🛠️ Ejercicio Práctico

📌 **Título:** Detección y verificación de una vulnerabilidad SQLi en DVWA

🎯 Objetivo:

Identificar una posible vulnerabilidad de **inyección SQL** en la aplicación DVWA y comprobar su existencia mediante la introducción manual de un payload simple.

🧱 Escenario:

La empresa te ha solicitado revisar la seguridad básica de su aplicación web de pruebas (DVWA). Deberás analizar si el formulario de consulta de usuarios permite la manipulación de sentencias SQL desde su entrada de texto.

🔍 Instrucciones:

✅ Paso 1 – Acceso al entorno

1. Abre la aplicación DVWA desde: `http://localhost/dvwa`
2. Inicia sesión con las credenciales:
    - **Usuario:** `admin`
    - **Contraseña:** `password`
3. Cambia el nivel de seguridad a **Low** desde la opción "DVWA Security".

✅ Paso 2 – Análisis del módulo vulnerable

1. Accede al módulo **"SQL Injection"** desde el menú lateral.
2. Observa el campo de entrada para buscar información de un usuario por ID.

✅ Paso 3 – Realiza una prueba simple

En el campo, en lugar de escribir un número, ingresa el siguiente texto:

1' OR '1'='1

1.
2. Haz clic en “Submit” o “Enviar”.

✅ Paso 4 – Observa el resultado

- ¿La aplicación devolvió múltiples registros de usuarios en vez de uno solo?
- ¿Ocurrió un error visible de base de datos?
- ¿El sistema respondió con más información de la esperada?

📋 Entregables:

1. Captura de pantalla del comportamiento observado.
2. Breve explicación del resultado obtenido.
3. Conclusión sobre si existe o no una vulnerabilidad de inyección SQL.
4. Una recomendación para prevenir este tipo de fallas.

🧠 Preguntas guía:

- ¿Qué efecto tuvo el texto ingresado en la consulta?
- ¿Qué pasaría si un atacante usara esto para acceder a información confidencial?

- ¿Cómo se podría evitar esta situación desde el código?

✅ Solución Modelo – Ejercicio Práctico

📌 Detección y verificación de una vulnerabilidad SQLi en DVWA

🔍 Paso 1 – Acceso y configuración

- Se accedió correctamente a la aplicación **DVWA** desde `http://localhost/dvwa`.
- Inicio de sesión realizado con las credenciales:
    - Usuario: `admin`
    - Contraseña: `password`
- En la opción **DVWA Security**, el nivel de seguridad fue configurado en **Low** para permitir pruebas básicas de explotación.

🧪 Paso 2 – Análisis del módulo SQL Injection

- Desde el menú lateral, se accedió al módulo **"SQL Injection"**.
- Se identificó un formulario que solicita un **User ID** y muestra información relacionada al número ingresado.

🧪 Paso 3 – Prueba de vulnerabilidad

- En lugar de ingresar un número de ID, se utilizó el siguiente payload:

1' OR '1'='1

- Tras enviar la solicitud, el sistema respondió con **una lista de múltiples usuarios**, en lugar de uno solo.

✅ Resultado observado

- La lógica de la consulta SQL fue alterada exitosamente.
- El payload convirtió la condición en siempre verdadera (`' 1 ' = ' 1 '`), lo que provocó que la base de datos devolviera **todos los registros posibles**.

📸 Captura de pantalla (simulada):

(Aquí se mostraría una imagen donde el sistema lista múltiples usuarios sin restricción alguna.)

📘 Explicación del resultado

- El campo de entrada **no realiza ningún tipo de sanitización ni validación**.
- El sistema **concatena directamente la entrada del usuario a la consulta SQL**, sin uso de parámetros ni filtrado.
- Esto representa una **vulnerabilidad de tipo SQL Injection (SQLi)**.

🛡️ Recomendación básica de mitigación

1. **Usar consultas parametrizadas** (prepared statements) para evitar la inyección de código.
2. **Validar y sanear todas las entradas del usuario** antes de incluirlas en una consulta.
3. Configurar los entornos de producción en modo seguro y registrar intentos de manipulación.

🧠 Reflexión final

Este ejercicio demuestra cómo una simple omisión en la validación de entradas puede comprometer la integridad de una base de datos. La explotación de una vulnerabilidad como esta, en un entorno real, permitiría a un atacante leer, modificar o eliminar información crítica. La solución no es compleja, pero sí fundamental: **separar los datos del usuario de la lógica del sistema**.
