|  |  |
| --- | --- |
| Semester | T.E. Semester VI – Computer Engineering |
| Subject | Cryptography and cyber security |
| Subject Professor In-charge | Prof. Amit Nerurkar |
| Assisting Teachers | Prof. Amit Nerurkar |
| Laboratory | M312B |

|  |  |
| --- | --- |
| Student Name | Deep Salunkhe |
| Roll Number | 21102A0014 |
| TE Division | A |

**Title:**

**Simulation of SQL Injection (PBLE-1)**

**Explanation:**

SQL injection (SQLi) is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. This can allow an attacker to view data that they are not normally able to retrieve. This might include data that belongs to other users, or any other data that the application can access. In many cases, an attacker can modify or delete this data, causing persistent changes to the application's content or behavior.

In some situations, an attacker can escalate a SQL injection attack to compromise the underlying server or other back-end infrastructure. It can also enable them to perform denial-of-service attacks.

A screenshot of a computer

Description automatically generated

**Implementation:**

**1.Retrieving hidden data**

Imagine a shopping application that displays products in different categories. When the user clicks on the **Gifts** category, their browser requests the URL:

https://insecure-website.com/products?category=Gifts

This causes the application to make a SQL query to retrieve details of the relevant products from the database:

SELECT \* FROM products WHERE category = 'Gifts' AND released = 1

This SQL query asks the database to return:

* all details (\*)
* from the products table
* where the category is Gifts
* and released is 1.

The restriction released = 1 is being used to hide products that are not released. We could assume for unreleased products, released = 0.

The application doesn't implement any defenses against SQL injection attacks. This means an attacker can construct the following attack, for example:

https://insecure-website.com/products?category=Gifts'--

This results in the SQL query:

SELECT \* FROM products WHERE category = 'Gifts'--' AND released = 1

Crucially, note that -- is a comment indicator in SQL. This means that the rest of the query is interpreted as a comment, effectively removing it. In this example, this means the query no longer includes AND released = 1. As a result, all products are displayed, including those that are not yet released.

You can use a similar attack to cause the application to display all the products in any category, including categories that they don't know about:

https://insecure-website.com/products?category=Gifts'+OR+1=1--

This results in the SQL query:

SELECT \* FROM products WHERE category = 'Gifts' OR 1=1--' AND released = 1

The modified query returns all items where either the category is Gifts, or 1 is equal to 1. As 1=1 is always true, the query returns all items.

A screenshot of a computer

Description automatically generatedA screenshot of a website

Description automatically generated

**2. Subverting application logic**

Imagine an application that lets users log in with a username and password. If a user submits the username wiener and the password bluecheese, the application checks the credentials by performing the following SQL query:

SELECT \* FROM users WHERE username = 'wiener' AND password = 'bluecheese'

If the query returns the details of a user, then the login is successful. Otherwise, it is rejected.

In this case, an attacker can log in as any user without the need for a password. They can do this using the SQL comment sequence -- to remove the password check from the WHERE clause of the query. For example, submitting the username administrator'-- and a blank password results in the following query:

SELECT \* FROM users WHERE username = 'administrator'--' AND password = ''

This query returns the user whose username is administrator and successfully logs the attacker in

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA white rectangle with blue lines

Description automatically generatedA screenshot of a computer

Description automatically generated

**3. Retrieving data from other database tables**

In cases where the application responds with the results of a SQL query, an attacker can use a SQL injection vulnerability to retrieve data from other tables within the database. You can use the UNION keyword to execute an additional SELECT query and append the results to the original query.

For example, if an application executes the following query containing the user input Gifts:

SELECT name, description FROM products WHERE category = 'Gifts'

An attacker can submit the input:

' UNION SELECT username, password FROM users--

This causes the application to return all usernames and passwords along with the names and descriptions of products.

A screenshot of a computer

Description automatically generatedA screenshot of a website

Description automatically generatedA close-up of a chat box

Description automatically generatedA screenshot of a web page

Description automatically generated