**Informe Técnico**

**Explotación Controlada de Inyección SQL y Validación Manual con Burp Suite – Lección 4**

**1. Escenario**

Durante una auditoría de seguridad en un entorno de laboratorio (**DVWA en localhost**), se sospechaba de una vulnerabilidad de **SQL Injection (SQLi)** en la sección de búsqueda de usuarios. El objetivo fue confirmar manualmente la existencia del fallo mediante el uso de **Burp Suite**, validar su impacto en la aplicación y proponer medidas de mitigación bajo estándares de ciberseguridad.

**2. Metodología**

Se siguieron los siguientes pasos:

1. **Configuración y Reconocimiento:**
   * DVWA corriendo en http://localhost/dvwa con seguridad en nivel *Low*.
   * Burp Suite configurado como proxy en 127.0.0.1:8080.
2. **Interceptación y análisis de la solicitud:**
   * Se ingresó el valor 1 en el parámetro **ID**.
   * Se interceptó la petición en Burp Suite, identificando al parámetro **ID** como susceptible a inyección.
3. **Pruebas de inyección SQL manual:**
   * Payload verdadero:
   * 1' AND 1=1 --

→ Resultado: la aplicación devolvió resultados válidos.

* + Payload falso:
  + 1' AND 1=2 --

→ Resultado: la aplicación no devolvió datos.

* + Payload exploratorio:
  + 1' UNION SELECT 1, database(), 3 --

→ Resultado: se mostró el nombre de la base de datos, confirmando explotación.

1. **Validación del impacto:**
   * Diferencias claras entre payloads válidos y falsos confirmaron la inyección SQL.
   * Se comprobó que un atacante podría obtener información sensible (ej. nombre de la base de datos).

**3. Hallazgos**

* **Parámetro vulnerable:** ID
* **Payloads utilizados:**
  + 1' AND 1=1 -- (verdadero)
  + 1' AND 1=2 -- (falso)
  + 1' UNION SELECT 1, database(), 3 -- (extracción de datos)
* **Riesgo estimado:** **Alto** – acceso no autorizado a información confidencial.

**4. Recomendaciones de Seguridad**

1. Implementar **consultas parametrizadas (prepared statements)** en lugar de concatenar entradas.
2. Aplicar **validación de entrada** estricta en todos los parámetros recibidos.
3. Restringir el nivel de detalle en mensajes de error del servidor.
4. Aplicar un **WAF** (Web Application Firewall) como defensa adicional.
5. Capacitar a desarrolladores en **OWASP Top 10**, especialmente en prevención de inyecciones.

**5. Reflexión**

El comportamiento distinto de la aplicación frente a payloads verdaderos y falsos evidencia lo simple que resulta para un atacante confirmar una inyección SQL. De no mitigarse, esta vulnerabilidad permitiría extraer datos sensibles con gran facilidad. Validar manualmente cada parámetro es crítico antes de darlo por seguro.

**6. Conclusión**

El análisis confirmó que la aplicación es vulnerable a **SQL Injection** en el parámetro **ID**, lo que permite manipular consultas SQL y acceder a información confidencial. Se validó el impacto con pruebas controladas y se documentaron medidas de mitigación. Este laboratorio refuerza la importancia de integrar prácticas seguras de desarrollo y auditoría continua para proteger aplicaciones web contra ataques reales.