



## SQL Injection Básica

### Detalles Prácticos:

Descripción: Identificar usuarios almacenados en la base de datos

Nivel(Básico/Intermedio o avanzado)

Instructores:Rieradipe

### Tabla de contenidos:

Introducción	Contexto del taller
	Objetivos generales y específicos
Materiales Necesarios	Software requeridos
	Recursos Adicionales
Metodología	Desglose paso a paso del proceso
	Practicas recomendadas
Ejercicios Prácticos	Captura de solicitudes
	Análisis y pruebas
Resultados y Evaluación	Resultados esperados de las actividades
	Criterios de evaluación
Conclusión	Resumen de aprendizajes
	Preguntas y próximos pasos

### Objetivo del Laboratorio

El objetivo principal de este lab es identificar usuarios almacenados en la base de datos mediante técnicas de **inyección SQL**

### 🔍 Análisis de la vulnerabilidad.

- La página vulnerable se encuentra en el archivo i0x01.php, donde se presenta un campo de búsqueda de usuarios

### [Labs](#) / Injection 0x01

#### User search

- La aplicación construye internamente una consulta SQL; ej:

```
SELECT * FROM users WHERE username = '[alba]'
```

Esto permite **inyectar código malicioso** directamente desde el campo

### **Prueba de inyección:**

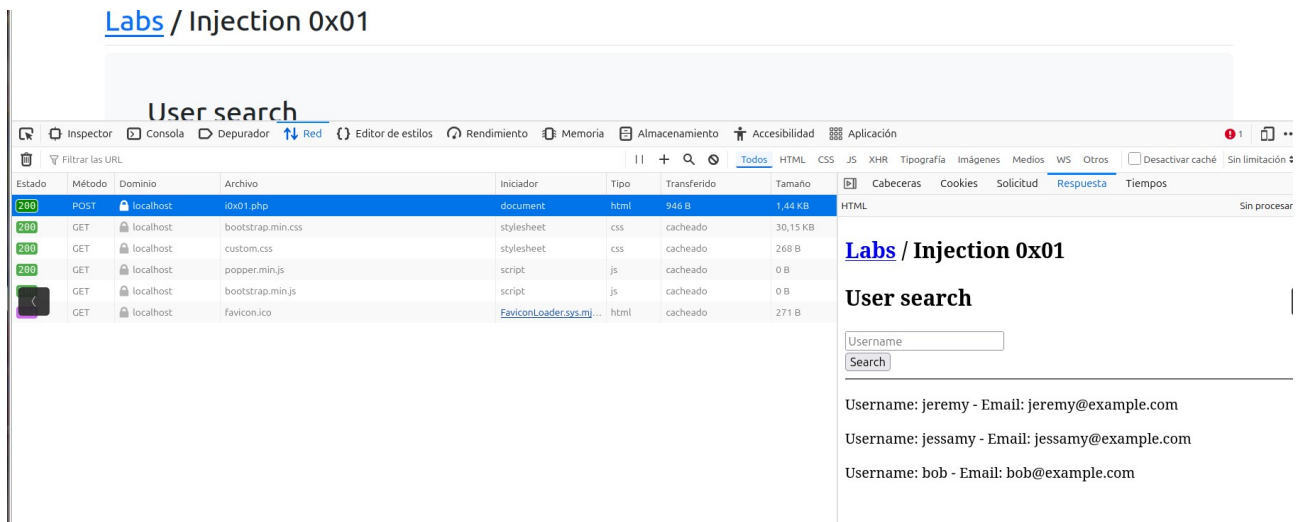
Se introduce en el campo de búsqueda el siguiente payload:

```
jeremy' OR '1'='1
```

Este payload **rompe la query** y fuerza una condición siempre verdadera  $1=1$ , lo que provoca que la bbdd devuelva **todos los registros**

### **Resultado observado:**

Tras enviar el formulario, la aplicación devolvió correctamente todos los usuarios.



The screenshot shows a web browser window with the title "Labs / Injection 0x01". The main content area displays a "User search" form with a "Username" input field and a "Search" button. Below the form, the search results are listed:

- Username: jeremy - Email: jeremy@example.com
- Username: jessamy - Email: jessamy@example.com
- Username: bob - Email: bob@example.com

The browser's developer tools are open, showing the network tab. The network tab displays a list of requests:

Estado	Método	Dominio	Archivo	Iniciador	Tipo	Transferido	Tamaño	Acciones
200	POST	localhost	index.php	document	html	946 B	1,44 KB	
200	GET	localhost	bootstrap.min.css	stylesheet	css	cacheado	30,15 KB	
200	GET	localhost	custom.css	stylesheet	css	cacheado	268 B	
200	GET	localhost	popper.min.js	script	js	cacheado	0 B	
200	GET	localhost	bootstrap.min.js	script	js	cacheado	0 B	
200	GET	localhost	favicon.ico	FaviconLoader.sys.mi...	html	cacheado	271 B	

Esto confirma que:

1. La aplicación es vulnerable a **SQL injection**
2. Se ha conseguido superar el laboratorio

### **Conclusión:**

Se ha explotado correctamente una vulnerabilidad de inyección SQL

El input del usuario no estaba correctamente validado ni espaciado, permitiendo modificar la lógica de la consulta SQL original y acceder a todos los registros de usuarios

### Recomendaciones para prevenir SQL injection.

1	<b>Usar consultas preparadas</b> que separan los datos de la lógica SQL (prepared statements)
2	<b>No concatenar</b> directamente los datos de usuario en la consulta
3	<b>Validar y sanitizar</b> el input, permitiendo <b>solo</b> caracteres seguros y esperados
4	Aplicar el <b>principio de mínimos privilegios</b> : el usuario de base de datos solo debe tener acceso a lo estrictamente necesario

Estas prácticas mitigan de forma efectiva los ataques de inyección SQL y protegen los datos sensibles del sistema.