**Informe Técnico**

**Detección y Mitigación de Vulnerabilidades Web en Sistema de Comentarios – Lección 5**

**1. Escenario**

Se analizó un fragmento de código en **PHP** correspondiente a un sistema de comentarios. El código original insertaba directamente datos de usuarios en la base de datos sin validación ni protección, lo que expuso al sistema a vulnerabilidades críticas como **Inyección SQL**, **Cross-Site Scripting (XSS)** y **CSRF**. El objetivo del laboratorio fue **identificar, explotar y corregir** estas vulnerabilidades aplicando medidas de seguridad estándar.

**2. Análisis de Vulnerabilidades**

1. **Inyección SQL (SQLi):**
   * El código concatenaba directamente valores de $\_POST en la consulta SQL.
   * Riesgo: un atacante podría inyectar sentencias destructivas (' OR '1'='1 o DROP TABLE).
2. **Cross-Site Scripting (XSS):**
   * El campo comentario permitía insertar código malicioso como:
   * <script>alert('XSS');</script>
   * Riesgo: ejecución arbitraria de JavaScript en navegadores de usuarios, robo de cookies o redirecciones.
3. **Cross-Site Request Forgery (CSRF):**
   * No existía validación del origen de las solicitudes.
   * Riesgo: un atacante podía forzar acciones (como envío de comentarios) desde otra página sin consentimiento del usuario autenticado.

**3. Código Seguro Implementado**

**3.1 Formulario con Token CSRF**

<form method="POST" action="comentarios.php">

<input type="text" name="nombre">

<textarea name="comentario"></textarea>

<input type="hidden" name="csrf\_token" value="<?php echo $\_SESSION['csrf\_token']; ?>">

<input type="submit" value="Enviar">

</form>

**3.2 Código PHP Seguro**

<?php

session\_start();

// Verificar token CSRF

if ($\_POST['csrf\_token'] !== $\_SESSION['csrf\_token']) {

die("CSRF Token inválido");

}

// Escapar salidas para prevenir XSS

$nombre = htmlspecialchars($\_POST['nombre'], ENT\_QUOTES, 'UTF-8');

$comentario = htmlspecialchars($\_POST['comentario'], ENT\_QUOTES, 'UTF-8');

// Uso de Prepared Statements para prevenir SQLi

$stmt = $conexion->prepare("INSERT INTO comentarios (nombre, comentario) VALUES (?, ?)");

$stmt->bind\_param("ss", $nombre, $comentario);

$stmt->execute();

?>

**4. Evaluación con Herramientas**

Se utilizó **OWASP ZAP** para analizar el sistema vulnerable antes y después de las correcciones.

**Hallazgos principales:**

1. **SQLi detectado en parámetro comentario.**
   * Estado inicial: consultas concatenadas, vulnerables.
   * Estado corregido: uso de *prepared statements* → ✅ Mitigado.
2. **XSS reflejado en salida de comentarios.**
   * Estado inicial: ejecución de <script>alert('XSS')</script>.
   * Estado corregido: uso de htmlspecialchars() → ✅ Mitigado.
3. **CSRF en envío de formularios.**
   * Estado inicial: formulario sin validación de origen.
   * Estado corregido: validación con **token CSRF único** → ✅ Mitigado.

**5. Recomendaciones de Ciberseguridad**

1. Integrar estas defensas de forma sistemática en todo el código (no solo en comentarios).
2. Implementar políticas de seguridad de cabeceras HTTP (CSP, X-Frame-Options, HSTS).
3. Monitorear logs de acceso para detectar intentos de inyección o explotación.
4. Aplicar pruebas periódicas con herramientas como **Burp Suite** y **OWASP ZAP**.
5. Capacitar al equipo de desarrollo en **OWASP Top 10** para reducir errores recurrentes.

**6. Reflexión**

Este laboratorio mostró que vulnerabilidades críticas pueden aparecer incluso en sistemas pequeños si no se aplican buenas prácticas. SQLi, XSS y CSRF son vectores comunes que pueden ser prevenidos fácilmente con validación, prepared statements y tokens de seguridad. La experiencia refuerza la importancia de considerar la seguridad como parte integral del desarrollo web.

**7. Conclusión**

La corrección del código original permitió transformar un sistema altamente vulnerable en una aplicación web más robusta, aplicando medidas de **seguridad por diseño**. Las pruebas con OWASP ZAP confirmaron la mitigación de las vulnerabilidades detectadas. Este ejercicio evidencia que el uso de **prácticas seguras y estándares OWASP** es esencial para proteger aplicaciones y usuarios frente a ciberataques reales.