SQL Injection Attacks and Prevention: An Overview

Anupama Chowdhary
Principal, Keen College
Bikaner (Rajasthan)
India

Abstract – Now-a-days, our Internet users interact almost daily with web applications which use databases at the backend. The web applications are publically accessible, so are exposed to various forms of security attacks. One of the most common attacks on database system is SQL injection. This paper provides a critical study regarding, various types of SQL Injection attacks, vulnerabilities and their detection and prevention techniques.

Index terms – SQL injection, Cross Site Scripting Attack (XSS), Denial of Service (DoS), spoofing, splicing.

I. INTRODUCTION

Today, almost all the services are online such as payment of bills, various photo identity cards (Aadhar, Driving license, passport, PAN card etc.), students' records of universities and boards, different other government web applications, and over the top very sensitive banking data. Great volumes of data are stored in Web application databases to serve this massive number of users. The users need to interact with the backend databases through the user interfaces for extracting data, updating data, making queries, etc. Design interface plays a vital role for all these operations. The quality and security measures opted for these interfaces has an excessive impact on the security of the stored data in the database.

Database faces several threats such as Cross Site Scripting Attack (XSS), Denial of Service (DoS), phishing and SQL injection attack due to poor design, configuration mistakes, or poor written code of the web application. A threat can be harmful for database, control of web application, and other components of web application, that are needed to be protected from all types of threat. SQL injection attack is the most effective method for stealing the sensitive information or data from the backend database.

In 1998, the concept of SQL injection was put forward by Rain Forest Puppy for the first time [1]. This attack is so common that almost all websites have encountered it and is also so flexible that it's hard to discover and avoid it.

The major cause of SQL injection vulnerabilities is invalidated input received from a Web form, cookie, input parameter, and so forth. The Web application developer must ensure that values received are validated before passing them to SQL queries that will be executed on a database server. The attacker may be able to execute the code on the back-end database if somehow an attacker can control the input sent to an SQL query and manipulate that input so that the data is interpreted as a code instead of as data.

For providing a better service to their customers some organizations use web applications with dynamic database to build a shared environment. Dynamic string building is a programming technique that enables developers to build SQL statements dynamically at runtime. A dynamic SQL statement is constructed at execution time, for which different conditions generate different SQL statements. Developers can achieve the same result in a much more secure fashion if they use parameterized queries. Parameters can be passed to these queries at runtime; parameters containing embedded user input would not be interpreted as commands to execute, and there would be no opportunity for code to be injected.

II. DATABASE SECURITY

Database security is the system processes that protect data in the database from unintended activity. The system process uses a wide range of data security controls to protect databases against any internal or external attacks so that database confidentiality, integrity and availability do not compromises. Protecting the confidential/sensitive data stored in a repository is actually the database security [2]. There are various security layers in a database. These layers are: database administrator system administrator, security officer, developers and employee [2] and security can be breached at any of these layers by an attacker [3].

Database security problems can be summarized in four categories namely: Inference, Active attacks, Passive attacks and SQL Injections attack [4].

Inference

Hackers derive the useful information from non-sensitive data in this method. At the core of the inference attack is a simple question. If the answer to this question is A then do Y; if the answer is B then do Z [5]. By this way, hackers do not need to steal the sensitive data directly; they only need to derive the useful information from non-sensitive data [6]. Suppose C=A+B, where A and C are public then B could be inferred from A and C. Hackers use mathematical methods such as indirect attack, linear system or tracker attack to get sensitive data.

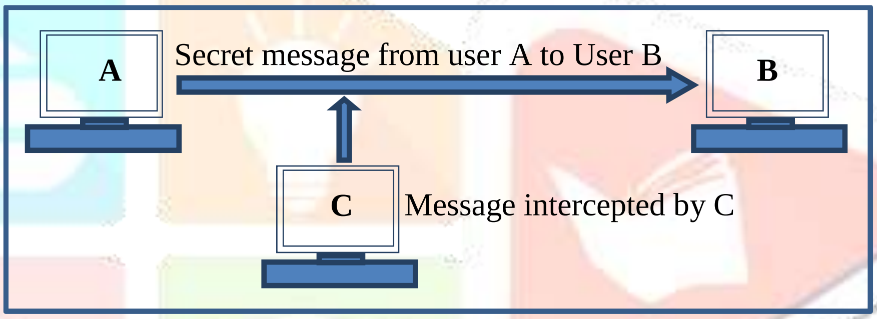

Figure 1 Passive Attack

Passive Attacks

In passive attacks hackers are only interested in the data which have already presented in the database he does not attempt to alter the system or change data. In general, there are three ways to do this attack: Static leakage, linkage leakage and dynamic leakage [7].

Static leakage: Information can be obtained from a database by observing snapshot of it.

Linkage leakage: Information about plain text values can be obtained by linking table values to position of those values in index.

Dynamic leakage: Changes performed in database over a period of time can be observed and information about plain text values can be obtained.

Active Attacks

Active attacks are more dangerous as hackers not only get the information from the database but also modify the actual data inadequately. The active attack can be performed by:

Spoofing: Using some algorithms and techniques, a value is generated and then the cipher text is replaced by that value.

Splicing: In this attack one cipher text value is replaced by another cipher text value.

Replay: In replay attack cipher text value is replaced with old version previously updated or deleted value [7].

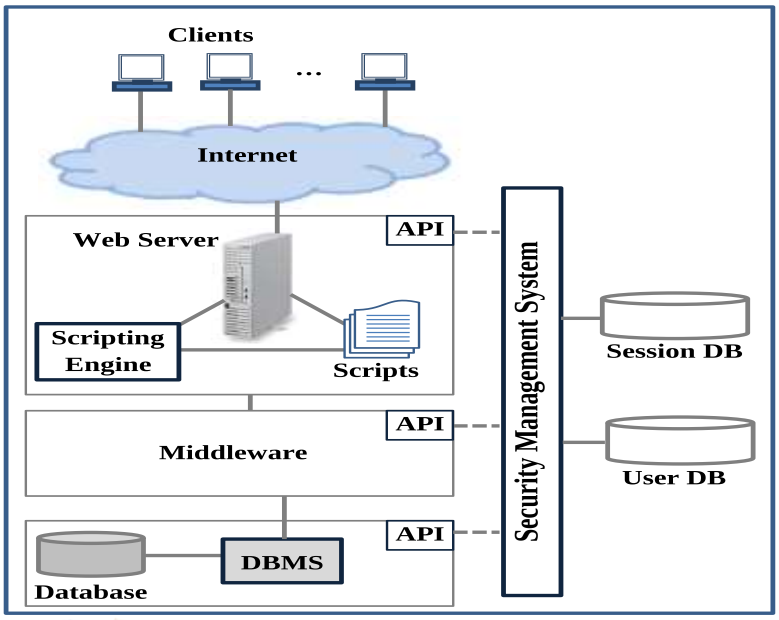

Figure 2 Database and Web Application Model

SQL Injection Attack

On April 2018 statista.com [8] states that over 4 billion people were active internet users. With the continuous development of Internet, more and more database server as a backend for web application. These databases are attractive targets for attackers as they contain important assets such as usernames, passwords, personal data, financial information etc. One of the major attacks on these up-to-date databases is SQL injection attack.

SQL injection attack is performed intentionally by an attacker either to gain unauthorized access to a database or to retrieve information directly or indirectly from the database. The attacker inserts a portion of SQL statement via not sanitized user input parameters into the original query and passes them to database server [9]. The input is accepted from users by web applications and then it is incorporated into dynamically generated code [10].

III. SQL INJECTION ATTACK (SQLIA)

SQL Injection attacks are aimed at database under web environment. Typically, database management system run on the top of an operating system and thus provides the security associated with a database. However, with web environment the situation changes and database is accessible via web API so hackers can access the database. SQL injection is the most common method hacker's use. SQLIA is a three phase process [11]:

1. An attacker sends the malicious HTTP request from a client to the web application as input.
2. Web API generates a SQL statement
3. Submits this SQL statement to the back end database server.

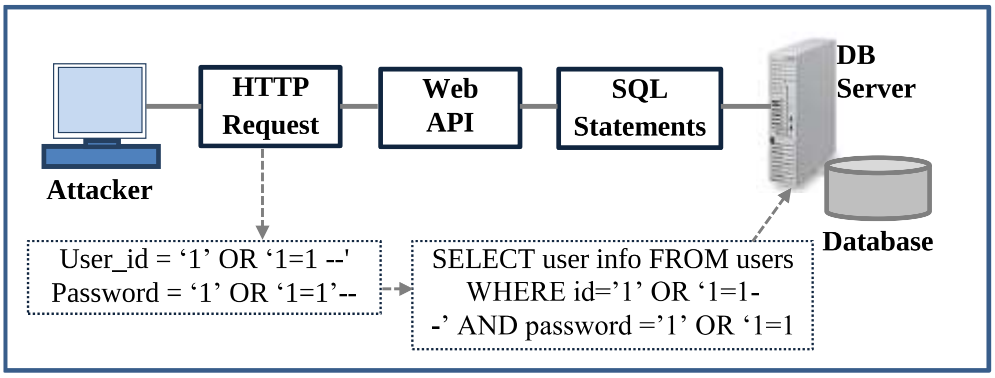
Attacker
HTTP Request
Web API
SQL Statements
DB Server
Database
User_id = '1' OR '1=1 --'
Password = '1' OR '1=1'--
SELECT user info FROM users WHERE id='1' OR '1=1--' AND password ='1' OR '1=1

Figure 3 SQL Injection Attack Process

The given input shown in figure 3 makes the WHERE clause in the SQL statement a tautology which always returns a true condition. The database will return all the user information in the table. Thus, the malicious user has been authenticated as a DBA without a valid login id and password [13].

Some syntax errors lead to weakness in the web programming language. The poor programming or coding practice is the major cause to vulnerabilities such as improper sanitizations of inputs, type checking, over privilege accounts and detailed error messages [12]. According to the specific kind of weakness present in the application the attacker can plan a specific attack.

IV. SQL INJECTION MECHANISMS AND GOALS OF ATTACKER

The major goals or intents of an attacker is to

- Get valuable data such as SSN, Bank Account, credit card password and other private information.
- Modify data such as delete some record or rewrite the original data.
- Change the schema of database.
- Cutting down the connection between server and database (DoS).
- Controlling system remotely to gain control of the whole system [14].

Various input mechanisms are used to inject malicious SQL statements into a vulnerable application.

Injection through cookies

Cookies are text files that contain state information generated by web applications and are attached to the browser and are stored on the client machine. Whenever a client connects to the same web server cookies can be used to restore the client's state information. The client has control over the storage of the cookie so a malicious client could tamper with the cookie's contents. If a Web application uses the cookie's contents to build SQL queries, an attacker could easily submit an attack by embedding it in the cookie [16].

Query based Injections

Attacks are generally not performed in segregation; many of them are used together or sequentially, depending on the specific goals of the attacker. Moreover, there are countless variations of each attack type. In these types of attacks data provided by the user is included in an SQL query in such a way that part of the user's input is treated as SQL code. Query based injection are summarized in table 1.

Table 1

SQLIA Type	Goal	Description
Tautologies	Bypassing authentication, identifying injectable parameters, extracting data.	The code injected in the condition (OR 1=1) transforms the entire WHERE clause into a tautology. (Figure 3)
Illegal/ Logically Incorrect Queries	Identifying injectable parameters, performing database finger-printing, extracting data.	The attack is considered a preliminary, information gathering step for other attacks. An attacker tries to inject statements that cause a syntax, type conversion, or logical error into the database. Syntax errors can be used to identify injectable parameters. Type errors can be used to deduce the data types of certain columns or to extract data. Logical errors often reveal the names of the tables and columns that caused the error
Union Query	Bypassing Authentication, extracting data	Using the statement of type `UNION SELECT <rest of injected query>`, an attacker can trick the application for returning data from a table different from the one that was intended by the developer.
Piggy-Backed Queries	Extracting data, adding or modifying	The attacker tries to include new and distinct queries that "piggy-back" on the original query. As a result, the

	data, performing denial of service, executing remote commands	database receives multiple SQL queries. Normally, in the value of any field query delimiter (";") is used for injecting second query.
Stored Procedures	Performing privilege escalation, performing denial of service, executing remote commands	SQLIAs can be crafted to execute stored procedures once an attacker knows which back end database is in use. He can also execute procedures that interact with the operating system. Stored procedures are often written in special scripting languages, they can contain other types of vulnerabilities, such as buffer overflows, that allow attackers to run arbitrary code on the server or escalate their privileges [18]
Inference	Identifying injectable parameters, extracting data, determining database schema	Attackers derive the useful information from non-sensitive data in this method. At the core of the inference attack is a simple question. If the answer to this question is A then do Y; if the answer is B then do Z [5].
Alternate Encodings	Evading detection	In this technique, attackers use different encoding method to inject database, such as ASCII, Unicode. For example [19] "`legalUser'; exec (0x73687574646f776e) - -`" could be injected in the login field. The stream of numbers in the second part of the injection is the ASCII hexadecimal encoding of the string "`SHUTDOWN`".

Injection through server variables

Server variables contain network headers, HTTP and environmental variables. Web applications use these server variables for logging usage statistics and identifying browsing trends. Attackers can forge the values that are placed in HTTP and network headers. If these variables are logged to a database without sanitization, this could create an SQL injection vulnerability [30].

Second-order injection

Let us consider an example given by Anely [17] to understand this concept.

- A user registers on a website using a seeded user name, such as "admin' --".
- The application properly escapes the single quote in the input before storing it in the database, preventing its potentially malicious effect.
- The user modifies his or her password for this web API checks that the user knows the current password and changes the password if the check is successful. For this, the Web application might construct an SQL command as:
  queryString="UPDATE users SET password='" + newPassword + "' WHERE userName='" + userName + "' AND password='" + oldPassword + "'"

As "--" is the SQL comment operator, everything after it is ignored by the database. Thus, the result of this query is that the database changes the password of the administrator ("admin") to an attacker-specified value. In second-order injections attacker are not trying to cause the attack to occur when the malicious input initially reaches the database. Instead their input is subsequently used to craft their attack.

V. PREVENTION TECHNIQUES

A wide range of techniques have been proposed by the researchers to resolve the problem of SQL injection. These techniques focus on development best practices to fully automated frameworks for detecting and preventing SQLIAs. Detecting and preventing injection attack primarily is pretty important for ensuring the safety of valuable information.

Defensive Coding

While coding an application the programmer should take these safety measures.
Input type checking: Simple check should be performed on inputs can prevent many attacks.
Identification of all input sources: There can be many possible sources of input to an application used to construct a query; so all input sources must be checked.
Positive pattern matching: Specify all the forms of legal inputs, as it is difficult to specify injection attack forms. Positive validation is a safer way to check inputs.
Encoding of inputs: Prohibit any usage of meta-characters in input string values or encode input string in such a way so that database interprets meta-characters as normal characters.

There are various methods developed to measure the effectiveness of the validity checking by the web programmer and the effectiveness of the tools applied to detect code that causes SQLIA [20].

Detection Approaches

For analysis of dynamic or runtime SQL query, detection approaches are useful. A web Application executes SQLIA detection method on user input data before posting a query to the database [20]. These approaches are based on the initialization of trusted or untrusted strings, which depends upon the developer.
Context Sensitive String Evaluation: The tool will first regard user input as non-trusted data. In a step-by-step procedure this input is analysed from the application which is regarded as trusted and the unsafe characters will be removed [21].
Positive tainting and Syntax aware evaluation: Positive tainting can address problems caused by incompleteness which may lead to false positives [22].
Parse tree evaluation based on grammar: When hackers inject SQL into a database query, the parse tree of the intended SQL query and the resulting SQL query do not match [23].

Although, many other methods were developed for detection of SQL injection but most of the techniques such as mutation testing, tokenization, multi-layer defence mechanism, service based approach and syntactic and semantic analysis are anomaly base or signature base. These techniques compare input with saved patterns but the hackers are constantly looking for new vulnerabilities to attack, even the previous validation functions are also misused [24].

Hash algorithm

This method contains three models [25]:
Web crawler: It is an interface to interact with the web application. The system can get the information from website including status-code, cookies and message-body and HTML content.
Injector: It send invalid payload to the affected parameter in the HTTP request.
Analyser: The system can use the hash value to detect if web has vulnerability.

Support Vector Machine

This Approach detects SQL injection by modelling SQL queries as graph of tokens and using the centrality measure of nodes to train a Support Vector Machine (SVM) [26, 27]. The system has following steps:

- Transform the SQL query to a sequence of tokens preserving its structural composition.
- Generate a graph consisting of tokens as the nodes and connection between them as weighted edges.
- Use the degree centrality measure of nodes to train the SVM classifier.
- Use the SVM model to address the malicious queries at runtime.

The system is designed to work at the database firewall layer.

Fast Flux Networks and SQLIA

For detecting fast flux networks and SQL Injection attacks the Expert Systems Machine learning methods were used by Holz et al. and Stalmans et al. [27, 28]. The emphasis was on C5.0 Classifier and Naive Bayesian Classifier.

DNS Hijacking and SQLIA

To prevent the DNS Hijacking lightweight session shield could be used that is a client side protection mechanism [29]. Stampar [30] discusses about the usage of SQLMap in protection of the SQLI + DNS Attack.

DDoS and SQLIA

The cluster analysis methodology was used to detect DDoS Attacks by Lee et al [31].

VI. CONCLUSION

Attacker use SQL injection statements to retrieve and manipulate information from back end database. If there is some loopholes in the security policy while developing web application, it is an extra advantage to attackers. Some attackers have made it a profitable business as a black economy such as trades of credit card numbers, personal information and bank accounts. In this paper, we have analysed how security of database is compromised by various types of attacks. Further, mechanism of SQL injection attacks into a web application and the goals of attacker are discussed. Various techniques for detecting and preventing SQLIAs are disused in the last section. To control these types of attacks the web application developer should have the good knowledge of these attacks.

REFERENCES

[1] Rain Forest Puppy, “NT Web Technology Vulnerabilities”, Phrack Magazine Volume 8, Issue 54 Dec 25th, 1998, article 08 of 12.

[2] Sohail IMRAN, Dr Irfan Hyder, “Security Issues in Database”, Second International Conference on Future Information Technology and Management Engineering, 2009.

[3] Shelly Rohilla, Pradeep Kumar Mittal, “Database Security: Threats and Challenges”, International Journal of Advanced Research in Computer Science and Software Engineering, Volume 3, Issue 5, May 2013.

[4] Mr. Saurabh Kulkarni, Dr. Siddhaling Urolagin, “Review of Attacks on Databases and Database Security Techniques, Facility” International Journal of Engineering Technology and Database Security Techniques Research, Volume 2, Issue 11, November-2012.

[5] Halfond, William G., Jeremy Viegas, and Alessandro Orso. "A classification of SQL injection attacks and countermeasures." Proceedings of the IEEE International Symposium on Secure Software Engineering. Vol. 1. IEEE, 2006.

[6] Pfleeger C P, Pfleeger S L. Security in Computing[M]. China Machine Press, 2004.

[7] Shmueli E, Vaisenberg R, Elovici Y, et al. Database encryption: an overview of contemporary challenges and design considerations[J]. Acm Sigmod Record, 2010, 38(3):29-34.

[8] https://www.statista.com/statistics/617136/digital-population-worldwide/

[9] Antunes, N. & Vieira, M., 2012. “Defending against Web Application Vulnerabilities”. 45(2), pp.66-72.

[10] Son, S., Mckinley, K.S. & Shmatikov, V., 2013. Diglossia: “Detecting Code injection attacks with precision and Efficiency”. ACM, pp.1181-91.

[11] Nithya, V., regan, R. & Vijayaraghavan, J., 2013. A survey on SQL injection attacks, their Detection and Prevention techniques. International Journal of Engineering and Computer Science, 2(4), pp.886-905.

[12] Sharma, C. & S.C.Jain, D., 2014. Analysis and Classification of SQL injection vulnerabilities and Attacks on Web Applications. In IEEE International Conference on Advances in Engineering and Technology Research (ICAETR). Kota, 2014.

[13] MeiJunjin, 2009. Anon vulnerability detection approach for SQL inject. IEEE, pp.1411-14.

[14] William G.J. Halfond, Jeremy Viegas and Alessandro Orso, “A Classification of SQL Injection Attacks and Countermeasures,” College of Computing Georgia Institute of Technology IEEE, 2006.

[15] M. Dornseif. Common Failures in Internet Applications, May 2005. http://md.hudora.de/presentations/

[16] T. M. D. Network. Request.servervariables collection. Technical report, Microsoft Corporation, 2005. http://msdn.microsoft.com/library/default.asp?url=/library/en us/iissdk/html/9768ecfe-8280-4407-b9c0-844f75508752.asp
[17] C. Anley. Advanced SQL Injection In SQL Server Applications. White paper, Next Generation Security Software Ltd., 2002.
[18] E. M. Fayo. Advanced SQL Injection in Oracle Databases. Technical report, Argeniss Information Security, Black Hat Briefings, Black Hat USA, 2005.
[19] M. Howard and D. LeBlanc. Writing Secure Code. Microsoft Press, Redmond, Washington, second edition, 2003.
[20] Das D, Sharma U, K. Bhattacharyya D. Rule based Detection of SQL Injection Attack[J]. International Journal of Computer Applications, 2012, 43(19):15-24.
[21] Pietraszek T, Berghe C V. Defending Against Injection Attacks Through Context Sensitive String Evaluation[C]// International Conference on Recent Advances in Intrusion Detection. Springer-Verlag, 2005:124-145.
[22] Halfond W G J, Orso A, Manolios P. Using positive tainting and syntax-aware evaluation to counter SQL injection attacks[C]// ACM Sigsoft International Symposium on Foundations of Software Engineering, FSE 2006, Portland, Oregon, Usa, November. DBLP, 2006:175-185.
[23] Buehrer G, Weide B W, Sivilotti P A G. Using parse tree validation to prevent SQL injection attacks[C]// International Workshop on Software Engineering and Middleware, Sem 2005, Lisbon, Portugal, September. DBLP, 2005:106-113.
[24] Yeole, A.S. & Meshram, B.B., 2011. Analysis of Different Technique for Detection of SQL Injection. International Conference and workshop on Emerging Trends in Technology(ICWET 2011). Mumbai, 25-26 February 2011.
[25] M Mahdi, AH Mohammad. Using hash algorithm to detect SQL injection vulnerability[J]. international journal of research in computer applications and robotics, 2016, 4(1):26-32.
[26] Solomon Ogbomon Uwagbole, William J. Buchanan, Lu Fan, "Applied Machine Learning Predictive Analytics to SQL Injection Attack Detection and Prevention", IFIP/IEEE IM 2017 Workshop: 3rd International Workshop on Security for Emerging Distributed Network Technologies, pg. 1087-1090.
[27] T. Holz, C. Gorecki, K. Rieck, and F. C. Freiling, "Measuring and detecting fast-flux service networks." in NDSS, 2008.
[28] E. Stalmans and B. Irwin, "A framework for dns based detection and mitigation of malware infections on a network," in Information Security South Africa (ISSA), 2011. IEEE, 2011, pp. 1–8.
[29] N. Nikiforakis, W. Meert, Y. Younan, M. Johns, and W. Joosen, "Sessionshield: Lightweight protection against session hijacking," in Engineering Secure Software and Systems. Springer, 2011, pp. 87–100.
[30] M. Stampar, "Data retrieval over dns in sql injection attacks," arXiv preprint arXiv:1303.3047, 2013.
[31] K. Lee, J. Kim, K. H. Kwon, Y. Han, and S. Kim, "Ddos attack detection method using cluster analysis," Expert Systems with Applications, vol. 34, no. 3, pp. 1659– 1665, 2008.