Caroline Smith

Dr. Chunyu Ai

Database Implementation, Application and Administration

30 October 2022

SQL Injection Attacks

Cybercrime is ever-increasing with the advancement of technology in society today. While many recognize the potential of technology to benefit society greatly, others recognize its potential for less innocent purposes. Some individuals use their knowledge of cybersecurity and code to carry out cyberattacks. These attacks have become an unforeseen part of warfare, used on scales as large as taking down entire internet service provider and postal systems during the war between Russia and Ukraine (Swan). A common motivator in cyberattacks is acquisition of data, which is often carried out through SQL injection attacks.

One of the most common methods of SQL injection involves utilizing "unsanitized" user input, such as usernames, passwords, or bank account information to create an always-true SQL statement that will query a database. For example, with a username prompt with no constraints, an attacker would be able to input a username as " or '1' = '1'." The query delivered to the database will appear as:

*"SELECT * FROM bankAccountInfo WHERE username = " + enteredUsername + ";"*

In this case, the attacker's input *" or '1' = '1'"* works with application code to build the query. Since "'*1' = '1*'" always evaluates to *True,* the attacker will have successfully altered the query to give them access to the information in the "*bankAccountInfo*" table of the database, which, in this case, can be a schema such as:

CREATE TABLE bankAccountInfo (

accNum int primary key;

balance double;

fName char(100);

lName char(100);

phoneNum varchar(10)

addr varchar(100)

);

Another type of SQL injection attack is referred to as "blind" SQL injection. In this case, an attacker cannot perform an SQL injection attack as mentioned above since the contents of the database are not displayed to the attacker after querying. Error messages, however, are displayed onto the webpage or application window. Typically, the attacker will evaluate responses (*TRUE* or *FALSE*) and/or time taken to return responses after querying the database, which shows vulnerabilities (Nera). For example, a normal query to display information on a webpage about a product may appear as:

"SELECT pName, pId, price FROM product WHERE pId = [selected product from site]"

where "*product*" is the table name, "*pName*" is the full name of the product, and "*price*" is the price of the product. An attacker can then insert a payload that returns a *False* result from the database (such as "*7 = 8*"), and then a payload that returns a *True* result from the database (such as "*1 = 1"*). This *True* return value indicates vulnerabilities within the database. Using the same logic, an attacker can initiate a time-based blind SQL injection by inserting a "*sleep(number of seconds)"* statement into the payload. The vulnerability is exposed if the "*sleep*" in the statement

is heeded and the response from the database is delayed by the specified number of seconds. This method is typically more complex for the attacker (Nera).

Unlike other cyberattacks, such as Denial of Service attacks, that are preventable by improved network security, SQL injection attacks can be prevented by improved security of the source code of a web application or website. The simplest way to ensure security within code is keeping frameworks, libraries, and server software up to date. Vendors implement important patching in newer software versions than previous versions. Using old versions of software, especially for organizations that house highly sensitive data, leaves that data at much greater risk of being seized by individuals or organizations with malicious intent. In addition, applications and sites should contain error handling and reporting that is not displayed directly to the frontend of the application. Attackers' ability to view details of a database error is a major contributor to the success of a blind SQL injection attack. Finally, in terms of code security, source code should be implemented to only accept the expected data types for input fields, and input fields should be kept to a minimum. Alternatives such as *Yes* or *No* (Boolean value) options, selection of options through buttons, and dropdown menus minimize the risk of an attacker attempting to use an open input field to carry out an SQL injection attack.

Another major factor contributing to the level of vulnerability of a database is the users accessing the database. Strict access control and the principle of least privilege for accounts that require access to the database should be implemented by the database administrator to ensure that users can only perform the necessary actions for their needs or job functions. For example, an employee of an organization who only needs to view the contents of a database (but not alter it) should only have the privilege to *SELECT* from that database; another employee from one department should only have access to *INSERT*, *DELETE*, or *UPDATE* tables that pertain to their

respective department and job function; and a high-up employee, such as a CEO, should not be able to *INSERT*, *DELETE*, or *UPDATE* a table that is below their "class" in the organization's hierarchy. These roles in an organization and utilization of a multilevel system minimize potential for tampering and breaches in security due to human error.

Increasing amounts of data are moving away from paper and into technology-based resources. With this change comes major responsibility of developers, database administrators, and cybersecurity professionals to ensure that private data, whether it be an individual's social security number or a government's highly sensitive data, is kept private.

Works Cited

Besic, Nera. “Blind SQL Injection: How It Works, Examples and Prevention.” *Bright Security*, 16 May 2022, https://brightsec.com/blog/blind-sql-injection/.

“How to Protect against SQL Injection Attacks.” *How to Protect Against SQL Injection Attacks | Information Security Office*, https://security.berkeley.edu/education-awareness/how-protect-against-sql-injection-attacks.

SQL Injection, https://www.w3schools.com/sql/sql_injection.asp.

Swan, David. “Cyberwarfare: Russia vs Ukraine.” *CSCIS*, https://cscis.org/2022/03/20/cyberwarfare-russia-vs-ukraine/.