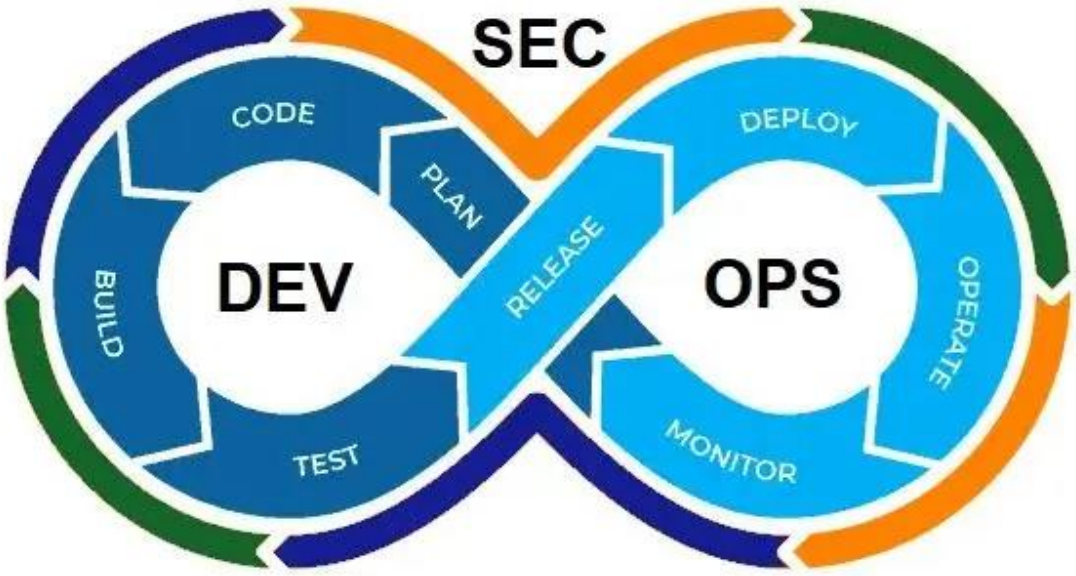
IMMUNE Technology Institute

Bootcamp Ciberseguridad

PRÁCTICA SECDEVOPS



Mateo Guerrero Serrano

1. IDEA DE LA APLICACIÓN

La idea es hacer una página web de un equipo de fútbol 7 de amigos, donde se puedan ver los resultados del equipo, consultar la clasificación en la liga o comprobar cuántos goles lleva cada jugador del equipo.

2. TECNOLOGÍA DE LA APLICACIÓN

La página web está desarrollada con los lenguajes HTML, CSS y PHP, este último para gestionar el acceso a una BBDD (MySQL) de donde se obtendrán los datos que se muestran en la página web.

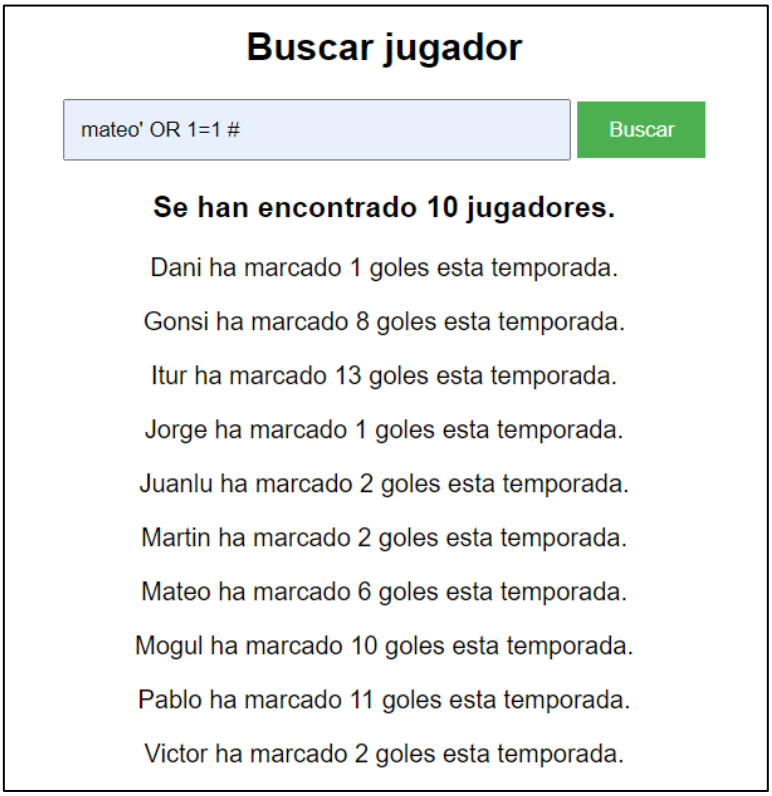
3. VULNERABILIDAD A SUBSANAR

La página web tiene un buscador en su página principal para consultar el nombre de un jugador y que devuelva el número de goles que ha marcado. Ese buscador está hecho con un formulario, y lo introducido en él se busca en la BBDD.

En un principio, tal y como está desarrollado el código, la web sería vulnerable a una inyección SQL. Aquí vemos la parte del código vulnerable:

```php
$sql = "SELECT nombre, goles FROM jugadores WHERE nombre LIKE '%" . $jugador ."%'";
$resultado = mysqli_query($conexion, $sql);
```

Y aquí comprobamos como podríamos hacer una inyección de código SQL:



Lo introducido no debería darme ningún resultado y, sin embargo, me devuelve todas las entradas de la tabla. Esto es debido a que la consulta no está parametrizada, ya que se concatena directamente la variable “$jugador” en la sentencia sin sanitizar ni parametrizar los datos.

4. SUBSANACIÓN DE LA VULNERABILIDAD

Para parametrizar la consulta y evitar así ataques de inyección SQL, se va a utilizar el siguiente código:

```php
// Con esta parte del código estaría subsanada esa vulnerabilidad, ya que la consulta estaría parametrizada.
$jugador = "%{$jugador}%";
$sql = "SELECT nombre, goles FROM jugadores WHERE nombre LIKE ?";
$statement = mysqli_prepare($conexion, $sql);
mysqli_stmt_bind_param($statement, "s", $jugador);
mysqli_stmt_execute($statement);
$resultado = mysqli_stmt_get_result($statement);
```

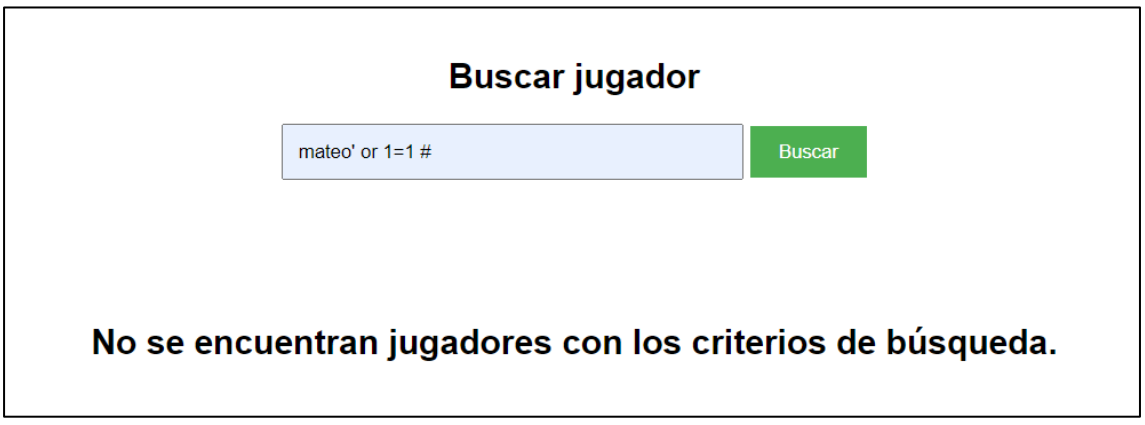
En este código, se utiliza una consulta preparada con un marcador de posición (?) en lugar de concatenar directamente la variable “$jugador”.

Después, se utiliza la función “mysqli_stmt_bind_param” para vincular el valor de “$jugador” al marcador de posición de manera segura.

Por último, “mysqli_stmt_execute()” se encarga de ejecutar la consulta preparada y “mysqli_stmt_get_result()” obtiene el resultado de la consulta para su posterior manipulación.

Este enfoque de parametrización asegura que el valor de “$jugador” se trate como un dato literal en lugar de parte de la estructura de la consulta. Como resultado, se evita la inyección SQL, ya que cualquier intento de manipular ese valor no afectará la lógica de la consulta y no se interpretará como código SQL malicioso.

A continuación, vemos como con el nuevo código la vulnerabilidad ha sido subsanada:



5. SUBIDA DEL CÓDIGO AL REPOSITORIO EN GITHUB

La aplicación desarrollada está subida en el siguiente repositorio de Github:

https://github.com/matguese/Fut7

Para subir todos los archivos al repositorio, se ha utilizado la terminal de GIT:

```
Mateo@DESKTOP-4V3G6JH MINGW64 /c/xampp/htd
ocs/secdevops/fut7 (master)
$ git push "https://github.com/matguese/Fu
t7.git" master
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 8 threads
Compressing objects: 100% (9/9), done.
Writing objects: 100% (9/9), 2.61 KiB | 1.30 MiB/s, done.
Total 9 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), done.
To https://github.com/matguese/Fut7.git
 * [new branch]      master -> master
```