




## Ejercicio Práctico

 **Título:** Identificación y Aplicación de Técnicas de Mitigación Comunes

---

### **Objetivo:**

Reconocer vulnerabilidades frecuentes en aplicaciones web y proponer técnicas básicas de mitigación adecuadas para cada una.

---

### **Escenario:**

La aplicación web "**BlogSeguro**" tiene reportes de comportamiento anómalo en algunas de sus funcionalidades. El equipo de desarrollo ha identificado los siguientes problemas:

1. Usuarios logueados reportan que sus comentarios insertan automáticamente **ventanas emergentes** cuando otros usuarios los visualizan.
  2. Algunos formularios permiten ingresar valores como `admin' OR '1'='1` en campos de texto, mostrando información de usuarios no autorizados.
  3. Un botón para borrar la cuenta del usuario se puede activar desde un **link externo**, sin confirmación previa.
- 

### **Actividades:**

1. **Identifica la vulnerabilidad presente en cada uno de los 3 casos descritos.**
  2. **Asocia una técnica de mitigación básica apropiada para cada caso.**
  3. **Explica brevemente por qué la técnica seleccionada es efectiva.**
-

### Formato de respuesta sugerido:

Caso	Tipo de Vulnerabilidad	Técnica de Mitigación	Justificación
1			
2			
3			

---

### Recomendaciones:

- Considera técnicas como: escaping, prepared statements, tokens CSRF, validación de entradas, cabeceras CSP.
  - Si lo deseas, incluye ejemplos de código básico como parte de tu respuesta.
- 

### Entregables esperados:

- Tabla completada.
  - Justificación breve para cada técnica seleccionada.
  - (Opcional) Fragmento de código que ilustre la solución.
-

## ✓ Solución Modelo – Ejercicio Práctico Nivel Básico

📌 **Escenario:** Aplicación "BlogSeguro"

🎯 **Objetivo:** Reconocer vulnerabilidades comunes y aplicar técnicas básicas de mitigación.

---

### 📋 Tabla de respuestas:

Caso	Tipo de Vulnerabilidad	Técnica de Mitigación	Justificación
1	Cross-Site Scripting (XSS)	Escapar caracteres y aplicar CSP	El XSS ocurre porque los comentarios no son codificados al mostrarse. Escapar caracteres como <code>&lt;</code> , <code>&gt;</code> y activar CSP evita la ejecución de scripts.
2	Inyección SQL (SQLi)	Uso de consultas preparadas (prepared statements)	El código SQL se ve alterado por entradas maliciosas. Las consultas parametrizadas evitan que los datos ingresados sean interpretados como comandos.
3	Cross-Site Request Forgery (CSRF)	Uso de token CSRF único por formulario	El enlace externo activa una acción sin control. Un token CSRF permite verificar si la acción proviene de una sesión legítima.

---

### 🧩 Fragmentos de código ilustrativos:

#### ✓ Caso 1 – Mitigación XSS (JavaScript):

```
function escapeHTML(str) {  
    return str.replace(/</g, "&lt;").replace(/>/g, "&gt;");  
}
```

#### ✓ Caso 2 – SQLi (PHP):

```
$stmt = $conn->prepare("SELECT * FROM usuarios WHERE email=?");  
$stmt->bind_param("s", $email);  
$stmt->execute();
```

#### ✓ Caso 3 – CSRF (HTML + backend):

```
<form method="POST">  
    <input type="hidden" name="csrf_token" value="abc123xyz">  
</form>
```

```
# Backend (Django-like pseudo):  
if request.POST["csrf_token"] != session["csrf_token"]:  
    return "403 - Request Rejected"
```

---

## Conclusión:

Este ejercicio refuerza la importancia de aplicar técnicas de mitigación básicas como **escapar salidas**, **parametrizar consultas**, y **proteger formularios críticos** con **tokens antifalsificación**. Estas medidas son fundamentales para cualquier equipo de desarrollo web moderno.

---