SQL Injections: Causes and Mitigation

Divya Aradhya

Saint Leo University

Author's Note

Contact: divya.aradhya@saintleo.edu

August 4, 2017

Table of Contents

Abstract 3

Brief Description 4

Primary Issues 6

Current Mitigation Solutions 7

Insights and Remarks 9

References 11

Abstract

The heart of an organization is its database, and all the security measures employed revolve around keeping the database secure and protected. Structured Query Language is the language used to communicate with the database to create, modify, and maintain data. The SQL injection is a simple, yet deadly, mechanism that can wreak havoc on databases.

This paper is a study of the SQL injection, its relevance and significance in the realm of information security, the primary issues pertaining to it, the current mitigation methods employed ("what has been done"), and insights and remarks based on these findings and study.

Brief Description

Structured Query Language (SQL) is a database language used by Oracle, Microsoft SQL, MySQL, and other enterprise databases to query, operate, and administer databases and tables. While each database has created its own flavor of SQL, at the core they are all based on the principles of the generic SQL statements. Databases consist of tables, populated with data, and metadata, and it represents the backbone of an organization or business. Database systems are commonly used to provide backend functionality to many types of web applications. Applications are created to interact with this data and form a bridge between the user and the database to accept input, query the database and display results, based on roles, permissions, and authorization ("Understanding SQL Injections").

A SQL injection is a form of input on the application that manipulates the database to bypass the security restriction and cause it to execute in unexpected and destructive ways. While SQL injections can be accidental, they are almost always very carefully crafted malicious attacks that are launched by attackers to gain access, manipulate, or even delete and destroy data.

The Open Web Application Security Project – OWASP – puts SQL Injections as #1 on their list of top ten vulnerabilities and attacks ("Top 10 2013-Top 10")

Relevance and Significance

The earliest description of SQL injection attacks can be traced to 1998 when Jeff Forristal, a security researcher, used a pseudonym moniker 'rain.forest.puppy', to publish his findings in a hacker journal called 'Phrack'. He wrote about the about various features of Microsoft's IIS 3 and 4 Web servers and its SQL injection vulnerabilities ("How security flaws work: SQL injection").

Since that time, SQL injection have been used effectively to exploit systems and compromise sensitive data in massive breaches. In 2016, a Florida native was charged with felony for hacking into a government election site and accessing sensitive data. He used SQL injection for his attack ("How a security pro's ill-advised hack of a Florida elections site backfired"). The SQLi Hall-Of-Shame on an average shows at least two major data breaches due to SQL injections, every month ("SQLi Hall-of-Shame")

The SQL Injection heat map depicts how widespread the attack origins are throughout the world map. Hence not only is SQL injections a grave widespread problem, it is also impossible to restrict by geo-blocking.

("Website Attacks - SQL Injection And The Threat They Present").

A Barclays report states that 97 percent of data breaches resulting in hundreds of thousands of dollars of losses are due to SQL injection attacks ("Barclays: 97 percent of data breaches still due to SQL injection").

Primary Issues

The primary issue around SQL injections is that they can technically be simple to launch, as the point of entry is an exposed webpage, but the ramifications they can cause are multi-fold and devastatingly destructive.

The effects of SQL injections can result in the following security breaches-

- *Authentication Bypass*: The attacker gains elevated privileges and role-based access into tables and rows and columns of data that is typically not accessible for the end user. He can also get elevated to database administrative privileges without supplying a valid username and password, and run Data Definition Language statements (CREATE, ALTER, DROP) which can complete destroy the very structure of the database.
- *Information Disclosure*: The attacker gains access and privy to sensitive, restricted, and protected data that can result in a breach of confidentiality.
- *Compromised Data Integrity*: The attacker injects malicious code to manipulate data within the database. An attack of this sort leads to a breach data integrity and leads of unimaginable losses if the database of a financial institution is attacked, and the table containing the money in the account is altered. An attacker can also deface a public website with false data leading to political tension, riots, and mayhem.
- *Compromised Availability of Data*: The attacker accesses the data, makes a copy, and deletes valuable information, or the entire dataset. An organization can be brought down to its knees when its database is deleted overnight; and
- *Remote Command Execution*: An attacker can perform remote command execution through a database and eventually compromise the host operating system. These attacks often leverage an existing, predefined stored procedure for host operating system

command execution. The most recognized variety of this attack uses the xp_cmdshell stored procedure that is common to Microsoft SQL Server installations or leverages the ability to create an external procedure call on Oracle databases (“Understanding SQL Injection”).

Current Mitigation Solutions

Having realized the gravity of the destruction caused by SQL injections, organizations like the OWASP, have complied the findings of various security research publications and put out guidelines for best practices.

In order to prevent SQL injections, the primary defenses would include-

- using pre-pared SQL statements in parameterized queries: eg, in C#.NET

```
String query =
 "SELECT account_balance FROM customer_table WHERE user_name = ?";
try
{
        OleDbCommand ODcommand = new OleDbCommand(query, connection);
        ODcommand.Parameters.Add(new OleDbParameter("customerName", CustomerName
Name.Text));
        OleDbDataReader ODreader = ODcommand.ExecuteReader();
        // …
}
catch (OleDbException se)
{
        // error handling
}
```

- using safe stored procedures; eg. In VB.NET

```
Try
        Dim sCommand As SqlCommand = new SqlCommand("storedproc_getAccountBalance",
connection)
        sCommand.CommandType = CommandType.StoredProcedure
        sCommand.Parameters.Add(new SqlParameter("@CustomerName", CustomerName.Text))
        Dim sReader As SqlDataReader = sCommand.ExecuteReader()
‘ …
Catch se As SqlException
        ‘ error handling
End Try
```

- whitelisting valid input

  All SQL database and table identifying input should be validated against a strict whitelist.

  Eg. In C#.NET-

  String tableName;

```
switch(PARAM):
  case "Value1": tableName = "userTable";
          break;
  case "Value2": tableName = "productTable";
          break;
    ...
  default     : throw new InputValidationException("table name not in whitelist – cannot process it")
```

- escaping all user supplied input

  This technique is to escape user input supplied before putting it in a query. It is very specific to database used (Oracle, MS-SQL, MySQL) in its implementation. Eg. In Oracle-

```
String query = "SELECT user_id FROM user_data WHERE user_name = '" +
request.getParameter("userID")
 + "' and user_password = '" + request.getParameter("password") +"'";
 try {
    Statement DBstatement = connection.createStatement( … );
    ResultSet results = DBstatement.executeQuery( query );
 }
```

Insights and Remarks

Based on the research and findings, it is clear that SQL injections cause complete breach of confidentiality, integrity, and availability, of data. The repercussions of SQL injections have caused severe financial damages, loss of compliance leading to fines and penalties, data breaches, and even bankruptcy. This problem continues to grow and shows no sign of being obsolete.

The work done so far in identifying mitigation methods and ways to prevent SQL injection attacks are very well developed and technically sound.

However, the reason for the existence and continued exploitation of this grave vulnerability is far more than technical limitations.

Software development lifecycles are drastically shortened in order to roll out solutions and quick deliverables. This ends up with untested applications being launched to the public vulnerable to be exploits. Organizations need to take security more seriously and invest time and budget in penetration testing and aggressive tiger team testing in order to ensure that the interface open to the public are thoroughly tested and completely safe.

Also systems rarely take stock of their legacy systems, and old unknown, un-updated, and unmaintained isolated pages can still be hosted on the website – and hackers only have to crawl through the portal, find it, and exploit it ("10 Reasons SQL Injection Still Works"). Organizations need to regularly take stock of each page, and each interface that is exposed on the DMZ, and ensure the legacy and retired pages are taken down. This minimizes the surface area of attack and helps the firm stay on top of their security. SQL injections are clever, simple, and easy to launch. An organization who exposes even a simple contact form on its portal should do all it can in terms of policies, secure software practices, secure coding, and through pen testing in order to safeguard themselves from exploits in the wild.

References

Understanding SQL Injection. (2016, December 16). Retrieved from http://www.cisco.com/c/en/us/about/security-center/sql-injection.html

Top 10 2013-Top 10. (n.d.). Retrieved from https://www.owasp.org/index.php/Top_10_2013-Top_10

Bright, P. (2016, October 28). How security flaws work: SQL injection. Retrieved from https://arstechnica.com/information-technology/2016/10/how-security-flaws-work-sql-injection/

Goodin, D. (2016, May 09). How a security pro's ill-advised hack of a Florida elections site backfired. Retrieved from https://arstechnica.com/security/2016/05/how-a-security-pros-ill-advised-hack-of-a-florida-elections-site-backfired/

Curmudgeon, C. (n.d.). SQLi Hall-of-Shame. Retrieved from http://codecurmudgeon.com/wp/sql-injection-hall-of-shame/

Cid, D. (2016, May 24). Website Attacks - SQL Injection And The Threat They Present. Retrieved from https://blog.sucuri.net/2014/10/website-attacks-sql-injection-and-the-threat-they-present.html

Curtis, S. (2012, January 19). Barclays: 97 percent of data breaches still due to SQL injection. Retrieved from http://www.techworld.com/news/security/barclays-97-percent-of-data-breaches-still-due-sql-injection-3331283/

SQL Injection Prevention Cheat Sheet. (n.d.). Retrieved from https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet

Chickowski, E. (2013, August 5). 10 Reasons SQL Injection Still Works. Retrieved from http://www.darkreading.com/application-security/database-security/10-reasons-sql-injection-still-works/d/d-id/1139702