



TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE TLAXIACO

GESTIÓN DE PROYECTO DE SOFTWARE

PRACTICA 4. INYECCIÓN SQL

CARRERA:

INGENIERIA EN SISTEMAS COMPUTACIONALES

GRUPO: 7US

NOMBRE DE LOS INTEGRANTES DE EQUIPO:

LUZ ARLETH LOPEZ BAUTISTA -21620036

SAUL LOPEZ BAUTISTA - 21620073

DOCENTE

ING. EDWARD OSORIO SALINAS

Tlaxiaco, Oax., 01 de octubre del 2024.



"Educación, ciencia y tecnología, progreso día con día"®

Contenido

1. CREAR UNA BASE DE DATOS CON UNA TABLA QUE CONTenga AL MENOS 3 REGISTROS.....	4
2. CREAR UNA APLICACIÓN WEB QUE PERMITA BUSCAR UN REGISTRO POR SU ID, NOMBRE O DESCRIPCIÓN.....	5
CODIGO DE NUESTRA APLICACIÓN	5
INTERFAZ DE NUESTRA APLICACIÓN	6
CODIGO VULNERABLE PARA INYECCIÓN DE CODIGO MALICIOSO.	7
3. REALIZAR PRUEBAS DE INYECCIÓN DE CÓDIGO EN LA APLICACIÓN WEB.	8
CODIGO INYECTADO POR CODIGO MALICIOSO.	8
RESULTADOS DE LA INYECCION DE CODIGO MALICIOSO A TRAVES DE LA INTERFAZ.	10
COMO PREVENIR DICHO ATAQUE.....	11
TEMAS A INVESTIGAR.....	13
1. Inyección de SQL (SQL Injection).....	13
2. Blind SQL Injection (Inyección SQL Ciega)	13
3. SQL Injection basada en errores (Error-based SQL Injection)	13
4. SQL Injection basada en tiempo (Time-based SQL Injection)	13
5. SQL Injection en procedimientos almacenados	14
6. SQL Injection en ORM (Object-Relational Mapping).....	14
7. Herramientas para detectar y prevenir SQL Injection	14

ILUSTRACIONES

Ilustración 1 Creación de la base de datos.....	4
Ilustración 2 datos.....	4
Ilustración 3 creación de la interfaz	5
Ilustración 4 re direccionamiento	5
Ilustración 5 ingreso de un registro	6
Ilustración 6 Resultados de búsqueda	6
Ilustración 7 Conexión a nuestra base de datos	7
Ilustración 8 Búsqueda.....	7
Ilustración 9 Consulta vulnerable.....	7
Ilustración 10 impresión de resultados de consulta	8
Ilustración 11 Cierre del programa	8
Ilustración 12 Conexión a nuestra base de datos	9
Ilustración 13 Comprobación de búsqueda	9
Ilustración 14 Código inyectable	9
Ilustración 15 Resultado de consulta	10
Ilustración 16 ingresar dato	10
Ilustración 17 Búsqueda por el usuario.....	10
Ilustración 18 Consulta arrojando datos no solicitados	11
Ilustración 19 Creación de parámetros.....	11
Ilustración 20 mejora de seguridad en la consulta con prepare.....	11
Ilustración 21 Marcadores de posición	11
Ilustración 22 ejecución de consultas preparadas	11
Ilustración 23 Conjunto de resultados	12
Ilustración 24 Mostrar resultado de consulta.....	12
Ilustración 25 cierre del programa.....	12

1. CREAR UNA BASE DE DATOS CON UNA TABLA QUE CONTENGA AL MENOS 3 REGISTROS.

Paso 1. Creamos nuestra base de datos que ocuparemos para la inyección de código en este caso creamos una base de datos con el nombre Juego el cual contiene 3 registros solicitados.

```
CREATE DATABASE IF NOT EXISTS `juego_db`;  
  
USE `juego_db`;  
  
CREATE TABLE IF NOT EXISTS `personajes` (  
  `id` int(11) NOT NULL AUTO_INCREMENT,  
  `nombre` varchar(100) DEFAULT NULL,  
  `descripcion` text DEFAULT NULL,  
  PRIMARY KEY (`id`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;  
  
INSERT INTO `personajes` (`id`, `nombre`, `descripcion`) VALUES  
(1, 'Guerrero', 'Un fuerte luchador con una espada.'),  
(2, 'Mago', 'Un maestro de la magia elemental.'),  
(3, 'Arquero', 'Un arquero experto con una precisión mortal.');
```

Ilustración 1 Creación de la base de datos

Paso 2. Podemos visualizar los campos que contiene nuestra base de datos en Xampp.



The screenshot shows the XAMPP database management interface. At the top, there are tabs for 'Perfilando', 'Editar en línea', 'Editar', 'Explicar SQL', 'Crear código PHP', and 'Actualizar'. Below these, there are controls for 'Mostrar todo', 'Número de filas' (set to 25), 'Filtrar filas' (with a search box), and 'Ordenar seg'. An 'Opciones extra' button is also present. The main table has columns 'id', 'nombre', and 'descripcion'. It contains three rows of data: a Warrior (Guerrero), a Mage (Mago), and an Archer (Arquero). Each row has action buttons for 'Editar', 'Copiar', and 'Borrar'. At the bottom, there is a 'Seleccionar todo' checkbox and a summary for the selected elements with 'Editar', 'Copiar', and 'Borrar' options.

	id	nombre	descripcion
<input type="checkbox"/>	1	Guerrero	Un fuerte luchador con una espada.
<input type="checkbox"/>	2	Mago	Un maestro de la magia elemental.
<input type="checkbox"/>	3	Arquero	Un arquero experto con una precisión mortal.

Ilustración 2 datos

2. CREAR UNA APLICACIÓN WEB QUE PERMITA BUSCAR UN REGISTRO POR SU ID, NOMBRE O DESCRIPCIÓN.

- Esta aplicación debe ser vulnerable a inyección de código, esto significa que si el usuario ingresa un valor malicioso en el campo de búsqueda, la aplicación debe mostrar información que no debería ser accesible o permitir realizar acciones que no deberían ser posibles.

CODIGO DE NUESTRA APLICACIÓN

Paso 3. Creamos nuestra aplicación el cual consiste en buscar un nombre de un personaje que se encuentre registrado en nuestra base de datos creado en xampp.

```
1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4   <meta charset="UTF-8">
5   <title>Buscar Personaje</title>
6 </head>
7 <body>
8   <h1>Buscar Personaje</h1>
```

Ilustración 3 creación de la interfaz

Paso 4. Nuestra aplicación contiene el archivo re direccionado que se llama vulne.php el cual contiene nuestra conexión a nuestra base de datos vulnerable para cualquier atacante, la búsqueda nos mostrara el id, el nombre y la descripción del personaje buscado.

```
9 <form action="vulne.php" method="get">
10   <label for="busqueda">Buscar por ID, Nombre o Descripción:</label>
11   <input type="text" id="busqueda" name="busqueda" required>
12   <input type="submit" value="Buscar">
13 </form>
14 </body>
15 </html>
```

Ilustración 4 re direccionamiento

INTERFAZ DE NUESTRA APLICACIÓN

Paso 5. Nos dirigimos al navegador y ejecutamos nuestra aplicación, como vemos se aprecia nuestra interfaz de la aplicación en este caso tenemos como un pequeño buscador de personaje, nosotros como usuarios ingresamos un nombre de alguno de los personajes que se encuentra en la base de datos, ingresamos el nombre del personaje Guerrero y después le damos clic en buscar.

Buscar Personaje

Buscar por ID, Nombre o Descripción:

Ilustración 5 ingreso de un registro

Paso 6. Podemos observar que nos mostró un resultado de búsqueda que se encuentra almacenado en nuestra base de datos.

ID: 1 - Nombre: Guerrero - Descripción: Un fuerte luchador con una espada.

Ilustración 6 Resultados de búsqueda

CODIGO VULNERABLE PARA INYECCIÓN DE CODIGO MALICIOSO.

Paso 7. Aquí se realizamos una conexión a nuestra base de datos MySQL mediante mysqli. Si la conexión falla, se muestra un mensaje de error y el script se detiene.

```
<?php
$servidor = "localhost";
$usuario = "root";
$contrasena = "";
$basedatos = "juego_db";

$conn = new mysqli($servidor, $usuario, $contrasena, $basedatos);

if ($conn->connect_error) {
    die("Conexión fallida: " . $conn->connect_error);
}
```

Ilustración 7 Conexión a nuestra base de datos

Paso 8. Creamos un parámetro búsqueda utilizando el método \$_GET, lo que significa que el valor puede ser cualquier cosa que el usuario proporcione a través de la URL. Lo cual si el usuario ingresa código SQL malicioso, este será directamente utilizado en la consulta, lo que deja la puerta abierta a inyecciones SQL.

```
$busqueda = $_GET['busqueda'];
```

Ilustración 8 Búsqueda

Paso 9. Agregamos una consulta SQL utilizando el valor de \$busqueda directamente dentro de la cadena de consulta. Aquí es donde se presenta la vulnerabilidad de inyección SQL. Dado que el valor de \$busqueda no es filtrado o escapado, un atacante podría inyectar código SQL en este parámetro, lo que permitiría modificar la consulta.

```
/* Consulta vulnerable a inyección de SQL */
$sql = "SELECT * FROM personajes WHERE nombre LIKE '%$busqueda%' OR descripcion LIKE '%$busqueda%' OR id = '$busqueda'";
```

Ilustración 9 Consulta vulnerable

Paso 10. Si la consulta devuelve resultados, estos se muestran en pantalla. Aunque esta parte del código no presenta un problema directo, los datos devueltos podrían haber sido manipulados si la consulta original fue alterada por una inyección.

```
if ($resultado->num_rows > 0) {  
    while ($fila = $resultado->fetch_assoc()) {  
        echo "ID: " . $fila["id"]. " - Nombre: " . $fila["nombre"]. " - Descripción: " . $fila["descripcion"]. "<br>";  
    }  
} else {  
    echo "No se encontraron resultados";  
}
```

Ilustración 10 impresión de resultados de consulta

Paso 11. Cerramos la conexión a la base de datos una vez que se ha finalizado la operación.

```
$conn->close();  
?>
```

Ilustración 11 Cierre del programa

3. REALIZAR PRUEBAS DE INYECCIÓN DE CÓDIGO EN LA APLICACIÓN WEB.

CODIGO INYECTADO POR CODIGO MALICIOSO.

Paso 12. Reutilizamos nuestro código de la conexión haciéndole unos pequeños cambios, tenemos de primera lo que es la conexión a nuestra base de datos

```
$servidor = "localhost";  
$usuario = "root";  
$contrasena = "";  
$basedatos = "juego_db";  
  
$conn = new mysqli($servidor, $usuario, $contrasena, $basedatos);  
  
if ($conn->connect_error) {  
    die("Conexión fallida: " . $conn->connect_error);  
}
```

Ilustración 12 Conexión a nuestra base de datos

Paso 13. En este apartado realizamos un cambio, donde comprobamos si el parámetro búsqueda fue proporcionado a través de la URL (\$_GET). Si está definido, se almacena en la variable \$busqueda.

```
if (isset($_GET['busqueda'])) {  
    $busqueda = $_GET['busqueda'];  
}
```

Ilustración 13 Comprobación de búsqueda

Paso 14. En este apartado ya está la inyección SQL realizado por el atacante donde la consulta incluye OR '1'='1', lo que asegura que la condición de búsqueda siempre se evalúe como verdadera, lo que hace que todos los registros de la tabla se devuelvan, independientemente del valor proporcionado en el parámetro búsqueda. Como consecuencia tenemos que el resultado es que la consulta no realiza realmente una búsqueda sobre la base de datos, sino que devuelve toda la información disponible en la tabla personaje, donde un atacante podría manipular fácilmente el SQL para acceder a datos confidenciales o ejecutar otras consultas dañinas, como eliminar registros.

```
$sql = "SELECT * FROM personajes WHERE nombre LIKE '%' OR '1'='1%' OR descripcion LIKE '%' OR '1'='1%' OR id = '' OR '1'='1'";
```

Ilustración 14 Código inyectable

Paso 15. Si la consulta devuelve resultados (lo que sucederá siempre debido a la inyección), se muestran los datos de los personajes en formato HTML.

```
if ($resultado->num_rows > 0) {  
    while ($fila = $resultado->fetch_assoc()) {  
        echo "ID: " . $fila["id"]. " - Nombre: " . $fila["nombre"]. " - Descripción: " . $fila["descripcion"]. "<br>";  
    }  
} else {  
    echo "0 resultados";  
}
```

Ilustración 15 Resultado de consulta

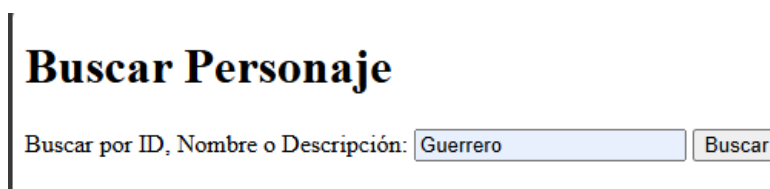
Paso 16. Si no se define el parámetro búsqueda en la URL, se muestra un mensaje pidiendo que se ingrese un término de búsqueda.

```
} else {  
    echo "Por favor, ingresa un término de búsqueda.";  
}
```

Ilustración 16 ingresar dato

RESULTADOS DE LA INYECCION DE CODIGO MALICIOSO A TRAVES DE LA INTERFAZ.

Paso 17. Volvemos a re dirigirnos a la interfaz de nuestra aplicación e ingresamos en busqueda el nombre de Guerrero.



Buscar Personaje

Buscar por ID, Nombre o Descripción:

Ilustración 17 Búsqueda por el usuario

Paso 18. Podemos observar que el atacante puede acceder y obtener más información no solicitada, eso quiere comprobar que la inyección de código fue un éxito.

ID: 1 - Nombre: Guerrero - Descripción: Un fuerte luchador con una espada.
ID: 2 - Nombre: Mago - Descripción: Un maestro de la magia elemental.
ID: 3 - Nombre: Arquero - Descripción: Un arquero experto con una precisión mortal.

Ilustración 18 Consulta arrojando datos no solicitados

COMO PREVENIR DICHO ATAQUE

Paso 19. Aquí se verifica si el parámetro búsqueda fue proporcionado en la URL. Si existe, se guarda en la variable \$busqueda. Usamos `real_escape_string()` para sanitizar la entrada del usuario. Esto asegura que cualquier carácter especial que pueda ser utilizado para inyecciones SQL (como comillas simples ', comillas dobles ", o caracteres de control) sea tratado de manera segura.

```
if (isset($_GET['busqueda'])) {  
    $busqueda = "%" . $conn->real_escape_string($_GET['busqueda']) . "%";  
}
```

Ilustración 19 Creación de parámetros

Paso 20. En lugar de construir la consulta SQL directamente con la entrada del usuario (lo que puede ser peligroso), aquí se utiliza una consulta preparada con el método `prepare()`.

```
$stmt = $conn->prepare("SELECT * FROM personajes WHERE nombre LIKE ? OR descripcion LIKE ? OR id = ?");
```

Ilustración 20 mejora de seguridad en la consulta con prepare

Paso 21. Creamos la función `bind_param()` se utiliza para enlazar los valores reales que serán usados en los marcadores de posición ? de la consulta preparada.

```
$stmt->bind_param("sss", $busqueda, $busqueda, $_GET['busqueda']);
```

Ilustración 21 Marcadores de posición

Paso 22. Se ejecuta la consulta preparada de manera segura con los valores proporcionados. Como se usan marcadores de posición, cualquier intento de inyección SQL es neutralizado, ya que los valores se tratan como datos, no como parte del código SQL.

```
$stmt->execute();
```

Ilustración 22 ejecución de consultas preparadas

Paso 23. Obtenemos el conjunto de resultados de la consulta mediante `get_result()`. Esto devuelve un objeto de resultados con el cual se pueden procesar las filas devueltas por la consulta.

```
$resultado = $stmt->get_result();
```

Ilustración 23 Conjunto de resultados

Paso 24. Si la consulta devuelve resultados, estos se muestran. El ciclo `while` recorre cada fila obtenida de la base de datos, mostrando el id, nombre, y descripción de cada personaje.

```
if ($resultado->num_rows > 0) {  
    while ($fila = $resultado->fetch_assoc()) {  
        | echo "ID: " . $fila["id"]. " - Nombre: " . $fila["nombre"]. " - Descripción: " . $fila["descripcion"]. "<br>";  
        | }  
    } else {  
        | echo "0 resultados";  
    }  
}
```

Ilustración 24 Mostrar resultado de consulta

Paso 25. Cierre del programa

```
$conn->close();
```

Ilustración 25 cierre del programa

TEMAS A INVESTIGAR

1. Inyección de SQL (SQL Injection)

La inyección de SQL es una vulnerabilidad de seguridad que ocurre cuando un atacante inserta o inyecta código SQL malicioso en una consulta que se ejecuta en una base de datos. Esto sucede cuando las aplicaciones no validan correctamente las entradas de usuario, permitiendo que comandos SQL no autorizados se ejecuten con privilegios elevados.

2. Blind SQL Injection (Inyección SQL Ciega)

La inyección SQL ciega es un tipo de ataque en el que el atacante no recibe directamente los resultados de la consulta inyectada en la base de datos, pero aún puede inferir información valiosa basándose en los comportamientos o respuestas de la aplicación, como cambios en los tiempos de respuesta, mensajes de error genéricos o cambios en el contenido de la página.

3. SQL Injection basada en errores (Error-based SQL Injection)

En este tipo de inyección SQL, el atacante aprovecha los mensajes de error que devuelve la base de datos para obtener información sensible. Muchas bases de datos proporcionan detalles específicos cuando ocurre un error, lo que puede incluir la estructura de las tablas, nombres de columnas u otra información interna.

4. SQL Injection basada en tiempo (Time-based SQL Injection)

La inyección SQL basada en tiempo es una variante de la inyección SQL ciega, donde el atacante utiliza funciones de retraso de tiempo (como ``SLEEP()`` en MySQL) para deducir si una consulta es verdadera o falsa basándose en la demora en la respuesta de la aplicación.

Este método se utiliza cuando no hay mensajes de error visibles ni cambios observables en el contenido de la página, pero el atacante puede notar cambios en los tiempos de respuesta.

5. SQL Injection en procedimientos almacenados

Los procedimientos almacenados son bloques de código SQL predefinidos y almacenados en la base de datos que se ejecutan como una unidad. Si estos procedimientos almacenados no gestionan adecuadamente las entradas del usuario, también pueden ser vulnerables a inyecciones SQL.

6. SQL Injection en ORM (Object-Relational Mapping)

Los ORMs son herramientas que permiten a los desarrolladores interactuar con bases de datos utilizando objetos en lugar de escribir directamente consultas SQL. Sin embargo, si el ORM no se usa adecuadamente o permite la inserción de parámetros directamente en las consultas, es posible realizar una inyección SQL.

7. Herramientas para detectar y prevenir SQL Injection

Existen diversas herramientas y prácticas para detectar y prevenir inyecciones de SQL:

a) Detección:

1. sqlmap: Una de las herramientas más populares para la automatización de la detección y explotación de vulnerabilidades de inyección SQL.

2. Burp Suite: Herramienta utilizada por expertos en seguridad para pruebas de penetración, que incluye módulos para detectar inyecciones SQL.

3. OWASP ZAP (Zed Attack Proxy): Otro conjunto de herramientas populares para pruebas de seguridad web, que también tiene capacidades para detectar inyecciones SQL.

b) Prevención:

1. Parameterized Queries (Consultas preparadas): Las consultas parametrizadas separan los datos de las consultas SQL, evitando que las entradas maliciosas se interpreten como comandos SQL.

2. ORM: Utilizar un ORM adecuado que gestione de forma segura las consultas a la base de datos.

3. WAF (Web Application Firewall): Un firewall de aplicaciones web puede ayudar a prevenir ataques comunes, incluidos los ataques de inyección SQL.

4. Validación y sanitización de entradas: Validar y limpiar todas las entradas de usuario para asegurarse de que no contengan comandos SQL maliciosos.

5. Escapar adecuadamente los caracteres: Utilizar las funciones de escape proporcionadas por el lenguaje de programación o la base de datos para evitar que se ejecuten consultas maliciosas.

CONCLUSION

En el proceso de creación de una base de datos, desarrollo de una aplicación web y la realización de pruebas de inyección de código SQL, se destaca la importancia de comprender y mitigar las vulnerabilidades de seguridad. Inicialmente, se creó una base de datos sencilla con tres registros y una aplicación vulnerable a inyecciones SQL, donde un atacante podría explotar las fallas en la validación de entradas para obtener información no autorizada o modificar el comportamiento del sistema. Durante las pruebas de inyección, se demostró cómo, al no aplicar medidas de seguridad, es posible manipular las consultas SQL para acceder a todos los datos de la tabla, sin respetar las condiciones de búsqueda originales. Esto subraya lo crucial que es evitar la construcción de consultas SQL directamente con entradas de usuarios, ya que facilita el acceso a información sensible, comprometiendo la integridad y confidencialidad de la base de datos. Finalmente, la implementación de medidas preventivas, como el uso de consultas preparadas y la sanitización de entradas con funciones como `real_escape_string()`, demostró ser una solución efectiva para evitar estos ataques. Estas prácticas aseguran que las entradas de los usuarios sean tratadas como datos y no como código ejecutable, eliminando así la posibilidad de que un atacante manipule las consultas.

BIBLIOGRAFIA

[SQL Injection Prevention - OWASP Cheat Sheet Series](#)

[The Ultimate SQL Injection Survival Guide. From Detection to Defense & Beyond. - zenarmor.com](#)

<https://www.bing.com/ck/a?!&&p=e26ee24dca563610JmltdHM9MTcyNzc0MDgwMCZpZ3VpZD0yNTQyZjNhNi1mNjFiLTU5NWQtMWI4ZS1lMDZhZjc0YzY4NDQmaW5zaWQ9NTI1MA&ptn=3&ver=2&hsh=3&fclid=2542f3a6-f61b-695d-1b8e-e06af74c6844&psq=gu%C3%ADa+de+inyecci%C3%B3n+SQL.&u=a1aHR0cHM6Ly9raW5zdGEuY29tL2VzL2Jsb2cvaW55ZWVjaW9uLXNxbC8&ntb=1>