulnerabilities, such as cross-site scripting and SQL injection [11]. Our framework was deployed by setting up Eclipse development environment with Java Program. Apache HTTP Client API library has been installed in our machine to extract HTTP header from response page. In test case generation phase, we generate 124 attack injection for SQL injection attack. Each parameter will be tested with 124 test cases. In this paper, we focus on the SQL injection vulnerability through input field such as login, search and registration field.

We test the framework with injecting attack test cases, and the results are summarized in Table 5. The response results will consider vulnerable if error messages and bypass authentication result are appeared in HTML document header. Based on testing results, we can conclude that all input forms are vulnerable to website. Response generator will analyze which input test cases have been generated to produce vulnerabilities.

Table 5. Vulnerability detection result

Web application	Parameter involved	HTTP method	No of test cases injected	# of vuln. detected
WackoPicko	Login	Post Get	496	20
	Password			
	Search			
	Register			
Mutillidae	Login	Post Get	248	9
	Password			

5 Conclusion

In this paper, we have proposed a method to generate test cases by using attack technique pattern with applying permutation algorithm to generate it automatically. This framework has been successfully tested and including several type of SQL injection vulnerabilities. Our framework is able to addresses the vulnerabilities based on results in Table 5. Future work will also be focused on extending our approach to cover other types of vulnerabilities such as Cross Site Scripting vulnerabilities.

Acknowledgment. This work was supported by the Advanced Informatics School (AIS), University Technology of Malaysia and National Defence University of Malaysia

References

1. Vermatt, S.: Discovering Computers 2009, Complete. Cengage Learning Course Technology (2009)
2. Anastacio, M., Blanco, J.A., Villalba, L., Dahoud, A.: E-Government: benefits, risks and a proposal to assessment including cloud computing and critical infrastructure. In: International Conference on Information Technology (2013)
3. Internet World Stats, Usage and Population Statistics (2013). http://www.internetworldstats.com/stats.htm
4. Symantec Corp.: Web Based Attacks (2013). http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/web_based_attacks_02-2009.pdf
5. Software Security Testing, Software Assurance Pocket Guide Series: Development, vol. III, Version 1.0, 21 May 2012
6. Gu, T.-Y., Shi, Y.-S., Fang, Y.-U.: Research on software security testing. World Academy of Science, Engineering and Technology **69**, 647–651 (2010)
7. Halfond, W.G.J., Choudhary, S.R., Orso, A.: Improving penetration testing through static and dynamic analysis. In: ICST 2009, the Second IEEE International Conference on Software Testing, Verification and Validation, vol. 21, pp. 195–214 (2011). doi:10.1002/stvr
8. Khan, S.A., Khan, R.A.: Software security testing process: phased approach. In: Agrawal, A., Tripathi, R.C., Do, E.Y.-L., Tiwari, M.D. (eds.) IITM 2013. CCIS, vol. 276, pp. 211–217. Springer, Heidelberg (2013)

9. Djuric, Z.: A black-box testing tool for detecting SQL injection vulnerabilities. In: 2013 2nd International Conference on Informatics and Applications, ICIA 2013, pp. 216–221 (2013). doi:10.1109/ICoIA.2013.6650259
10. Akrout, R., Alata, E., Kaaniche, M., Nicomette, V.: An automated black box approach for web vulnerability identification and attack scenario generation. J. Braz. Comput. Soc. **20**, 4 (2014). doi:10.1186/1678-4804-20-4
11. Awang, N.F., Manaf, A.A., Zainudin, W.S.: A survey on conducting vulnerability assessment in web-based application. In: Hassanien, A.E., Tolba, M.F., Taher Azar, A. (eds.) AMLTA 2014. CCIS, vol. 488, pp. 459–471. Springer, Heidelberg (2014)
12. Halfond, W.G.J., Halfond, W.G.J., Viegas, J., Viegas, J., Orso, A., Orso, A.: A classification of SQL injection attacks and countermeasures (2006)
13. Stuttard, D., Pinto, M.: The web application hacker's handbook: discovering and exploiting security flaws. Wiley Publishing, Inc., Indianapolis (2007)
14. Bisht, P., Madhusudan, P., Venkatarish-nan, V.N.: CANDID: dynamic candidate evaluations for automatic prevention of SQL injection attacks. ACM Trans. Inf. Syst. Secur. **13**(2), 1–39 (2010). Article 14
15. Ezumalai, R., Aghila, G.: Combinatorial approach for preventing SQL injection attacks. IEEE International Advance Computing Conference, IACC (2009)
16. Kindy, D.A., Pathan, A.S.K.: A detailed survey on various aspects of SQL injection in web applications: Vulnerabilities, innovative attacks and remedies. Int. J. Commun. Netw. Inf. Secur. **5**, 80–92 (2013)
17. Wodarz, P.N.: Algorithms for Generating Permutations and Combinations, pp. 1–7 (2008)
18. He, K., Feng, Z., Li, X.: An attack scenario based approach for software security testing at design stage. In: 2008 International Symposium on Computer Science and Computational Technology, pp. 782–787. IEEE Computer Society (2008)
19. Wassermann, G., Yu, D., Chander, A., Dhurjati, D., Inamura, H., Su, Z.: Dynamic test input generation for web applications. In: International Symposium on Software Testing and Analysis (ISSTA), pp. 249–259 (2008)
20. Alata, E., Kaaniche, M., Nicomette, V., Akrout, R.: An automated approach to generate web applications attack scenarios. In: Proceedings - 6th Latin-American Symposium on Dependable Computing, LADC 2013, pp. 78–85 (2013). doi:10.1109/LADC.2013.22
21. Bozic, J., Wotawa, F.: XSS pattern for attack modeling in testing. In: 2013 8th International Workshop on Automation of Software Test, AST 2013 - Proceedings, pp. 71–74 (2013). doi:10.1109/IWAST.2013.6595794
22. Bozic, J., Wotawa, F.: Security testing based on attack patterns. In: Proceedings - IEEE 7th International Conference on Software Testing, Verification and Validation Workshops, ICSTW 2014, pp. 4–11 (2014). doi:10.1109/ICSTW.2014.58
23. Chen, J.M., Wu, C.L.: An automated vulnerability scanner for injection attack based on injection point. In: ICS 2010 - International Computer Symposium, pp. 113–118 (2010). doi:10.1109/COMPSYM.2010.5685537
24. Duchene, F., Richier, J., Groz, R.: KameleonFuzz: Evolutionary Fuzzing for Black-Box XSS Detection. In: CODASPY (2014)