Hola futuros hackers!

Cuando alguien intenta foguearse y pulsa el botón, está enviando esa información a una sentencia SQL que será ejecutada contra la base de datos de la aplicación, una instrucción que podría ser similar a esta de acá:

$sql = SELECT \* FROM usuarios WHERE usuario = ‘$usuario’ and password = ‘$pass’;

Aquí “$usuario” y “$pass” serían las variables que contienen los valores introducidos por el visitante.

El código del profe es una SQL Injection. La instrucción que ejecuta se salta esta autenticación cumpliendo una condición que siempre será TRUE. La SQL Injection comprueba en la tabla usuarios si hay alguien registrado con un determinado username y si su contraseña es una en especifico o bien que 1 sea igual a 1. Como esto segundo siempre es cierto, logaríamos entrar al sistema con el usuario sin saber su verdadera contraseña.

**SQL Injection:** Try Hack Me > SQL Injection Lab > Broken Authentication 2

En el login escribir lo siguiente: ' or 1=1 -- - Pasword: cualquiervalor

**SQL Injection para Login Bypass**

La inyección SQL (SQL Injection) es una técnica de ataque en la que un atacante puede ejecutar comandos SQL maliciosos a través de una entrada de una aplicación web, con el objetivo de comprometer la seguridad de la base de datos.

**Ejemplo de Login Bypass con SQL Injection**

Vamos a suponer que tenemos una página de login con los siguientes campos:

* **Usuario**: username
* **Contraseña**: password

El código SQL que la aplicación podría estar utilizando para verificar las credenciales sería algo así:

sql

Copy code

SELECT \* FROM users WHERE username = 'input\_username' AND password = 'input\_password';

El atacante puede intentar inyectar código SQL malicioso en estos campos de entrada para alterar la consulta original. Un ejemplo común es introducir algo como:

* **Usuario**: ' OR '1'='1
* **Contraseña**: ' OR '1'='1

La consulta SQL resultante sería:

sql

Copy code

SELECT \* FROM users WHERE username = '' OR '1'='1' AND password = '' OR '1'='1';

Debido a que '1'='1' siempre es verdadero, la consulta devolverá todos los registros de la tabla users, permitiendo al atacante acceder al sistema sin credenciales válidas.

**Otros Ejemplos Comunes**

1. **Inyección de Comentarios**:
   * **Usuario**: admin' --
   * **Contraseña**: cualquier cosa
   * Esto convierte la consulta en:

sql

Copy code

SELECT \* FROM users WHERE username = 'admin' -- ' AND password = 'cualquier cosa';

* + El -- comenta el resto de la consulta, ignorando la parte de la contraseña.

1. **Inyección de Condición Falsa**:
   * **Usuario**: admin' AND '1'='2
   * **Contraseña**: cualquier cosa
   * Esto convierte la consulta en:

sql

Copy code

SELECT \* FROM users WHERE username = 'admin' AND '1'='2' AND password = 'cualquier cosa';

* + Como '1'='2' es falso, no devuelve ningún registro, pero demuestra cómo se puede manipular la lógica de la consulta.

**Medidas de Mitigación**

Para protegerse contra SQL Injection, se deben seguir estas prácticas:

1. **Usar Consultas Preparadas y Parámetros**:
   * Las consultas preparadas con parámetros en lenguajes como PHP, Java, y .NET son inmunes a la inyección SQL.
   * Ejemplo en PHP con PDO:

php

Copy code

$stmt = $pdo->prepare('SELECT \* FROM users WHERE username = :username AND password = :password');

$stmt->execute(['username' => $input\_username, 'password' => $input\_password]);

$user = $stmt->fetch();

1. **Validación y Saneamiento de Entrada**:
   * Validar y limpiar todas las entradas del usuario.
   * Utilizar funciones de escape adecuadas para cada base de datos.
2. **Usar ORM (Object-Relational Mapping)**:
   * Los ORM, como Hibernate o Entity Framework, generan consultas SQL de manera segura.
3. **Menos Privilegios**:
   * Minimizar los privilegios de las cuentas de la base de datos utilizadas por la aplicación.
4. **Monitoreo y Auditoría**:
   * Implementar mecanismos de detección y respuesta a inyecciones SQL.

**Ejemplo de Código Inseguro y Seguro**

**Inseguro:**

php

Copy code

$username = $\_POST['username'];

$password = $\_POST['password'];

$query = "SELECT \* FROM users WHERE username = '$username' AND password = '$password'";

$result = mysqli\_query($conn, $query);

**Seguro:**

php

Copy code

$username = $\_POST['username'];

$password = $\_POST['password'];

$stmt = $conn->prepare('SELECT \* FROM users WHERE username = ? AND password = ?');

$stmt->bind\_param('ss', $username, $password);

$stmt->execute();

$result = $stmt->get\_result();

Implementar medidas de seguridad es fundamental para proteger las aplicaciones web contra ataques de inyección SQL y otros vectores de ataque similares.