Chapter 52
Prevention of SQL Injection Attacks in Web Browsers

Kannan Balasubramanian
Mepco Schlenk Engineering College, India

ABSTRACT

Applications that operate on the Web often interact with a database to persistently store data. For example, if an e-commerce application needs to store a user's credit card number, they typically retrieve the data from a Web form (filled out by the customer) and pass that data to some application or script running on the company's server. The dominant language that these database queries are written in is SQL, the Structured Query Language. Web applications can be vulnerable to a malicious user crafting input that gets executed on the server. One instance of this is an attacker entering Structured Query Language (SQL) commands into input fields, and then this data being used directly on the server by a Web application to construct a database query. The result could be an attacker's gaining control over the database and possibly the server. Care should be taken to validate user input on the server side before user data is used.

INTRODUCTION

Web applications are becoming more sophisticated and increasingly technically complex. They range from dynamic Internet and intranet portals, such as e-commerce sites and partner extranets, to HTTP-delivered enterprise applications such as document management systems and ERP applications. The availability of these systems and the sensitivity of the data that they store and process are becoming critical to almost all major businesses, not just those that have online e- commerce stores. Web applications and their supporting infrastructure and environments use diverse technologies and can contain a significant amount of modified and customized code. The very nature of their feature-rich design and their capability to collate, process, and disseminate information over the Internet or from within an intranet makes them a popular target for attack. Also, since the network security technology market has matured and there are fewer opportunities to breach information systems through network based vulnerabilities, hackers are increasingly switching their focus to attempting to compromise applications.

DOI: 10.4018/978-1-5225-3422-8.ch052

SQL injection is an attack in which SQL code is inserted or appended into application/user input parameters that are later passed to a back-end SQL server for parsing and execution (Clarke, 2009; Pauli, 2013). Any procedure that constructs SQL statements could potentially be vulnerable, as the diverse nature of SQL and the methods available for constructing it provide a wealth of coding options. The primary form of SQL injection consists of direct insertion of code into parameters that are concatenated with SQL commands and executed. A less direct attack injects malicious code into strings that are destined for storage in a table or as metadata. When the stored strings are subsequently concatenated into a dynamic SQL command, the malicious code is executed. When a Web application fails to properly sanitize the parameters which are passed to dynamically created SQL statements (even when using parameterization techniques) it is possible for an attacker to alter the construction of back-end SQL statements. When an attacker is able to modify an SQL statement, the statement will execute with the same rights as the application user; when using the SQL server to execute commands that interact with the operating system, the process will run with the same permissions as the component that executed the command (e.g., database server, application server, or Web server), which is often highly privileged.

To illustrate this, let's return to the previous example of a simple online retail store. If you remember, we attempted to view all products within the store that cost less than $100, by using the following URL.

This time, however, you are going to attempt to inject your own SQL commands by appending them to the input parameter *val*. You can do this by appending the string 'OR '1'= '1 to the URL:

```
 http://www.victim.com/products.php?val=100' OR '1'='1
```

This time, the SQL statement that the PHP script builds and executes will return all of the products in the database regardless of their price. This is because you have altered the logic of the query. This happens because the appended statement results in the *OR* operand of the query always returning *true*, that is, 1 will always be equal to 1. Here is the query that was built and executed:

```
SELECT *
FROM ProductsTbl
WHERE Price < '100.00' OR '1'='1'
ORDER BY ProductDescription;
```

The preceding simple example demonstrates how an attacker can manipulate a dynamically created SQL statement that is formed from input that has not been validated or encoded to perform actions that the developer of an application did not foresee or intend. The example, however, perhaps does not illustrate the effectiveness of such a vulnerability; after all, we only used the vector to view all of the products in the database, and we could have legitimately done that by using the application's functionality as it was intended to be used in the first place. What if the same application can be remotely administered using a content management system (CMS)? A CMS is a Web application that is used to create, edit, manage, and publish content to a Web site, without having to have an in-depth understanding of the ability to code in HTML. You can use the following URL to access the CMS application:

```
http://www.victim.com/cms/login.php?username=foo&password=bar
```