

# **Informe de ataque a “Catálogo de plantas”**

Galder García

Entrega 1 - Versión 1 (2022-12-18)

# Índice de contenidos

Índice de contenidos	2
Introducción	3
Plan de ataque	4
Primer contacto con el sistema objetivo	5
Comienza el ataque	6
El problema	7
La solución	7
Nuevo ataque	8
Conclusión	9

# Introducción

Esta documentación es a la parte final del proyecto práctico de la asignatura Sistemas de Gestión de Seguridad de la Información.

El objetivo de esta práctica consiste en realizar un ataque a uno de los sistemas web creados por otros compañeros para el proyecto de esta asignatura. En concreto, se atacará explotando las vulnerabilidades de la primera versión del proyecto.

En nuestro caso, el sistema web que nos ha tocado atacar es “Catálogo de plantas” del grupo formado por Hugo Ran, Paula Pinto y Oier Diez; alojado en el repositorio de github: [https://github.com/huranech/docker-lamp/tree/Entrega\\_1](https://github.com/huranech/docker-lamp/tree/Entrega_1)

# Plan de ataque

Para atacar a nuestra víctima nos aprovecharemos de que no se sanitizan correctamente los parámetros introducidos por el usuario antes de ejecutar las instrucciones SQL en la base de datos.

El plan es realizar un ataque de *inyección SQL* y alterar la base de datos del sistema sin tener permisos para ello.

Cuando el usuario rellena un formulario y lo envía, se recogen los parámetros introducidos en los campos y se ejecuta en la base de datos una instrucción SQL con estos parámetros. La instrucción SQL tendrá el siguiente formato (o similar):

```
INSERT INTO users (usuario, contraseña, email)
VALUES ('$usuario', '$contraseña', '$email');
```

Los parámetros *\$usuario*, *\$contraseña*, y *\$email* tendrán los datos introducidos por el usuario, por ejemplo: *Galder*, *1234*, y *galder@correo.com*, respectivamente.

Si en uno de los parámetros introducimos una comilla simple, un cierre de paréntesis y un punto y coma, por ejemplo: En *\$email*, tendremos *galder@correo.com')*;. Al sustituir los parámetros en la instrucción SQL obtenemos:

```
INSERT INTO users (usuario, contraseña, email)
VALUES ('Galder', '1234', 'galder@correo.com'););
```

La parte roja provocará un “syntax error”, por eso añadimos también: espacio, dos guiones, y otro espacio: *–* . Con esto hacemos que lo que venga después quede como comentario y no produzca errores:

```
INSERT INTO users (usuario, contraseña, email)
VALUES ('Galder', '1234', 'galder@correo.com'); –');
```

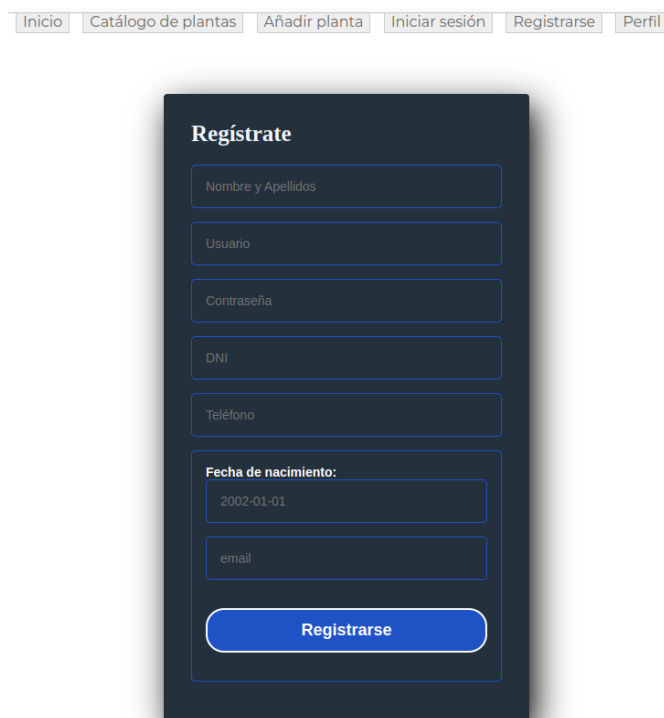
Ahora tenemos la instrucción original terminada con punto y coma, y el resto comentado. Podemos escribir entre el punto y coma y el comentario otras instrucciones que queramos que se ejecuten en la base de datos. Por ejemplo, en la variable *\$email*: *galder@correo.com')*; *DROP TABLE users*; *–* .

```
INSERT INTO users (usuario, contraseña, email)
VALUES ('Galder', '1234', 'galder@correo.com'); DROP TABLE users; –');
```

Con esto se ejecutarán dos instrucciones: Primero la instrucción original que inserta un nuevo usuario en la tabla *users*, y acto seguido la instrucción maliciosa que borra toda la tabla *users*.

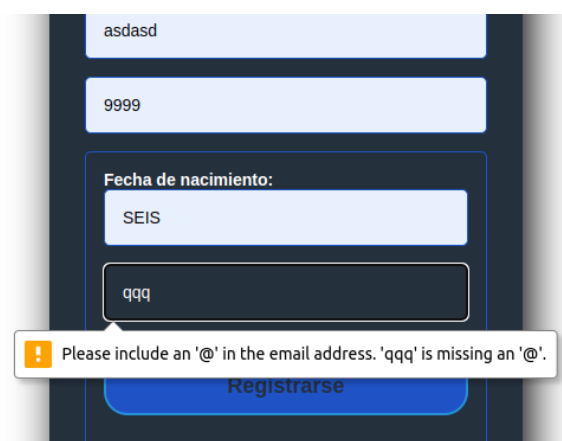
## Primer contacto con el sistema objetivo

Tras descargar y desplegar la web objetivo, abrimos el apartado de registro de usuario (**Imagen 1**) donde planeamos ejecutar nuestro ataque. Nos encontramos con un formulario en el que debemos introducir: Nombre (y apellidos), nombre de usuario, contraseña, número (y letra) de DNI, teléfono, fecha de nacimiento, y email. Procedemos a probar algunas combinaciones para familiarizarnos con el sistema.



**Imagen 1**

Rápidamente nos damos cuenta de que algunos campos requieren un formato específico que, en caso de no cumplirse, nos impiden enviar el formulario. Por ejemplo el email tiene que tener formato de email (**Imagen 2**). Otros campos, aunque permiten enviar el formulario y no sale ningún aviso, detectan anomalías en el formato y no se completa el registro en la base de datos. Por ejemplo si el nombre contiene números o el teléfono contiene letras.



**Imagen 2**

## Comienza el ataque

Comenzamos el ataque intentando inyectar “DROP TABLES” a través del campo “usuario” (**Imagen 3**), ya que sabemos que para el nombre se comprueba que solo contenga letras, pero para usuario no. El resto de campos que requieren un formato específico son rellenados cumpliendo el formato, de lo contrario no se efectuará el registro y no podremos inyectar el código malicioso.



The image shows a registration form titled "Regístrate" on a dark background. The form contains several input fields with the following values: "Galder" in the first field, a SQL injection payload in the second field, three dots in the third field, "79138114-J" in the fourth field, "666666666" in the fifth field, "2000-12-12" in the date field, and "galder@gmail.com" in the email field. A blue "Registrarse" button is at the bottom.

Field	Value
Nombre	Galder
Usuario	';";";";"); DROP TABLES; --
Apellido	...
DNI	79138114-J
Código	666666666
Fecha de nacimiento	2000-12-12
Correo electrónico	galder@gmail.com

**Imagen 3**

Ejecutamos el ataque pero no ocurre nada, la base de datos sigue en pie.

Probamos a cambiar el número y el orden de los parámetros, por si en la instrucción SQL no están en el mismo orden que en el formulario, y a introducir valores en los parámetros por si en la base de datos tienen activado el flag “NOT NULL”. Sin éxito.

Cambiamos la instrucción a inyectar por otras como UPDATE o DELETE, probamos en otros formularios como “iniciar sesión”, “añadir planta” o “modificar planta”. Nada funciona.

## El problema

Después de realizar múltiples pruebas de todo tipo y seguir sin resultados. Pensamos que el problema podría estar en la forma en la que la web envía la instrucción SQL a la base de datos.

Tras una breve investigación sobre el tema, descubrimos que la función *mysqli\_query()* de php solo permite ejecutar una única instrucción SQL, por lo que no podemos ejecutar el INSERT y el DROP TABLES a la vez.

## La solución

Aunque no podamos ejecutar nuevas instrucciones SQL, aún podemos modificar la instrucción original de formas que no estaban previstas en el diseño original del sistema.

Pensamos un nuevo ataque: Cambiar la instrucción INSERT para saltarnos las comprobaciones de formato que se realizan en el código e insertar valores inesperados en sitios que no corresponden.

## Nuevo ataque

En este ataque vamos a reemplazar los valores que se obtienen de los campos del formulario por otros que no cumplan con los formatos requeridos, de la siguiente manera:

**Nombre:** Como va delante del campo usuario se mantendrá igual.

**Usuario:** No requiere formato especial, lo usaremos para hacer la inyección.

**Contraseña:** No requiere formato especial, pero podemos cambiarla con la inyección.

**DNI:** Requiere formato de número (y letra) de DNI, introduciremos la palabra "NO".

**Teléfono:** Requiere formato de número de teléfono, introduciremos la palabra "nokia".

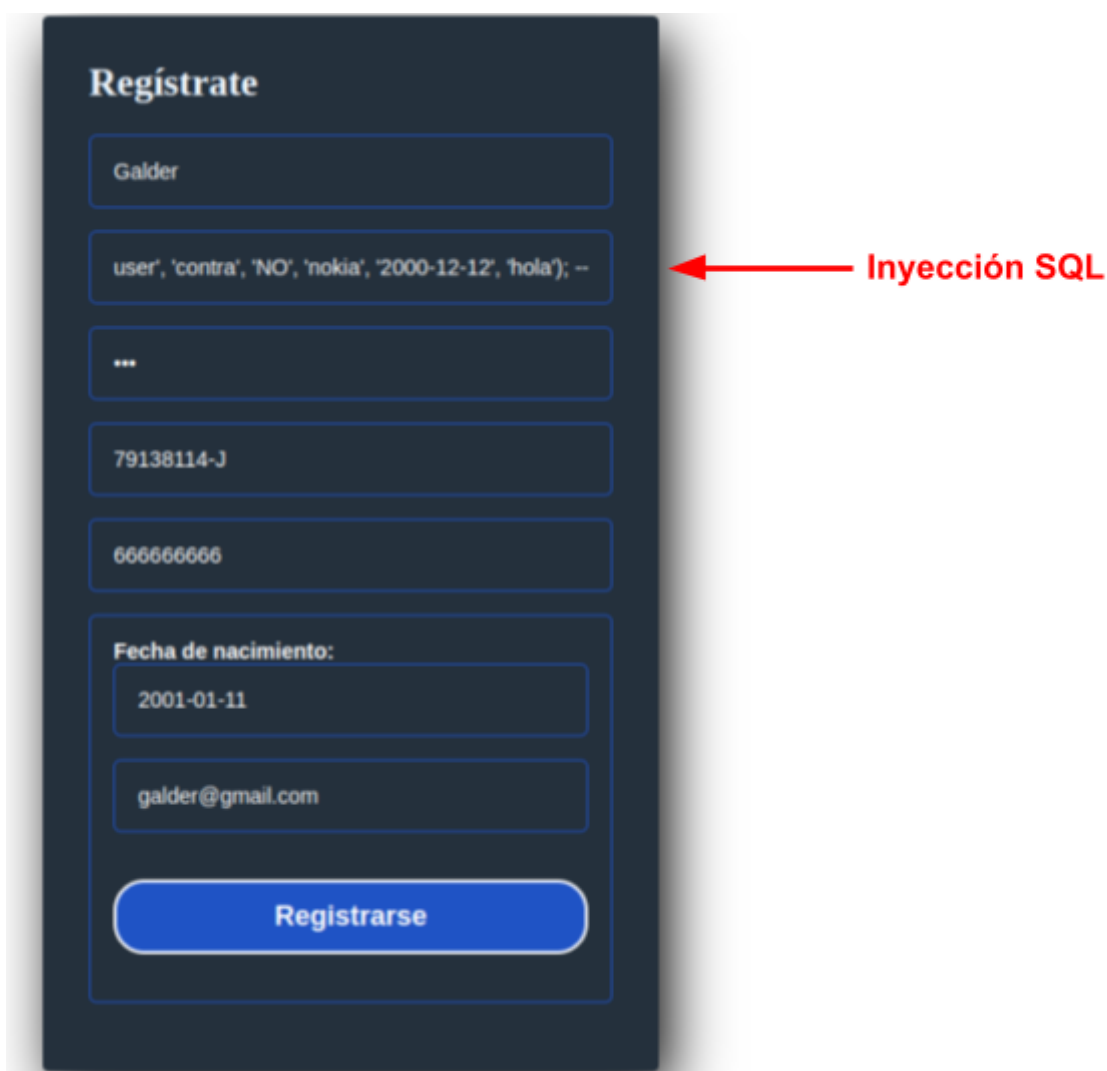
**FechaNacimiento:** La base de datos requiere un tipo date, la cambiaremos por otra fecha.

**Email:** Requiere formato de correo electrónico, introduciremos la palabra "hola".

Para lograr esto, vamos a introducir valores que cumplan con el formato indicado en todos los campos menos en el campo usuario, en el que introduciremos lo siguiente (**Imagen 4**):

**user', 'contra', 'NO', 'nokia', '2000-12-12', 'hola');** --

Los valores de los otros campos se descartarán y se sustituirán por los introducidos en la inyección.



The image shows a registration form titled "Regístrate" on a dark blue background. The form contains several input fields: a name field with "Galder", a password field containing the SQL injection payload `user', 'contra', 'NO', 'nokia', '2000-12-12', 'hola');` --, a field with three asterisks, a DNI field with "79138114-J", a phone field with "666666666", a date of birth section with the label "Fecha de nacimiento:" and a field with "2001-01-11", and an email field with "galder@gmail.com". A blue "Registrarse" button is at the bottom. A red arrow points from the text "Inyección SQL" to the password field.

Imagen 4



Ejecutamos el ataque y, como podemos comprobar al mirar en la base de datos (**Imagen 5**), el ataque ha sido un éxito. Los valores están en la base de datos, incumpliendo el diseño original y burlando las comprobaciones del código.

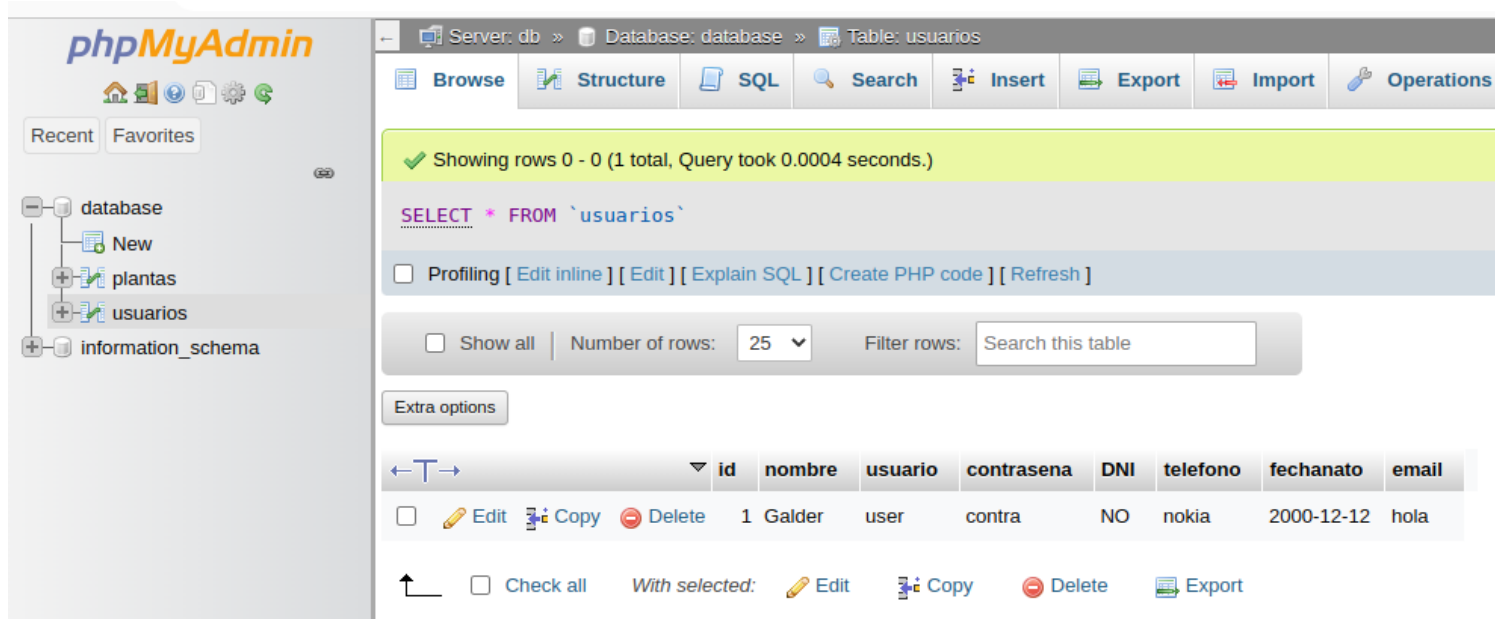


Imagen 5

## Conclusión

A pesar de las claras vulnerabilidades del sistema, el haber usado como base tecnología “moderna” como php 5.0+ y la función `mysqli_query()` ha hecho que se reduzca de manera significativa el daño potencial del ataque.

Aún así hemos conseguido aprovechar las vulnerabilidades y causar daños menores en el sistema con una simple cadena de texto en el lugar adecuado.