Atlantis Highlights in Computer Sciences, volume 4
Proceedings of the 3rd International Conference on Integrated Intelligent Computing Communication & Security (ICIIC 2021)

A Study of SQL Injection Hacking Techniques

Foong Yew Joe[1,*], Vinesha Selvarajah[2]

[1,2] *Asia Pacific University of Technology & Innovation (APU), Malaysia.*
[*] *Corresponding author. Email: Tp054538@mail.apu.edu.my*

ABSTRACT
Data is the most valuable asset of a person in the current cyber world. More and more data are being collected by applications for multi-purposes. These valuable data are stored inside a database. Standard Query Language (SQL) is a database query language for managing databases. SQL injection attack is the most common attack being used by attackers to gain unauthorized access to the database although it has been used for more than a decade. Many security professionals have proposed countermeasures against SQL injection attacks, but it is still listed as one of the Top 10 Web Application Security Risks today. The concept of SQL injection attack is to inject SQL codes into the database server and execute the injected codes to retrieve the desired result. SQL injection attacks can be classified into different categories depending on the characteristics of the attack. The severity of a SQL injection attack may vary, depending on the vulnerability and the permission assigned. It may only be causing leakage of some insensitive data or it might be causing the destruction and major modification of the database. This paper includes an overview of SQL injection attacks and a demonstration of attacking the database. Moreover, the characteristics and examples of exploiting different types of SQL injection vulnerabilities were discussed.

Keywords: *SQL Injection, SQL, Hacking, Cyber Security, Penetration Testing, Database.*

1. INTRODUCTION

Today, data is the most valuable asset of a business. Businesses collect users' data and make use of it. Purchase records, browsing records, online duration, and every other data are being collected by companies [23-27]. Many applications including Facebook, Wechat, and others are collecting data not only from their application, but their trackers also track the users' activity on other applications such as Google Chrome and send it back to their server. In our daily life, it is noticeable that after searching for something with Google, other applications like Facebook and Google Chrome will pop up a related product advertisement. That is the power of data in advertising. Companies can target their clients more accurately with data analysis [28-30].

It is important to manage the database well with so much data collected. Standard Query Language (SQL) is a database query language used in a relational database management system (RDBMS) for storing, manipulating, and retrieving data [1]. A relational database management system is designed for storing data in a structured format, using rows and columns [2]. SQL is one of the most used and powerful languages for managing databases.

SQL injection is an attack by inserting SQL code into application input parameters that are passed back to the back-end SQL server [3]. An attacker can inject SQL code to manipulate or retrieve data from the database. The database often contains sensitive information of users like passwords and usernames. SQL injection vulnerabilities have been described as one of the most serious threats to web applications [4]. Also, SQL injection was listed as the latest top 10 web application vulnerability by the Open Web Application Security Project (OWASP) [5] [30-34].

SQL injection vulnerability occurs due to improper validation of input from the user passed to the back-end SQL query. Characters like quotation marks, semicolons, number signs, and so on should be restricted because they could be used to attack the database and retrieve unauthorized data.

2. CLASSIFICATION OF SQL INJECTION

SQL injection attacks are mainly classified into three categories, which are Order wise, Blind, and Against database.

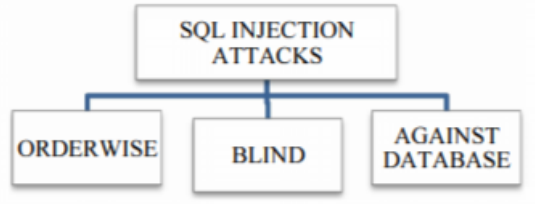

Figure 1 Classification of SQL Injection Attacks [6]

2.1. Order Wise SQL Injection Attack

In order wise category, it contains first order injection attack, second-order injection attack, and lateral injection attack. A first-order injection attack is where the attacker gets the desired results directly from the response of the application they are interacting with or other response mechanisms [7]. The attacker could get unlimited and unauthorized access to the database. For example, when an attacker injects Figure 2 SQL code, the database will return information of all users whose names start with 'harry'. Besides, the attacker could also inject "OR 1=1 –" to retrieve all data from a table.

```
SELECT email, password, login-id FROM members
WHERE email= ' ' OR fullname_LIKE '%harry%';
```

Figure 2 First-order SQL injection code [6]

A second-order injection attack is an attacker who does not get the result immediately. The malicious code is stored in the database instead of being executed immediately by the application. The attacker gets the data when the injected code is utilized by the application [8].

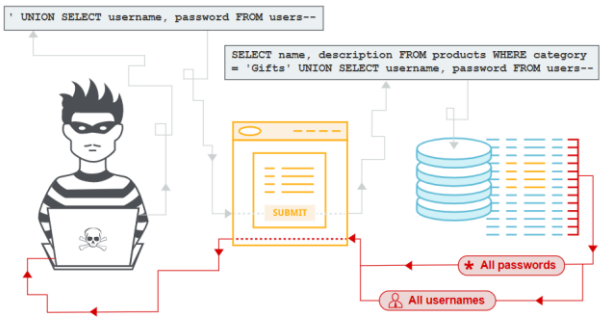

Figure 3 Second-order injection attack flow [9]

Moreover, a Lateral injection attack is the attacker exploits a PL/SQL procedure that does not take input from users. This type of attack is not common. For example, when the system wants to get the system date from Oracle, the syntax is "select sysdate from dual;". The sysdate format is NLS_Date_Format. The attacker can alter NLS_Date_Format with malicious code to get the database data when the system calls the sysdate [6].

2.2. Blind SQL Injection Attack

When a web application is vulnerable to SQL injection, the attacker changes the input of parameters, and the web application will display an error message as the back-end SQL server has an error when executing the malicious SQL code injected. In some cases, the developer hides the details of the error message and only displays a generic error page. This increased the difficulty of exploitation, but it is still possible for the attacker to exploit. The attacker can try a list of true/false statements until they find the right SQL command to retrieve data from the database [6].

Boolean-based blind SQL injection attack is the attacker tries a series of true/false queries to the web application and analyzes the responses from the application to determine if it is vulnerable to SQL injection and then proceed to extract data. For example, a web page is displaying details of a particular item, and the URL of the webpage takes a parameter of that item's id as in Figure 4. The attacker could try adding a true or false statement at the parameter like Figure 5. Since the statement is false, if the webpage displays no items on the page, it indicates that the website is vulnerable to a Boolean-based SQL injection attack [10].

```
http://www.shop.local/item.php?id=34
```

Figure 4 URL of a webpage [10]

```
http://www.shop.local/item.php?id=34 and 1=2
```

Figure 5 Adding statement to the parameter [10]

If the database errors are handled well, the attacker will not receive a different response by injecting different SQL queries [11]. In this case, the attacker tries to make a time-intensive operation on the database. If the database response follows the injected time operation, the webpage is vulnerable. One of the most popular time operations is sleep. For example, the attack could enumerate each letter of the desired piece of data by injecting code to let the response wait for N seconds if the first database's name letter is "A" [12]. Following this logic, the attacker could enumerate the full database name and continue to find more data.

2.3. Against Database SQL Injection Attack

There are four sub-categories of SQL against database attacks - SQL Manipulation, Code Injection, Function Call, and Buffer Overflow.

2.3.1. SQL Manipulation

SQL manipulation is the process of modifying the SQL statement with operations such as UNION or changing the WHERE clause to get a different result [13].

Tautology is a type of attack that falls under SQL manipulation. The goal of a tautology SQL attack is to inject one or more conditional statements so that it always evaluates to true. It is used to bypass authentication and retrieve data. For example, the back-end SQL server authenticates the username and password with "SELECT name FROM users WHERE username='$_POST[username]' AND password='$_POST[password]';". In this case, it will return true only when the user has entered the correct username and password at the login page.

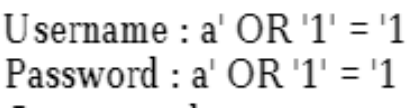

Figure 5 Tautology SQL injection attack [14]

If the user enters the username and password in Figure 5, the query will look like "SELECT name FROM users WHERE username='a' OR '1'='1' AND password='a' OR '1'='1';" in the back-end SQL server. It will return all data in the table because the WHERE clause always returns True. Now the attacker can get all usernames and passwords from the database [14].

Besides, a logically incorrect query is also an attack that falls under SQL manipulation. This type of attack takes advantage of the error message that is returned to the application by the database server. For example, an attacker inserts " ' HAVING 1='1';-- " in a login page as it takes input parameters from users. The server could respond to an error message like Figure 6 if the error message is not managed. Now the attacker gets the table name and column name of the database and proceeds to the extraction of data [13].

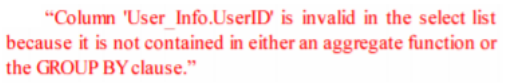

Figure 6 Error message from the database server [13]

Other than that, a Union query is an attack representing SQL manipulation and Code Injection. A UNION operator is used to combine two or more queries. The attacker can use a UNION query to add another SELECT statement for retrieving data from the table. Besides, the Union query also can be used to discover other table information like column number, table name, and column datatype [15].

2.3.2. Code Injection

The process of a new SQL statement being inserted into the SQL server is called a code injection attack. This type of attack requires the server to support multiple SQL statements per database request [13].

Piggy-backed queries attack is the practice of appending or manipulating unchecked values to web-based SQL queries [16]. The attacker could change records in the table with UPDATE, DELETE, INSERT operations. For example, the attacker could close the quotation mark of the query to break out from the strings and add another query to manipulate the database. For example, the application shows the SQL parameter in the URL like "http://test.com/presslist.asp?id=1", and in the server, the back-end query could look like " SELECT * FROM User_info WHERE User_ID = 1". The attacker could change the value "1" in the URL into " ' '; DELETE FROM User_info". If it has a high privilege, the table User_info will be deleted.

Inference-based attacks also fall under the category of Code Injection. Previously discussed Blind injection attacks including Boolean-based and timing attacks are under Inference based attacks. This type of attack is to observe the server's responses.

2.3.3. Function Call

A function call attack is the insertion of the database built-in function into a SQL statement. The attacker calls the system function to manipulate the database. If the attacker could inject SQL strings successfully, the system stored procedures could be exploited by the attacker. However, if the application user privilege on the database is not high, it may not be able to call the system stored procedures. Anyway, mostly the result of successful execution of calling the system procedure will not be shown in the response to the user [7].

2.3.4. Buffer Overflow

Buffer Overflow vulnerability has been identified in several databases. If the database is un-patched and not up to date, it might be exposed to a buffer overflow attack. Since most of the applications and web servers were unable to handle loss connection to the database, buffer overflow attacks can be exploited to achieve a denial-of-service attack as the application is not responding without the database connection [7].

Furthermore, the injection of SQL query could be encoded to escape the input filter as some developers filter the user input that contains dangerous characters like UNION, ";", and so on. Encodings combined with other attack techniques could evade the security mechanism and exploit the database.

3. THE SCENARIO OF SQL INJECTION ATTACK

To find a SQL injection vulnerability, the attacker will investigate all parameters within the web application, either in the URL or other HTML input types. Now assume that there is a vulnerable site taking an ID parameter in the URL. This is the URL of the vulnerable site https://example.com/index.php?id=1. The attacker could enter other characters and numbers into the "id" parameter to find out whether it is exploitable. The back-end SQL query of the web application could look like "SELECT * FROM users WHERE id = '1';".

If the attacker attempts to manipulate the ID parameter into "?id=' or 1=1". The web application could return all data of the users' table if there is no limit implemented because the WHERE clause is always true. The attacker could go even further for retrieving other tables' data with the UNION clause. The attacker could use a UNION clause to retrieve all information of the database including the tables' names and columns' names, then retrieve the data inside the table's column. A UNION-based attack will be demonstrated later. Other than accessing unauthorized areas, the attacker could drop the tables using DROP TABLE or manipulate the data inside the table.

4. IMPACT OF SQL INJECTION ATTACK

Although SQL injection is one of the oldest exploits, it is still a serious threat nowadays because some developers did not implement a good security mechanism against it. According to Entrust Solutions, SQL injection attacks constitute 65% of web-based attacks between 2017 and 2019 [17]. The attack could gain control over the web database through SQL injection. It could cause data breaches, or destruction of the data system depends on the attacker's action.

SQL injection attacks not only affect the database data and the availability of service, but they may also lead to reputation damage and being fined too. The General Data Protection Regulation (GDPR) in Europe requires organizations to be responsible for the personal information stored in their databases. It applies to an organization that has European citizen information within its database. According to Fierce IT Security, most of the data breaches are the result of SQL injection attacks [18]. An organization could be fined 20 million Euros or even more with the GDPR, so the organization should implement a full security mechanism to prevent SQL injection attacks.

5. THE FUNCTIONALITY OF SQL INJECTION ATTACK

SQL injection could be used to satisfy the following intentions:

1. Identifying the injectable parameters – the attacker could try different queries on the URL to find the vulnerability.
2. Identifying the database fingerprint – the attacker identifies the type and version of the database as different databases may have different queries and attacks.
3. Discovering the database schema – database schema is important as it is the structure of the database system. Database schema contains table names, column names, and column data type that is needed for the attack to modify or retrieve data from the database.
4. Extracting data – the attacker extracts data from the database. Some databases may contain sensitive information like passwords and credit cards that are highly desirable to the attacker [4].
5. Modifying data – the attacker alters the data in a database into their desired data.
6. Bypassing authentication – the attacker bypasses the authentication mechanism of the database and application to get unauthorized access and privileges.
7. Performing denial-of-service – the attacker could perform destruction actions on the database like deleting the table, thus other users could not access the service.
8. Escalating privileges – the attacker may escalate their privileges, so they can access unauthorized content and services [19].
9. Executing remote commands – executes arbitrary commands on the database.

6. DEMONSTRATION OF SQL INJECTION ATTACK

A demonstration of SQL injection attack will be demonstrated in a legal environment. Damn Vulnerable Web Application (DVWA) is used for demonstrating the attack instead of using an actual public web application. As an ethical hacker, always obey the law and do not attempt to hack any organization unless it is authorized by the owner. The DVWA security setting is set to High for a more secured website, but with some improper practice in the security implemented environment.

A step-by-step tutorial of retrieving sensitive data from the database is demonstrated.

1. Open DVWA
2. There is a "click here to change your ID" selection on the page. A pop-up window is appeared and asked for input for the new ID number to retrieve the information related to that

ID number. Now Enter "2" into it and see what it retrieves.

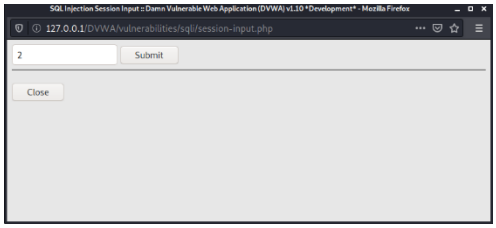

Figure 7 Pop up window for entering a new ID

3. The first name and surname of ID number 2 appear under the change ID selection.

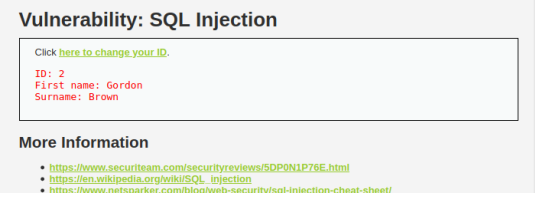

Figure 8 Information of that ID number

4. Now try entering an ID of "3-1" to check if it has a string quotation mark inside the query or not. The result of ID number is 3-1, indicating the input is inside a quotation mark in the back-end query.

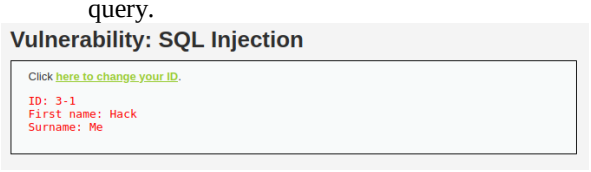

Figure 9 Test the SQL query

5. Since it has a quotation mark in the query to treat all the input as strings, now enter a quotation mark as the ID number and check whether the application has input restriction implemented. The page shows "something went wrong" and no other information is shown. This means the SQL query has an error occurred, but the web application has handled the error message and responds a general error message back to the client instead of the error details.

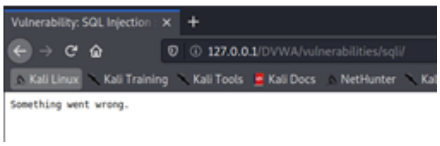

Figure 10 Test the Input Validation

6. Try adding a comment delimiter to ignore all SQL commands behind the input in the query. The ID is changed to " ' # " and now the error is not showing up and it is back to the original page. No information of that ID is shown because the ID parameter in the query is empty.

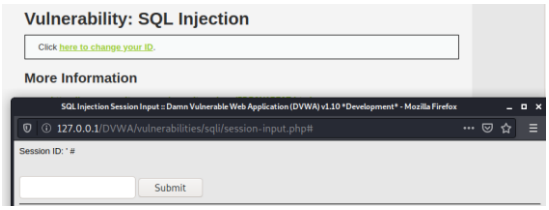

Figure 11 Result of adding the comment delimiter

7. Add a true statement in the query to retrieve all information in the table. It only shows the first name and surname because the back-end query is probably something like "SELECT firstname, surname FROM xxx……". Anyway, it is known that there are only 5 people recorded in the database.

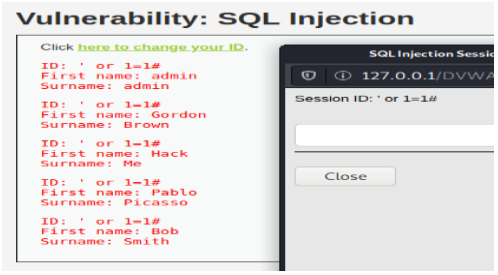

Figure 12 Retrieve all information since the query is True

8. To retrieve more information in the database, the Union query is used to select more information to show up from the database. A Union query must have the same number of columns retrieved as the original query. To test the number of columns retrieved in the query, the ORDER BY clause is used for testing, starting from 1. The page has an error showing up when testing the " ' ORDER BY 3 #". Thus, there are two columns retrieved in the original query.

Figure 13 Testing number of columns retrieved in back-end query

9. After getting the number of columns retrieved in the query, the next step is to test the column's data type. But since in Figure 5.7, the first name and surname data is a string, so it is not needed to test the data type anymore as a string can hold any characters and numbers.
10. Now the name of the database can be retrieved with " ' UNION SELECT 1, database()# ". Database() is a built-in function to retrieve the database name. The result returned shows that the database name is dvwa.

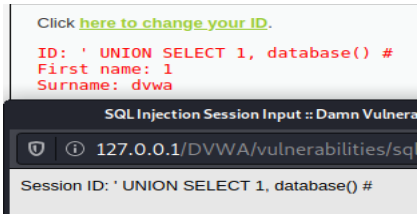

Figure 14 Retrieve the database name

11. After discovering the database name, the next step is to retrieve the table names inside dvwa database with " ' UNION SELECT 1, table_name FROM information_schema.tables WHERE table_schema='dvwa' ", and the result showed that there are two tables inside the dvwa database, which are the guestbook and users.

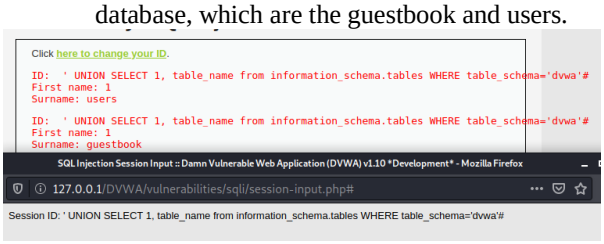

Figure 15 Retrieve all tables in the dvwa database

12. There are two tables in the database based on the result returned. The command " ' UNION SELECT column_name,column_type from information_schema.columns WHERE table_name='users' AND table_schema='dvwa'#" is entered to check all columns inside the 'users' table and determine which column to retrieve.

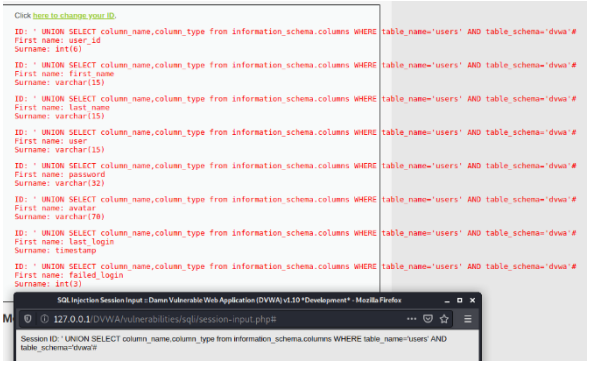

Figure 16 Retrieve all columns inside the 'users' table

13. The result showed that the 'users' table contains user_id, first_name, last_name, user, password, avatar, last_login, and failed_login columns. Now the data in these columns can be retrieved. The user and password columns probably containing the username and password of that user, so these two columns are selected to check if it is the desired information.
14. By entering " ' UNION SELECT user, password FROM users#", the username and password are returned. The username and password of all five users are leaked to an unauthorized user now. The password stored is hashed.

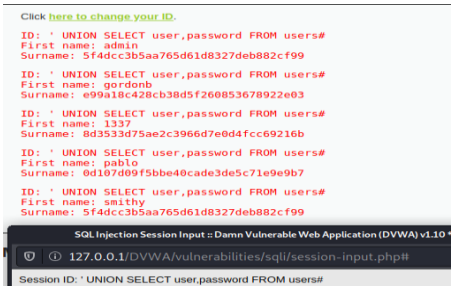

Figure 17 Username and Password of all users

7. CRITICAL ANALYSIS

In 5. Step by step exploiting the SQL injection vulnerability, the process of injecting SQL code into the database to retrieve sensitive information was demonstrated. The desired data could not be returned directly, but it can be returned after enumerating the database piece by piece, the database name, tables inside, and columns. After discovering all this information, it is ready to get any stored data within the database. In the demonstration, the username and password are retrieved. But in other applications, it may store more sensitive personal information like credit card number, home address, phone number, et cetera. If the attacker had successfully exploited a SQL injection vulnerability, it might cause a lot of trouble, especially when the database contains sensitive information.

In the demonstration, the SQL codes are injected manually to test the vulnerability and exploitation. There are some powerful tools available that could automate this process such as SQLmap. It is an open-source tool for detecting and exploiting SQL injection flaws for many variations of database servers such as MySQL, PostgreSQL, MariaDB, and many more [20]. If the SQLmap tool is used in the demonstration, the attack process could be shortened to 1-2 minutes. However, using automated tools is not a good practice for beginners as it automates all the processes, the concepts and techniques behind it are unknown.

In the demonstration web application, the vulnerability of SQL injection is due to no input validation. Users could enter any characters and symbols they like into it. So, the attacker could enter some SQL functional symbols like comment delimiter and quotation mark for adding or changing the query to attack the database. For example, close the quotation mark and use UNION to retrieve other data in the database.

There are a few countermeasures to SQL injection. The first is to use parameterized queries. The concept of parameterized queries is the SQL codes are defined before passing each parameter to the query. This allows the database to distinguish between data and code [21]. In this case, the code injected in the demonstration will

not break out from the ID parameter strings, it will treat all the input as strings and search for that string in ID because the input is defined as data. However, this countermeasure could affect the performance in rare circumstances. Figure 18 is an example of using parameterized queries with Java. The user input is not directly passed back to the query, it used the "PreparedStatement" function to bind variables, and the input is treated as string data.

```
// This should REALLY be validated too
String custname = request.getParameter("customerName");
// Perform input validation to detect attacks
String query = "SELECT account_balance FROM user_data WHERE user_name = ? ";
PreparedStatement pstmt = connection.prepareStatement( query );
pstmt.setString( 1, custname);
ResultSet results = pstmt.executeQuery( );
```

Figure 18 Example of using Java parameterized queries [21]

Besides that, another countermeasure is implementing the input validation. The input validation will check whether the user input is within the accepted length, format, pattern, and so on before passing it to the query [22]. If the user input does not match the validation rules, it will generate an error. For example, the change ID input in our demonstration could set an input validation to allow only numeric values. If the input contains characters or symbols, it will not pass the validation, causing an error to be returned. With this, the attacker could not put SQL commands and symbols to attack the database.

Other than that, escaping user input is also a countermeasure to SQL injection. This technique is to escape the specified characters in the user input before passing them to the query. For example, if escaping character rules include the number sign (#) and single quotation mark (‘), the SQL code injected in the demonstration will not work because the commend delimiter and the single quotation mark to break out from the string are escaped. Each database management system supports one or more-character escape schemes. If the proper schemes are used, it could avoid any possible SQL injection vulnerabilities [21].

SQL injection has been one of the most used and severest vulnerable due to improper security mechanisms implemented by the developer. If the countermeasures are implemented well by the developer, SQL injection flaws could be avoided and increase the safety and privacy of the database.

8. CONCLUSION

The existence of SQL injection vulnerability is still high currently, many attackers are taking advantage of this vulnerability. Actions like modifying and deleting data and shutting down the database could be performed. Although developers have implemented the security countermeasures, some bad practices in the code could leave a flaw in the system, causing the system to be vulnerable to SQL injection attacks. Databases should be protected well as much information is stored.

There are many tools available for scanning SQL injection vulnerability. Developers could utilize these tools to scan the application. Applications on the Internet can never be immune to being attacked, but the risk of being successfully attacked by hackers can be lowered with proper security countermeasures implemented, decreasing the loss and occurrence of an attack.

REFERENCES

[1] R. Kumar, N. Gupta, S. Charu, S. Bansal and K. Yadav, "Comparison of SQL with HiveQL," International Journal for Research in Technological Studies, vol. 1, no. 9, pp. 28-30, 2014.

[2] P. Christensson, "RDBMS," 2017. [Online]. Available: https://techterms.com/definition/rdbms#:~:text=Stands%20for%20%22Relational%20Database%20Management,format%2C%20using%20rows%20and%20columns.. [Accessed 29 January 2021].

[3] J. Clarke-Salt, SQL Injection Attacks and Defense, Elsevier, 2009.

[4] W. G. Halfond, J. Viegas and A. Orso, "A Classification of SQL Injection Attacks," Proceedings of the IEEE international symposium on secure software engineering, vol. 1, pp. 13-15, 2006.

[5] OWASP, "OWASP Top Ten," 2020. [Online]. Available: https://owasp.org/www-project-top-ten/. [Accessed 31 January 2021].

[6] C. Sharma and D. S. Jain, "Analysis and Classification of SQL Injection Vulnerabilities and Attacks on Web Applications," Unnao, 2014.

[7] K. Ahmad, J. Shekhar and K. Yadav, "Classification of SQL Injection Attacks," VSRD Technical & Non-Technical Journal, vol. 1, no. 4, pp. 235-242, 2010.

[8] F. Cookie, "What is a Second-Order SQL Injection and how can you exploit it successfully?," 2019. [Online]. Available: https://medium.com/bugbountywriteup/the-wrath-of-second-order-sql-injection-c9338a51c6d. [Accessed 31 January 2021].

[9] PortSwigger, "SQL injection," n.d.. [Online]. Available: https://portswigger.net/web-security/sql-injection. [Accessed 31 January 2021].

[10] Acunetix, "What Are Blind SQL Injections," n.d.. [Online]. Available: https://www.acunetix.com/websitesecurity/blind-sql-injection/. [Accessed 31 January 2021].

[11] R. Singh, "How Blind SQL Injection Works?," 2020. [Online]. Available: https://www.indusface.com/blog/how-blind-sql-injection-works/. [Accessed 31 January 2021].

[12] OWASP, "Blind SQL Injection," n.d.. [Online]. Available: https://owasp.org/www-community/attacks/Blind_SQL_Injection#:~:text=Description,based%20on%20the%20applications%20response.&text=This%20makes%20exploiting%20the%20SQL,difficult%2C%20but%20not%20impossible.%20.. [Accessed 31 January 2021].

[13] P. Kumar and R. Pateriya, "A Survey on SQL Injection Attacks, Detection and Prevention Techniques," Coimbatore, India, 2012.

[14] K. Jhala and S. Umang D, "Tautology based Advanced SQL Injection Technique A Peril to Web Application," Ahmedabad, India, 2017.

[15] IndominusByte, "SQL injection UNION attack," 2019. [Online]. Available: https://medium.com/@nyomanpradipta120/sql-injection-union-attack-9c10de1a5635. [Accessed 31 January 2021].

[16] M. Borland, "Advanced SQL Command Injection: Applying defense-in-depth practices in web-enabled database applications," 2002. [Online]. Available: http://www.palecrow.com/content/GCIH/Matt_Borland_GCIH.html#:~:text=SQL%20piggybacking%2C%20or%20SQL%20command,material%20entitled%20'Web%20Application%20Attacks.. [Accessed 31 January 2021].

[17] E. Solutions, "4 TYPES OF DATA BREACHES TO KNOW IN 2020," 2020. [Online]. Available: https://www.entrustsolutions.com/2020/09/02/4-types-of-data-breaches-2020/#:~:text=In%20an%20SQL%20injection%20attack,location%20of%20the%20hacker's%20choosing.. [Accessed 1 February 2021].

[18] TeskaLabs, "SQL Injection - Are Developers to Blame for Data Security Breaches?," n.d.. [Online]. Available: https://teskalabs.com/blog/sql-injection-are-developer-to-blame-for-data-security-breach. [Accessed 1 February 2021].

[19] S.-T. Sun, T. Han Wei, S. Liu and S. Lau, "Classification of SQL Injection Attacks," 17 November 2007. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.620.9731&rep=rep1&type=pdf. [Accessed 31 January 2021].

[20] sqlmap, "Introduction," n.d.. [Online]. Available: http://sqlmap.org/. [Accessed 12 February 2021].

[21] OWASP, "SQL Injection Prevention Cheat Sheet," n.d.. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html. [Accessed 12 February 2021].

[22] Positive Technologies, "How to prevent SQL injection attacks," 2019. [Online]. Available: https://www.ptsecurity.com/ww-en/analytics/knowledge-base/how-to-prevent-sql-injection-attacks/#3. [Accessed 12 February 2021].

[23] Naeem, M.A., Nguyen, T.N., Ali, R., Cengiz, K., Meng, Y. and Khurshaid, T., 2021. Hybrid Cache Management in IoT-based Named Data Networking. IEEE Internet of Things Journal.

[24] Nguyen, T.N., Liu, B.H. and Wang, S.Y., 2019. On new approaches of maximum weighted target coverage and sensor connectivity: Hardness and approximation. IEEE Transactions on Network Science and Engineering, 7(3), pp.1736-1751.

[25] Dash, R.K., Nguyen, T.N., Cengiz, K. and Sharma, A., 2021. Fine-tuned support vector regression model for stock predictions. Neural Computing and Applications, pp.1-15.

[26] Subramani, P., Al-Turjman, F., Kumar, R., Kannan, A. and Loganthan, A., 2021. Improving medical communication process using recurrent networks and wearable antenna s11 variation with harmonic suppressions. Personal and Ubiquitous Computing, pp.1-13.

[27] Subramani, P., Srinivas, K., Sujatha, R. and Parameshachari, B.D., 2021. Prediction of muscular paralysis disease based on hybrid feature extraction with machine learning technique for COVID-19 and post-COVID-19 patients. Personal and Ubiquitous Computing, pp.1-14.

[28] Puttamadappa, C. and Parameshachari, B.D., 2019. Demand side management of small scale loads in a smart grid using glow-worm swarm optimization technique. Microprocessors and Microsystems, 71, p.102886.

[29] Rajendran, G.B., Kumarasamy, U.M., Zarro, C., Divakarachari, P.B. and Ullo, S.L., 2020. Land-use and land-cover classification using a human group-based particle swarm optimization algorithm with an LSTM Classifier on hybrid pre-processing remote-sensing images. Remote Sensing, 12(24), p.4135.

[30] Parameshachari, B.D., 2021, March. Logistic Sine Map (LSM) Based Partial Image Encryption. In 2021 National Computing Colleges Conference (NCCC) (pp. 1-6). IEEE.

[31] F. Ding, G. Zhu, M. Alazab, X. Li, and K. Yu, “Deep-Learning-Empowered Digital Forensics for Edge Consumer Electronics in 5G HetNets”, IEEE Consumer Electronics Magazine, doi: 10.1109/MCE.2020.3047606.

[32] C. Feng, K. Yu, M. Aloqaily, M. Alazab, Z. Lv and S. Mumtaz, "Attribute-Based Encryption with Parallel Outsourced Decryption for Edge Intelligent IoV," IEEE Transactions on Vehicular Technology, vol. 69, no. 11, pp. 13784-13795, Nov. 2020, doi: 10.1109/TVT.2020.3027568.

[33] Y. Sun, J. Liu, K. Yu, M. Alazab, K. Lin, “PMRSS: Privacy-preserving Medical Record Searching Scheme for Intelligent Diagnosis in IoT Healthcare”, IEEE Transactions on Industrial Informatics, doi: 10.1109/TII.2021.3070544.

[34] K. Yu, L. Tan, X. Shang, J. Huang, G. Srivastava and P. Chatterjee, “Efficient and Privacy-Preserving Medical Research Support Platform Against COVID-19: A Blockchain-Based Approach”, IEEE Consumer Electronics Magazine, doi: 10.1109/MCE.2020.3035520.