Volume 6, Issue 3, March 2016 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Analysis of SQL Injections Attacks and Vulnerabilities

Dr. Amit Chaturvedi*
Assistant Prof., MCA Deptt.,
Govt. Engineering College, Ajmer,
Rajasthan, India

Shailendra Bagdi
M.Tech. Scholar
Bhagwant University, Ajmer,
Rajasthan, India

Vikas Choudhary
Assistant Prof., CSE Deptt.,
Bhagwant University, Ajmer,
Rajasthan, India

Abstract: SQL is a code injection technique, used to attack data-driven applications, in which malicious SQL statements are inserted into an entry field for execution (e.g. to dump the database contents to the attacker). SQL injection must exploit a security vulnerability in an application's software, for example, when user input is either incorrectly filtered for string literal escape characters embedded in SQL statements or user input is not strongly typed and unexpectedly executed. In this paper, we have presented the SQL Injection in detail with its various types. This analysis will help the readers to understand the SQL Injection attacks.

Keywords: SQLi, attack, application server, website, authentication.

I. INTRODUCTION

SQL injection (SQLi) refers to an injection attack wherein an attacker can execute malicious SQL statements that control a web application's database server. Since an SQL injection vulnerability could possibly affect any website or web application that makes use of an SQL-based database, the vulnerability is one of the oldest, most prevalent and most dangerous of web application vulnerabilities.

By leveraging SQL injection vulnerability, given the right circumstances, an attacker can use it to bypass a web application's authentication and authorization mechanism and retrieve the contents of an entire database. SQL injection can also be used to add, modify and delete records in a database, affecting data integrity [1].

To such an extent, SQL injection can provide an attacker with unauthorized access to sensitive data including, customer data, personally identifiable information (PII), trade secrets, intellectual property and other sensitive information.

Keeping the above in mind, when considering the following, it's easier to understand how lucrative a successful SQL injection attack can be for an attacker.

- An attacker can use SQL injection to bypass authentication or even impersonate specific users.
- One of SQL's primary functions is to select data based on a query and output the result of that query. SQL injection vulnerability could allow the complete disclosure of data residing on a database server.
- Since web applications use SQL to alter data within a database, an attacker could use SQL injection to alter data stored in a database. Altering data affects data integrity and could cause repudiation issues, for instance, issues such as voiding transactions, altering balances and other records.
- SQL is used to delete records from a database. An attacker could use an SQL injection vulnerability to delete data from a database. Even if an appropriate backup strategy is employed, deletion of data could affect an application's availability until the database is restored.
- Some database servers are configured (intentional or otherwise) to allow arbitrary execution of operating system commands on the database server. Given the right conditions, an attacker could use SQL injection as the initial vector in an attack of an internal network that sits behind a firewall.
- Structure query language is most important language that executes data bases. SQL is defined in three main category commands are: DDL (Data Definition Language) DML (Data Manipulation Language) and DCL (Data Control Language)[2,3,4]

II. SQL INJECTION

SQL is a code injection technique, used to attack data-driven applications, in which malicious SQL statements are inserted into an entry field for execution (e.g. to dump the database contents to the attacker). SQL injection must exploit a security vulnerability in an application's software, for example, when user input is either incorrectly filtered for string literal escape characters embedded in SQL statements or user input is not strongly typed and unexpectedly executed. SQL injection is mostly known as an attack vector for websites but can be used to attack any type of SQL database.SQL injection attacks allow attackers to spoof identity, tamper with existing data, cause repudiation issues such as voiding transactions or changing balances, allow the complete disclosure of all data on the system, destroy the data or make it otherwise unavailable, and become administrators of the database server.

SQL injection (SQLI) is considered one of the top 10 web application vulnerabilities of 2007 and 2010 by the Open Web Application Security Project. In 2013, SQLI was rated the number one attack on the OWASP top ten. There are four main sub-classes of SQL injection: Classic SQLI, Blind or Inference SQL injection, Database-specific SQLI,

Compounded SQLI,SQL injection + insufficient authentication ,SQL injection + DDoS attacks ,SQL injection + DNS hijacking, SQL injection + XSS. The Storm Worm is one representation of Compounded SQLI. This classification represents the state of SQLI, respecting its evolution until 2010—further refinement is underway[5,6].

III. INCORRECTLY FILTERED ESCAPE CHARACTERS

This form of SQL injection occurs when user input is not filtered for escape characters and is then passed into an SQL statement. This result in the potential manipulation of the statements performed on the database by the end-user of the application .The following line of code illustrates this vulnerability:

Statement = "SELECT * FROM users WHERE name = '" + username + "';"

This SQL code is designed to pull up the records of the specified username from its table of users. However, if the "username" variable is crafted in a specific way by a malicious user, the SQL statement may do more than the code author intended. For example, setting the "userName" variable as:

‘OR '1'='1

or using comments to even block the rest of the query (there are three types of SQL comments). All three lines have a space at the end:

‘OR '1'='1' --

‘OR '1'='1' ({

‘OR '1'='1' /*

Renders one of the following SQL statements by the parent language:

SELECT * FROM users WHERE name = '' OR '1'='1';

SELECT * FROM users WHERE name = '' OR '1'='1' -- ';

If this code were to be used in an authentication procedure then this example could be used to force the selection of every data field (*) from all users rather than from one specific user name as the coder intended, because the evaluation of '1'='1' is always true.

The following value of "username" in the statement below would cause the deletion of the "users" table as well as the selection of all data from the "user info" table (in essence revealing the information of every user), using an API that allows multiple statements:

a’; DROP TABLE users; SELECT * FROM user info WHERE 't' = 't

This input renders the final SQL statement as follows and specified:

SELECT * FROM users WHERE name = 'a’; DROP TABLE users; SELECT * FROM user info WHERE 't' = 't';

While most SQL server implementations allow multiple statements to be executed with one call in this way, some SQL APIs such as PHP's mysql_query() function do not allow this for security reasons. This prevents attackers from injecting entirely separate queries, but doesn't stop them from modifying queries[7,8,9].

IV. BLIND SQL INJECTION

Blind SQL Injection is used when a web application is vulnerable to an SQL injection but the results of the injection are not visible to the attacker. The page with the vulnerability may not be one that displays data but will display differently depending on the results of a logical statement injected into the legitimate SQL statement called for that page. This type of attack can become time-intensive because a new statement must be crafted for each bit recovered. There are several tools that can automate these attacks once the location of the vulnerability and the target information has been established. Conditional responses, one type of blind SQL injection force the database to evaluate a logical statement on an ordinary application screen. As an example, a book review website uses a query string to determine which book review to display. So the URL http://books.example.com/showReview.php?ID=5 would cause the server to run the query

SELECT * FROM book reviews WHERE ID = ‘Value (ID)';

from which it would populate the review page with data from the review with ID 5, stored in the table book reviews. The query happens completely on the server; the user does not know the names of the database, table, or fields, nor does the user know the query string. The user only sees that the above URL returns a book review. A hacker can load the URLs http://books.example.com/showReview.php?ID=5 OR 1=1 and http://books.example.com/showReview.php?ID=5 AND 1=2, which may result in queries

SELECT * FROM book reviews WHERE ID = '5' OR '1'='1';

SELECT * FROM book reviews WHERE ID = '5' AND '1'='2';

Respectively. If the original review loads with the "1=1" URL and a blank or error page is returned from the "1=2" URL, and the returned page has not been created to alert the user the input is invalid, or in other words, has been caught by an input test script, the site is likely vulnerable to a SQL injection attack as the query will likely have passed through successfully in both cases. The hacker may proceed with this query string designed to reveal the version number of MySQL running on the server: http://books.example.com/showReview.php?ID=5 AND substring (@@version, 1, INSTR(@@version, '.') - 1)=4, which would show the book review on a server running MySQL 4 and a blank or error page otherwise. The hacker can continue to use code within query strings to glean more information from the server until another avenue of attack is discovered or his or her goals are achieved[10,11].

V. SECOND ORDER SQL INJECTION

Second order SQL injection occurs when submitted values contain malicious commands that are stored rather than executed immediately. In some cases, the application may correctly encode an SQL statement and store it as valid SQL.

Then, another part of that application without controls to protect against SQL injection might execute that stored SQL statement. This attack requires more knowledge of how submitted values are later used. Automated web application security scanners would not easily detect this type of SQL injection and may need to be manually instructed where to check for evidence that it is being attempted.

Mitigation: An SQL injection is a well known attack and easily prevented by simple measures. After an apparent SQL injection attack on Talk, security experts were stunned that such a large company would be vulnerable to it[12].

VI. DIFFERENT TYPES OF SQL INJECTIONS?

SQL injections can be classified and categorized in different ways, based on the type of data extraction channel, the response received from server, how server responses aid in leveraging the successful exploitation, impact point, etc.

Based on the data extraction channel

- In band or inline
- Out-of-band

SQL injections that use the same communication channel as input to dump the information back are called in-band or inline SQL Injections. This is one of the most common methods, readily explained on the Internet in different posts. For example, a query parameter, if inject-able, leads to the dumping of info on the web page. Injections that use a secondary or different communication channel to dump the output of queries performed via the input channel are referred to as out-of-band SQL injections. For example, the injection is made to a web application and a secondary channel such as DNS queries is used to dump the data back to the attacker domain.

Based on the response received from the server Error-based SQL injections

- Union query type.
- Double query Injections.

Blind SQL Injections

- Boolean-based blind injections.
- Time based blind injections.

Error-based SQL injections are primarily those in which the SQL server dumps some errors back to the user via the web application and this error aids in successful exploitation. In the image below, the yellow line displays the error. These will be discussed further in this post and in related posts to come. Blind SQL injections are those injections in which the backend database reacts to the input, but somehow the errors are concealed by the web application and not displayed to the end users. Or the output is not dumped directly to the screen. Therefore, the name "blind" comes from the fact that the injector is blindly injected using some calculated assumptions and tries.

Based on how the input is treated in SQL query (what data type)

- String-based
- Numeric- or integer based

Based on how the input parameter would be treated in the back end SQL query, an injection can be classified as string- or integer-based.

Based on the degree/order of injections (where the impact happens)

- First-order injections.
- Second-order injections.

The degree or the order of injection identifies the way in which the injection yields the output. If the injection directly delivers the result, it is considered to be a first-order injection, but if the injection input yields no successful result in extraction, but instead impacts some other result which the attacker can take advantage of on some other place/page, it is called a second-order injection. Consider second-order injections similar to stored XSS injections, where the input is stored in the application and later rendered on some other page, thereby impacting that page indirectly because of initial malicious input.

Based on the injection point location

- Injection through user input form fields.
- Injection through cookies.
- Injection through server variables. (headers-based injections)

Generally when an application is communicating with the backend database, it does so in the form of queries with the help of an underlying database driver. This driver is dependent on the application platform being used and the type of backend database, such as MYSQL, MSSQL, DB2, or ORACLE. A generic login query would look something like this: `SELECT Column1, Column2,Column3 FROM table name WHERE username='$variable1′ AND

password='$variable2';` We can split this query into two parts, code section and the data section. The data section is the $variable1 and $variable2 and quotes are being used around the variable to define the string boundary.

Let us try to walk through the process in a crude way. Say at the login form, the username entered is Admin and password is p@ssw0rd which is collected by application and values of $variable1 and $variable2 are placed at their respective locations in the query, making it something like this.

SELECT Column1, column2, Column3 FROM table_name WHERE username='Admin' AND password='p@ssw0rd';`

Now the developer assumes that users of his application will always put a username and password combination to get a valid query for evaluation by database backend. What if the user is malicious and enters some characters which have some special meaning in the query? For example a single quote. So, instead of putting Admin, he puts Admin', thereby causes an error thrown by the DB driver. Why? Because of the unpaired quote entered by the user breaking the application logic.We will discuss the process in detail. To summarize: Whenever an attacker is able to escape the data boundaries, he can append data which then gets interpreted as code by the DB Driver and is executed on the SQL backend, thereby causing SQL injection.

ERROR-based SQL injections

In general, all programming languages give developers a flexibility to debug and fix their applications by using some inbuilt error-handling functions/libraries. These could be some explicit function, classes, or methods that deliver friendly error messages so that the troubleshooting experience can be streamlined and detecting the part of code responsible for raising those exceptions can be easier .These functions should be controlled before an application goes to production because they can dump a lot of sensitive info about the application and underlying logic, thereby making it easy for a bad guy to exploit the application. Therefore, those applications where these error-handling functions are available to aid in gaining useful info about the application or in dumping the database info by means of SQL interaction are classified as error-based SQL Injections[13.14.15]. Based on the way data is extracted using helpful errors, the error-based injections can be classified into two main types:

- Union-query type
- Double-query type

VII. CONCLUSION

After analyzing the various schemes or approaches for protecting the web application databases from SQL Injection attacks. We find that the more research is required in the authentication and authorization phase. The latest encryption algorithms like Advanced Encryption Standard, Secure Hashing techniques, etc should be applied for validating the user with examining its pre-recorded nature and behavior. So, the intensions of malicious users may be pre-measured. The security may be ensured by only permitting the authenticated user to do the database transaction. Many banking systems, reservation systems, etc. are using OTP/ Security Key concept to stop the malfunctioning in the system. These keys are send on their registered mobile no/email. So, that unauthenticated transactions may be stopped. Various methods for generating security key are proposed. Future researcher may work on this secure key generation or on its application on application server.

REFERENCES

[1] Inyong Lee, Soonki Jeong, Sangsoo Yeo, Jongsub Moon, A novel method for SQL injection attack detection based on removing SQL query attribute values, Journal of Mathematical and Computer Modeling, Elsevier Ltd, 2011, pages: 1-11.

[2] Kiezun, A., Guo, P. J., Jayaraman, K., & Ernst, M. D. (2009). Automatic Creation of SQL Injection and Cross-Site Scripting Attacks. ICSE '09. 199-209. Vancouver, Canada.

[3] Wright, C., Freedman, B., & Liu, D. (2008). The IT Regulatory and Standards Compliance Handbook. Burlington, MA, USA: Syngress.

[4] Midian, P. (2003). How to ensure effective penetration test. Information Security Technical Report, 8(4), 65- 77.

[5] Basta, A., & Halton, W. (2008). Computer Security and Penetration Testing. USA: Thomson Course Technology.

[6] Halfond, W. G. J., & Orso, A. (2005). EMNESIA: Analysis and Monitoring for Neutralizing SQL-Injection Attacks. ASE '05, 174-183. Long Beach, California, USA.

[7] Kemalis, K., & Tzouramanis, T. (2008). SQL-IDS: A Specification-based Approach for SQL-Injection Detection. SAC '08. 2153-2158. Fertaleza, Ceara, Brazil.

[8] Newson, A. (2005). Network Threats and Vulnerability Scanner, Network Security, pp. 13-15.

[9] Su, Z., & Wassermann, G. (2006, January 11). The Essence of Command Injection Attack in Web Applications. POPL '06, 372-382, Charleston, South California, USA.

[10] W.G.J. Halfond, A. Orso, P. Manolios, WASP: protecting web applications using positive tainting and syntax-aware evaluation, IEEE Transactions on Software Engineering, 2008, vol. 34 (1), pages: 65–81.

[11] MeiJunjin, An approach for SQL injection vulnerability detection, IEEE Sixth International Conference on Information Technology: New Generations, pages: 1411-1414, 2009.

[12] Lijiu Zhang, Qing Gu, Shushen Peng, Xiang Chen, Haigang Zhao, Daoxu Chen, "D-WAV: A Web Application Vulnerabilities Detection Tool Using Characteristics of Web Forms", IEEE Fifth International Conference on Software Engineering Advances, pages: 501-507, 2010.

[13] K. Natarajan, S. Subramani, "Generation of SQL-injection free secure algorithm to detect and prevent SQL-injection attacks", Procedia Technology 4 (2012) pp. 790 – 796

[14] S. W. Boyd and A. D. Keromytis. SQLrand: Preventing SQLinjection attacks. In Proceedings of the 2nd Applied Cryptography and Network Security (ACNS) Conference, pages 292–302, June 2004.

[15] "Web Application Attack Prevention for Tiered Internet Service " Susanta Nanda, Lap Chung Lam, Fourth Intenational Conference IEEE 2008.

[16] Abhishek Kumar Baranwal, Approaches to detect SQL Injection and XSS in web applications,EECE 571b,Term Survey paper, April 2012.

[17] Sonam Panda, 1 Ramani S2, "Protection of Web Application against Sql Injection Attacks", International Journal of Modern Engineering Research (IJMER) Vol.3, Issue.1, Jan-Feb. 2013 pp-166-168 ISSN: 2249- 6645.

[18] Mihir Gandhi, JwalantBaria,s "SQL INJECTION Attacks in Web Application" International Journal of Soft Computing and Engineering (IJSCE) ISSN: 2231-2307, Volume-2, Issue-6, January 2013.

[19] jin-cherng li and jan-min chen "The Automatic Defence Mechanism for Malicious Injection Attack". Seventh international conference on computer and information technology 2007.

[20] Zeinab Raveshi, Sonali R.Idate "Investigation and Analysis of SQL Injection Attacks on Web Applications: Survey" International Journal of Engineering and Advanced Technology (IJEAT) ISSN: 2249 – 8958, Volume-2, Issue-3 February 2013.

[21] Priyanka, Vijay Kumar Bohat," Detection of SQL Injection Attack and Various Prevention Strategies", International Journal of Engineering and Advanced Technology (IJEAT) ISSN: 2249 – 8958, Volume-2, Issue-4, April 2013.

[22] R.Rani, B.S.Kumar, L.T.R.Rao, V.T.S. Jagdish, M.Pradeep, "Web Security by Preventing SQL Injection Using Encryption in Stored Procedures", IJCSIT, Vol 3(2), 2012,3689-3692, ISSN : 0975-9646.

