# SQL Injection Attacks: What They Are and How to Secure Against Them

Jonathan B. Mcleod

East Carolina University

SQL Injection Attacks: What They Are and How to Secure Against Them

There are many threats these days that IT security professionals have to protect against when maintaining applications that are accessible over the Internet. One of the most common attacks against web applications is SQL injection attacks. SQL Injection is an application layer attack that takes advantage of security vulnerabilities in websites and applications, and when executed gives the hacker access to an underlying database. SQL injection (SQLi) attacks remained the dominant web attack vector in q4 2017. They made up 50% of all web application attacks in q4 (Akamia - State of the Internet, 2017). While attacks like denial of service attacks (DDOS) target specific servers, web application attacks tend to be attacks of opportunity. The vast majority of web application attacks are the result of untargeted scans looking for any vulnerable system, but a few are directed attempts to compromise a specific target (Akamia - State of the Internet, 2017).

Successfully implementing a SQL injection attack can allow the attacker to carry out actions against an applications' database that were not intended by the developer. These attacks remain prevalent due to the lack of safeguards put in place by companies developing and hosting the applications. Attackers will continue to utilize these vectors to gain access to systems if applications do not take the simple but necessary step of sanitizing data input and output. These types of attacks are easily automated and scalable, looking for any vulnerable system, rather than targeting specific organizations (Akamia - State of the Internet, 2017).

## What is a SQL Injection Attack?

When someone hears of an attack against a company's IT infrastructure the first thought is an attack against the network. SQL injection attacks can be successfully carried out against the

most hardened network security due to the fact they attack weaknesses in software applications instead of attacking network vulnerabilities. Akamai.com describes SQL injection as "SQL Injection is an application layer attack that takes advantage of security vulnerabilities in websites and applications, and when executed gives the hacker access to an underlying database. Along with Malware and DDoS, SQL Injection Attacks are one of the most common forms of cyber-security attacks" (Akamia SQL Injection Attack, n.d.). Techopedia.com describes a vulnerability as "Vulnerability is a cyber-security term that refers to a flaw in a system that can leave it open to attack. A vulnerability may also refer to any type of weakness in a computer system itself, in a set of procedures, or in anything that leaves information security exposed to a threat." (SQL Injection Attack, n.d.).

The United States is listed as the top country in the world for having internet-based attacks carried out against it with 32% of all attacks targeting the country (Akamia - State of the Internet, 2017).

| Country | Attacks Sourced | Percentage |
|---|---|---|
| **Unites States** | **128,013,378** | **32.0%** |
| **Netherlands** | **47,433,432** | **11.9%** |
| **China** | **28,171,775** | **7.1%** |
| **Brazil** | **22,945,844** | **5.7%** |
| **Russia** | **18,370,802** | **4.6%** |
| **Ukraine** | **17,182,960** | **4.3%** |
| **India** | **16,489,773** | **4.1%** |
| **Germany** | **13,046,096** | **3.3%** |
| **United Kingdom** | **12,790,735** | **3.2%** |
| **Canada** | **12,634,269** | **3.2%** |

Note. Data for Internet-based attacks carried out in 2017 from Akamia [State of the Internet] (2017)

One reason web applications are an easy target for attack is the broad availability of targets and developers focused more on designing a product for their end users and less focused on securing

an application against threats from attackers. Network engineers primarily focus their firewalls to protect organizations against network attacks, viruses, and malware and do not have the resources or funding to also secure web applications against coding vulnerabilities. It is vital to the health of every enterprise that secure coding practices become part of the larger landscape to combat vulnerabilities at the source (Akamia - State of the Internet, 2017).

## Types of SQL Injection Attacks

There are a number of different ways an attacker can carry out a SQL injection attack. Some attacks are used to gather information about the web application, server, and the database and use that information to exploit other weaknesses in the web application. These attacks can be grouped into four categories: SQL Manipulation, Code Injection, Function Call Injection and Buffer Overflows (Rana, K., 2011).

### SQL Manipulation

SQL Manipulation looks to manipulate the WHERE clause of a SQL query to have the web application return results not intended by the program. The UNION and INTERSECT operators can also be used with this type of attack. SQL Manipulation is considered the simplest attack of the four and is easy for novice hackers to carry out (Rana, K., 2011).

### Buffer Overflows

Buffer overflow vulnerabilities allow execution of malicious code that overwrites the memory fragments IP (Instruction Pointer), BP (Base Pointer) and other registers of the process, resulting in exceptions, segmentation faults, denial of service (Deepa, G., & Thilagam, P. S., 2016). This attack can be triggered by an attacker inputting data that is designed to alter the way

the web application functions. The goal of the attack is to cause the application to crash, return sensitive data or cause a breach in security (Rana, K., 2011).

### Code Injection

Code injection attacks look to take advantage of bugs caused by processing invalid data. This attack looks to add additional SQL statements to the application and can have disastrous consequences (Rana, K., 2011). These consequences can range from an attacker gaining access to sensitive data to modifying the database using INSERT, UPDATE and DELETE operators. Depending on the security settings of the database management system, the attacker could also execute administration operations on the database or run commands on the operating system itself (OWASP - SQL Injection, n.d.).

### Function Call Injection

When implementing a function call injection, the attacker attempts to insert database function calls into the SQL query statement. These function calls could make changes to the database or execute commands to the operating system (Rana, K., 2011).

## Ways SQL Injection Attacks Are Carried Out

### Tautology SQL Injection

A tautology attack is classified as a SQL manipulation attack and looks to inject code into a web application that will cause a SQL statement to always evaluate to true. This is most commonly used to bypass authentication pages and extract data the user otherwise wouldn't have access to. In this type of attack, the attacker looks to exploit an injectable field that is used in a

WHERE condition of a SQL query (Jhala, Kritarth & Shukla, Umang. 2017). Umang (2017) gives an example query of how this attack can be used to circumvent a login page below.

Take as an example a login form with a “username” field and a “password” field.

Username entry: a’ OR ‘1’ = ‘1

Password entry: a’ OR ‘1’ = ‘1

A typical SQL query would take the entries entered by the user and look to validate the data against users stored in a database. An example of a query that might pull this information would be:

```
SELECT name FROM users WHERE username = form.username and password =
form.password;
```

With the entries from the tautology attack added the entire query would look like:

```
SELECT name FROM users WHERE username = a’ OR ‘1’ and password = a’ OR
‘1’;
```

Due to adding the “OR ‘1’” clause into the SQL statement this statement will always evaluate to true and allow the attacker to access web pages secured behind the authentication page. This SELECT statement will return all users from the user's table and assign the first user record to the attacker. If this user is the admin account, then this would grant the attacker admin access to the website.

**End of Line SQL Injection**

An end of line attack is classified as a SQL manipulation attack and involves inserting a comment character in the middle of a program's SQL statement, causing the query code after the comment to be nullified. The following example code from the OWASP website (OWASP Comment Injection Attack, n.d.) shows how inserting a comment can allow an attacker to return all rows in a database table.

```
SELECT body FROM items WHERE id = $ID limit 1;
```

The attacker sends the following as the value of $ID:

```
1 OR 1=1; #
```

This results in the following query being executed:

```
SELECT body FROM items WHERE id = 1 OR 1=1; # limit 1;
```

The hash symbol is seen as the start of a comment, so the code after the hash isn't executed. This results in all rows being returned instead of the one row that is intended. The comment symbol used is different based on the database being used so an attacker might have to try multiple attempts if the backend database is unknown.

**Illegal/Logically Incorrect Query**

An Illegal/Logically Incorrect Query is classified as a SQL manipulation attack and is used by an attacker to gather information to use in additional attacks (Rana, K., 2011)**.** An attacker uses this technique to gain knowledge of the database through generating errors. These

errors can give away table names, column names or parameter values used in the queries. This information can be used in conjunction with other SQL injection attacks. Here is one example of such an attack.

SQL query string:

```
SELECT * FROM employees WHERE username = $POST[‘username’] AND password = $POST[‘password’]
```

Attacker submits the following for the username field:

```
sss”
```

Having the double quotation marks in the string causes an error in the query due to their not being a closing quotation mark. Here is an example error this might cause:

```
Result: “Incorrect syntax near ‘sss’. Unclosed quotation mark after the character string “ AND password =’’
```

This error lets the attacker know the column name for the table that stores user information is named password.

**SQL Injection using Union Query**

In SQL the UNION statement is used to join two queries into one statement. An attacker can exploit this by writing his own query and, using the UNION statement, attached it to a query that

is executed by the web application. Attackers use this technique to guide servers to return data that were not intended to be returned by the developers (Rana, K., 2011). A union attack example is given below using a query example from the OWASP website (OWASP - Testing for SQL Injection, n.d.).

Original query:

```
SELECT Name, Phone, Address FROM Users WHERE Id=$id
```

The attacker passes in the following value for $id:

```
1 UNION ALL SELECT creditCardNumber,1,1 FROM CreditCardTable
```

This creates the following query:

```
SELECT Name, Phone, Address FROM Users WHERE Id=1 UNION ALL SELECT
creditCardNumber,1,1 FROM CreditCardTable
```

Using the word ALL as part of the query allows the attacker to return every entry in the credit card table. An example URL where an attacker could use this exploit is given below:

```
http://www.merchant.com/product.php?id=1 UNION ALL SELECT
creditCardNumber,1,1 FROM CreditCardTable
```

In this example, an attacker viewing a product page from the merchant could pass in the UNION statement using the id parameter and have the query return all credit cards in the CreditCardTable.

**Piggy-backed Query**

A piggy-backed query is a code injection attack and is similar to the UNION query injection attack. The piggy-backed query looks to execute an additional query to the one intended by the web application. For this technic to work, it does require the server to be configured to allow for the execution of several different queries within a single string of code (Rana, K., 2011). This can also be referred to as stacking queries. This is normally carried out by placing a query ending character like a semicolon in the injection code and then adding another query after the ending character. A piggy-back attack example is given below using a query example from the OWASP website **[8]**.

Original intended query:

```
SELECT * FROM products WHERE id_product=$id_product
```

Code the attacker could insert to execute multiple queries:

```
10; INSERT INTO users (…)
```

Resulting query:

```
SELECT * FROM products WHERE id_product=10; INSERT INTO users (…)
```

The usage of the semicolon to end the web application's query and begin a new query would allow the attacker to insert a new user record into the database and potentially create a login for himself and gain access to areas of the website that require authentication. This method could also be used to carry out attacks against a website. Imagine if a query using "droptable users" was added as the additional query. It could render a website unusable and have a devasting impact if there wasn't a backup of the database.

## SQL Injection Attack Prevention Methods

SQL injection attacks are common due to the number of SQL vulnerabilities and how attractive targeting a database of pontifical customer or credit card data is (OWASP – SQL Injection Prevention Cheat Sheet, n.d.). As we have seen in the preceding examples, hackers have a number of tools to use against web applications with the SQL injection methods. Protecting against these attacks is critical for an organization that collects and stores sensitive information. Losing customer data, credit card information or health records would be detrimental and could put the organization out of business. Application developers and IT security personnel have a number of tools and best practices in their tool kit to protect against these attacks and minimize the damage if an attack is successful.

### Testing for SQL Injection Vulnerability

During the development of the web application, a developer can test his SQL statements for SQL injection vulnerabilities by submitting code an attacker would attempt to use. The simplest test to run would be attempting to submit an input that contained a single quote or a semicolon. The single quote is used by SQL to terminate a string and the semicolon is used to

terminate a SQL query. Another simple test a developer can do is try to submit a string to an input that is expecting a number. The web application should prevent this from throwing a SQL error that could give an attacker information about the data structure of your database. Many of the attacks listed previously can also be run by the developer while developing the application, so any errors thrown by the application can be fixed before it goes into production.

**Defending Against SQL Injection Attacks Using Prepared Statements**

Writing code that interacts with the database using prepared statements can prevent many of the SQL injection attacks an attacker might use. Using prepared statements prevents the attacker's input from changing how the query functions, even if the attacker inputs SQL commands. A prepared statement has the developer separate the query command from the input given by the user. This way anything entered by the user is only treated as data (dates, strings, integers etc) and is not seen as part of the query (OWASP – SQL Injection Prevention Cheat Sheet, n.d.). Using prepared statements is not something built into the database management system but is built into the programming language the developer uses to write the web application. An example of a prepared statement using PHP is shown below.

```
$stmt = $conn->prepare("INSERT INTO Users (firstname, lastname, email) VALUES (?, ?, ?)");
$stmt->bind_param("sss", $firstname, $lastname, $email);

$firstname = "Jerry";
$lastname = "Brown";
$email = "brownj@website.com";

$stmt->execute();
```

You can see in the prepared statement that the input data is not included as part of the query. The query in the prepared statement has place holders for the input values. The input values are assigned to variables and are then bound to the prepared statement.

**Defending Against SQL Injection Attacks Using Stored Procedures**

Stored Procedures are similar to prepared statements except the SQL query code is stored in the database itself instead of in the application. This makes the query code language independent and allows applications written in different languages to use the same queries. An example given on the OWASP website using Java is given below (OWASP – SQL Injection Prevention Cheat Sheet, n.d.).

```
String custname = request.getParameter("customerName"); // This should REALLY be validated
 try {
        CallableStatement cs = connection.prepareCall("{call
sp_getAccountBalance(?)}");
        cs.setString(1, custname);
        ResultSet results = cs.executeQuery();
        // … result set handling
 } catch (SQLException se) {
        // … logging and error handling
 }
```

The area to look at here is “sp_getAccountBalance(?)”. In the prepared statement example the query code was written as part of the PHP code. Here, Java is calling a procedure “sp_getAccountBalance(?)” that is stored in the database and passing a value into it. When Java calls “cs.executeQuery()” it returns data using the stored procedure query.

**Defending Against SQL Injection Attacks by Escaping User Input**

An application developer should consider using the methods listed above before using the method described here and should only use this method as a last resort. This method is typically used to retrofit legacy code when using the previous methods is not feasible due to costs (OWASP – SQL Injection Prevention Cheat Sheet, n.d.). An example is given below using PHP's "mysqli_real_escape_string()" function to escape special characters before they are inserted into the database.

```
$con = mysqli_connect("localhost","my_user","my_password","my_db");
$name = "1' OR 1=1";
mysqli_real_escape_string($con,$name);
mysqli_query($con,"SELECT * from emp WHERE name = '$name'");
mysqli_close($con);
```

The escape character for MySQL is a backslash, so the PHP function mysqli_real_escape_string() will add a backslash in front of any special characters. In the example, a backslash will be placed in front of the single quote in the input preventing the single quote from ending the WHERE statement and starting the OR statement.

**Web Application Firewall**

Most of the discussion of preventing SQL injection attacks have been on the developer side. Network security personnel have options to protect against these attacks on the network side. A Web Application Firewall (WAF) filters, monitors, and blocks HTTP/S traffic to and from a web application to protect against malicious attempts to compromise the system or exfiltrate data. By inspecting HTTP/S traffic, a WAF can prevent web application attacks before they access the web application server (F5 – Web Application Firewall, n.d.). A web application firewall functions similarly to a network firewall by inspecting the web traffic to the application

server and looking for patterns that might show a threat against the server. WAF's can look for Cross-site scripting, SQL injection, Layer 7 DoS, Brute force and web scraping attacks.

## Summary

We have seen a number of ways an attacker can use SQL injection attacks against web applications that have not been hardened to prevent them. With SQL injection attacks being the number one attack against web applications it becomes critical that developers focus on securing their applications against SQL injection vulnerabilities. Attackers have a number of tools at their disposal for carrying out these attacks, and many of these can be carried out by novices using automated processes. Developers and network personnel have tools at their disposal to both prevent these attacks by following correct code writing procedures and by using detection appliances like web application firewalls.

References

*Deepa, G., & Thilagam, P. S. (2016). Securing web applications from injection and logic vulnerabilities: Approaches and challenges. Information and Software Technology, 74, 160-180. doi:10.1016/j.infsof.2016.02.005

*Rana, K. (2011). Classification of SQL injection attacks and using encryption as A countermeasure. *International Journal of Advanced Research in Computer Science, 2*(1) Retrieved from http://search.proquest.com.jproxy.lib.ecu.edu/docview/1443705484?accountid=10639

*Jhala, Kritarth & Shukla, Umang. (2017). Tautology based Advanced SQL Injection Technique A Peril to Web Application. Retrieved from https://www.researchgate.net/publication/321865053_Tautology_based_Advanced_SQL_Injection_Technique_A_Peril_to_Web_Application

OWASP - Comment Injection Attack (n.d.) Cheat Sheet Retrieved from https://www.owasp.org/index.php/Comment_Injection_Attack

OWASP - Testing for SQL Injection (n.d.) Cheat Sheet Retrieved from https://www.owasp.org/index.php/Testing_for_SQL_Injection_(OTG-INPVAL-005)

OWASP - SQL Injection (n.d.) Retrieved from https://www.owasp.org/index.php/SQL_Injection

Akamia [State of the Internet] / Security Q4 2017 Report (2017) Retrieved from https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/q4-2017-state-of-the-internet-security-report.pdf

Akamia SQL Injection Attack. (n.d.). Retrieved from https://www.akamai.com/us/en/resources/sql-injection-attack.jsp

SQL Injection Attack. (n.d.). Retrieved from https://www.akamai.com/us/en/resources/sql-injection-attack.jsp

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

F5 – Web Application Firewall (n.d.) Cheat Sheet Retrieved from https://www.f5.com/services/resources/glossary/web-application-firewall