

Práctica de Laboratorio: Inyección de Código (Cross-Site Scripting - XSS)

Objetivo de la práctica:

- Comprender la falla de seguridad de no sanitizar (limpiar) los datos que ingresa un usuario antes de mostrarlos en pantalla.
- Observar cómo un atacante puede ejecutar código JavaScript directamente en el navegador de la víctima.
- Relacionar este ataque con el **Session Hijacking** (Secuestro de sesión), donde el atacante roba las cookies del usuario para suplantar su identidad.

Paso 1: Crear la página web vulnerable

En tu terminal de Ubuntu WSL, en la misma carpeta donde hiciste las prácticas anteriores, crea un nuevo archivo llamado `xss_lab.php`.

Este código tendrá una falla clásica: toma el parámetro de la URL y lo imprime directamente en la página.

Bash

```
nano xss_lab.php
```

Pega el siguiente código fuente:

PHP

```
<?php
// Generar una cookie simulada para la práctica
// Esto simula que el usuario ya inició sesión en el sistema
setcookie("sesion_activa", "token_secreto_del_administrador_777", time() + 3600, "/");
?>
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Laboratorio de XSS</title>
</head>
<body style="font-family: Arial, sans-serif; margin: 40px;">
  <h2>Buscador de Perfiles (Vulnerable a XSS)</h2>

  <form method="GET" action="xss_lab.php">
    <label for="nombre">Busca tu nombre:</label>
    <input type="text" id="nombre" name="nombre" placeholder="ej. Carlos">
```

```

        <button type="submit">Buscar</button>
    </form>
    <hr>

    <?php
    // Lógica vulnerable: Se recibe el texto y se imprime sin ninguna validación
    if (isset($_GET['nombre'])) {
        $nombre = $_GET['nombre'];
        echo "<h3>Resultados para: " . $nombre . "</h3>";
        echo "<p>No se encontraron datos para este usuario.</p>";
    }
    ?>
</body>
</html>

```

Paso 2: Levantar el Servidor Web

Ejecuta el servidor web embebido de PHP en el puerto 8080 (el mismo método utilizado en el laboratorio de Inyección SQL):

Identificamos la ip de nuestro servidor:

```

Bash
    ip a

```

Nos respondería algo como :

```

1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
    group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    net 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet 10.255.255.254/32 brd 10.255.255.254 scope global lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever

2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1492 qdisc mq state UP
    group default qlen 1000
    link/ether 00:15:5d:08:4f:10 brd ff:ff:ff:ff:ff:ff

```

```
inet 172.19.73.227/20 brd 172.19.79.255 scope global eth0
    valid_lft forever preferred_lft forever
inet6 fe80::215:5dff:fe08:4f10/64 scope link
    valid_lft forever preferred_lft forever
```

Paso 3: Ejecutar el ataque XSS

En el navegador escribimos algo similar a:

`http://172.19.73.227/xss_lab.php`

1. **Búsqueda Legítima:** Escribe Carlos y presiona Buscar. La página simplemente mostrará "Resultados para: Carlos".
2. **La Inyección XSS Básica:** Escribe exactamente el siguiente código JavaScript en la barra de búsqueda y presiona enter:

```
<script>alert(';Vulnerabilidad XSS Encontrada!');</script>
```

Pudimos ejecutar nuestro texto en el servidor con esa sencilla linea

3. **Escalar el ataque (Simulación de Robo de Cookies):** Para generar un análisis más profundo sobre las consecuencias, inyectamos el siguiente código, en la barra de búsqueda y presiona enter:

```
<script>alert(document.cookie);</script>
```

Si la aplicación web manejara sesiones reales, esta simple línea mostraría en pantalla el identificador de sesión del usuario.

Al hacerlo en este orden, la alerta ya no saldrá vacía y mostrará el texto

```
sesion_activa=token_secreto_del_administrador_777
```

¿Qué pasa internamente?

El navegador de la víctima recibe el texto, pero al ver las etiquetas `<script>`, no lo interpreta como un nombre, sino como una instrucción de programación y ejecuta la ventana emergente.

En lugar de lanzar una simple alert(), el script inyectado estuviera programado para enviar ese dato (el token secreto) a un servidor controlado por el atacante, este último podría incrustar esa cookie en su propio navegador y entrar a la cuenta de la víctima sin necesidad de saber la contraseña. Esto es lo que se conoce como Session Hijacking (Secuestro de sesión).

Preguntas de Análisis para la Clase

Analizar

1. Según la clasificación general de ataques, ¿este ataque es pasivo o activo?
(Respuesta esperada: Es un **Ataque activo**, ya que el atacante modifica o genera datos maliciosos a través de un vector de aplicación).
2. Si un atacante guarda este código en un comentario de un foro y cualquiera que entra a leerlo es atacado, ¿cómo se relaciona esto con el "Session Hijacking"? (Respuesta esperada: El atacante puede robar las cookies de los visitantes para secuestrar su sesión activa y suplantarlos).

Paso 5: La Contramedida (Cómo arreglarlo)

Finalmente, enséñales cómo solucionar la vulnerabilidad editando el código PHP. Deben envolver la variable con la función de seguridad htmlspecialchars(), la cual convierte los caracteres especiales (como < y >) en texto inofensivo.

Cambiar:

```
echo "<h3>Resultados para: " . $nombre . "</h3>";
```

Por:

```
echo "<h3>Resultados para: " . htmlspecialchars($nombre) . "</h3>";
```

Al intentar inyectar el código malicioso nuevamente, verán el código escrito en pantalla como texto plano, bloqueando el ataque de forma exitosa.

Paso 6: Analizar

Obtener el token de sesión le permite al atacante ejecutar un ataque de red conocido como Session Hijacking (Secuestro de sesión), cuyo objetivo es robar una sesión activa para suplantar al usuario legítimo mediante el robo de cookies. Al apoderarse de este identificador, el atacante obtiene las siguientes ventajas críticas:

- **Evasión de autenticación:** No necesita conocer el nombre de usuario, la contraseña, ni superar un factor de doble autenticación (MFA) para acceder a la cuenta.
- **Suplantación total:** Al inyectar la cookie robada en su propio navegador, el servidor web asume automáticamente que el atacante es el usuario real que ya había iniciado sesión previamente.
- **Herencia de privilegios:** El atacante adquiere exactamente el mismo nivel de acceso que la víctima; si la cookie robada pertenece a un administrador, el atacante tendrá control administrativo total sobre la aplicación.
- **Ejecución de acciones maliciosas:** Puede robar bases de datos, realizar transferencias, modificar configuraciones o cambiar la contraseña para bloquear al dueño legítimo de la cuenta, dejando un rastro de auditoría que culpará a la víctima.