SQL INJECTION: A SURVEY PAPER

Amit Banchhor[1], Tushar Vaidya[2]

[1,2] M.Tech.Scholar, Dept.of Computer Science & Engg. CSIT, Durg, (India)

ABSTRACT

SQL injection is a code injection technique, used to attack data driven applications, in which malicious SQL statements are inserted into an entry field for execution[1]. SQL injection must exploit a security vulnerability in an application's software. Applications are filtered for string literal escape characters embedded in SQL statements. When the user input is not strongly typed and unexpectedly executed these causes injection.SQL injection is mostly known as an attack vector for websites but can be used to attack any type of SQL database. There are various types of injection techniques that are used by the hackers to get access to the database. This paper discusses all those types of techniques those are widely used by the hackers to cause a SQL injection.

Keywords: SQL, Security, Strongly Typed, Attack Vector, Database

I. INTRODUCTION

SQL injections attacks(SQLIA) are caused when there is change in the SQL query by insertion of new SQL keywords into the existing query. SQL injection vulnerabilities are one of the most serious threats for web applications. These injections allow an attacker to gain complete access to the database. In some case the hacker may use an SQL injection to take control and corrupt the system that hosts the web application.

SQL injection refers to a class of code-injection attacks in which 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. By taking advantage of these vulnerabilities, an attacker can submit SQL commands directly to the database. These attacks are serious to any type of web applications that receives information from the users as input and incorporate it into SQL queries to an underlying database.

The major cause of SQL injection is insufficient user validation, to address this problem developers have proposed a range of guidelines for coding, which promotes defensive coding practices, such as encoding user input and validation.[2] A rigorous and systematic application of these applications is an effective solution for preventing SQL injection vulnerabilities. However, in practice the application of such techniques is prone to human errors. Furthermore, fixing legacy code-bases that might contain SQL injection vulnerabilities can be extremely labor-intensive task.

II. SQLIA TECHNIQUES

The SQLIA that is SQL injection attacks can be categorized into three major groups. These groups are:

2.1 Injection Through User Input

In this technique, hacker injects SQL keywords or statements by selectively selecting the user's input. A web application usually reads the user's input in several ways depending on the environment in which it is developed. In most of the applications user's input is collected via online forms through HTTP GET or POST requests

2.2 Injection Through Cookies

In this technique cookies are used to create a malicious code. Cookies are the special files that contain information of the state of the web page generated by the web applications. These are stored locally on the client system. Cookies are used to restore the client's state information when there is switching among the web pages. Since the cookies are located locally hence a malicious code could be embedded in it to cause unintended activity in the application.

2.3 Injection Through Server Variables

Server variables are a collection of variables that contain HTTP, network headers, and environmentalvariables. Web applications use these server variables ina variety of ways, such as logging usage statistics and identifyingbrowsing trends. If these variables are logged to a database withoutsanitization, this could create an SQL injection vulnerability.Because attackers can forge the values that are placed in HTTP andnetwork headers, they can exploit this vulnerability by placing an SQLIA directly into the headers. When the query to log the servervariable is issued to the database, the attack in the forged header isthen triggered.

2.4 Second Order Injection

In second-order injections, attackers put malicious inputs into a system or database and indirectly trigger anSQLIA when that input is used at a later time. The objective ofthis kind of attack differs significantly from a regular injection attack. Second-order injections are not trying tocause the attack to occur when the malicious input initially reachesthe database. Instead, attackers rely on knowledge of where theinput will be subsequently used and craft their attack so that it occursduring that usage.

III. SQLIA TYPES

There are various types of SQLIA, these are not applied in isolation but are used in aggregation or in sequence depending on the target of the hacker. There are various variations of each SQLIA type, majority of them are as follows:

3.1 Tautology

The general goal of a tautology-based attack is to inject code in one or more conditional statements so that they always evaluate to true.[3] The consequences of this attack depend on how the results of the query are used within the application. The most common usages are to bypass authentication pages and extract data. In this type of injection, an attacker exploits an inject able field that is used in a query's WHERE conditional. Transforming the conditional into a tautology causes all of the rows in the database table targeted by the query to be returned. In general, for a tautology-based attack to work, an attacker must consider not only the inject able/vulnerable parameters, but also the coding constructs that evaluate the query results. Typically, the attack is successful when the code either displays all of the returned records or performs some action if at least one record is returned.

3.2 Illegal or Illogical Queries

This attack lets an attacker gather important information about the type and structure of the back end database of a Web application. The attack is considered a preliminary, information gathering step for other attacks. The vulnerability leveraged by this attack is that the default error page returned by application servers is often overly descriptive. In fact, the simple fact that an error messages is generated can often reveal vulnerable/injectable parameters to an attacker. Additional error information, originally intended to help programmers debug their applications, further helps attackers gain information about the schema of the back-end database. When performing this attack, 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.

3.3 Union Query

In union-query attacks, an attacker exploits a vulnerable parameter to change the data set returned for a given query. With this technique, an attacker can trick the application into returning data from a table different from the one that was intended by the developer. Attackers do this by injecting a statement of the form: UNION SELECT <rest of injected query>. Because the attackers completely control the second/injected query, they can use that query to retrieve information from a specified table. The result of this attack is that the database returns a dataset that is the union of the results of the original first query and the results of the injected second query.[3]

3.4 Piggy-Backed Queries

In this attack type, an attacker tries to inject additionalqueries into the original query. We distinguish this type from othersbecause, in this case, attackers are not trying to modify the originalintended query; instead, they are trying to include new and distinctqueries that “piggy-back” on the original query. As a result, the database receives multiple SQL queries. The first is the intendedquery which is executed as normal; the subsequent ones are the injected queries, which are executed in addition to the first. Thistype of attack can be extremely harmful. If successful, attackers can insert virtually any type of SQL command, including storedprocedures, into the additional queries and have them executed along with the original query. Vulnerability to this type of attackis often dependent on having a database configuration that allowsmultiple statements to be contained in a single string.

3.5 Stored Procedures

SQLIAs of this type try to execute stored procedures present in the database.[4] Today, most database vendors ship databases with a standard set of stored procedures that extend the functionality of the database and allow for interaction with the operating system. Therefore, once an attacker determines which back-end-database is in use, SQLIAs can be crafted to execute stored procedures provided by that specific database, including procedures that interact with the operating system. It is a common misconception that using stored procedures to write Web applications renders them invulnerable to SQLIAs. Developers are often surprised to find that their stored procedures can be just as vulnerable to attacks as their normal applications. Additionally, because 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.[3]

3.6 Inference

In this attack, the query is modified to recast it in the form of an action that is executed based on the answer to a true/false question about data values in the database. In this type of injection, attackers are generally trying to attack a site that has been secured enough so that, when an injection has succeeded, there is no usable feedback via database error messages. Since database error messages are unavailable to provide the attacker with feedback, attackers must use a different method of obtaining a response from the database. In this situation, the attacker injects commands into the site and then observes how the function/response of the website changes. By carefully noting when the site behaves the same and when its behavior changes, the attacker can deduce not only whether certain parameters are vulnerable, but also additional information about the values in the database. There are two well known attack techniques that are based on inference. They allow an attacker to extract data from a database and detect vulnerable parameters.

3.7 Blind Injection

In this technique, the information must be inferred from the behavior of the page by asking the server true/false questions. If the injected statement evaluates to true, the site continues to function normally. If the statement evaluates to false, although there is no descriptive error message, the page differs significantly from the normally-functioning page.

3.8 Timing Attacks

A timing attack allows an attacker to gain information from a database by observing timing delays in the response of the database. This attack is very similar to blind injection, but uses a different method of inference. To perform a timing attack, attackers structure their injected query in the form of an if/then statement, whose branch predicate corresponds to an unknown about the contents of the database. Along one of the branches, the attacker uses a SQL construct that takes a known amount of time to execute. By measuring the increase or decrease in response time of the database, the attacker can infer which branch was taken in his injection and therefore the answer to the injected question.

3.9 Alternate coding

In this attack, the injected text is modified so as to avoid detection by defensive coding practices and also many automated prevention techniques. This attack type is used in conjunction with other attacks. In other words, alternate encodings do not provide any unique way to attack an application; they are simply an enabling technique that allows attackers to evade detection and prevention techniques and exploit vulnerabilities that might not otherwise be exploitable. These evasion techniques are often necessary because a common defensive coding practice is to scan for certain known “bad characters,” such as single quotes and comment operators. To evade this defense, attackers have employed alternate methods of encoding their attack strings (e.g., using hexadecimal, ASCII, and Unicode character encoding). Common scanning and detection techniques do not try to evaluate all specially encoded strings, thus allowing these attacks to go undetected. Contributing to the problem is that different layers in an application have different ways of handling alternate encodings. The application may scan for certain types of escape characters that represent alternate encodings in its language domain. Another may use different escape characters or even completely different ways of encoding. For example, a database could use the expression char(120) to represent an alternately-encoded character “x”, but char (120) has no special meaning in the application language’s context. An effective code-based defense against alternate encodings is difficult to implement in practice because it requires developers to consider of all of the possible encodings that could affect a given query string as it passes through the different application layers. Therefore, attackers have been very successful in using alternate encodings to conceal their attack strings.[4]

IV. CONCLUSION

In this paper we have explored various types of SQLIA possible and their effects on the underlying database of the web application. Future evaluation work should focus on evaluating the techniques precision and effectiveness in practice.

REFERENCES

[1] http://en.wikipedia.org/wiki/SQL_injection
[2] AtefehTajpour, SuhaimiIbrahim, Mohammad Sharifi, “Web Application Security by SQL Injection Detection Tools”, IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 2, No 3, March (2012).
[3] Dharam, R., “Runtime monitors for tautology based SQL injection attacks”, Cyber Security, Cyber Warfare and Digital Forensic, IEEE, ISBN:978-1-4673-1425-1, June (2012).
[4] Pinzon, C., “AIIDA-SQL: An Adaptive Intelligent Intrusion Detector Agent for detecting SQL Injection attacks.” ,Hybrid Intelligent Systems (HIS),IEEE, ISBN: 978-1-4244-7363-2, August (2010).