PRATICA S6L2:

Esercizio del Giorno

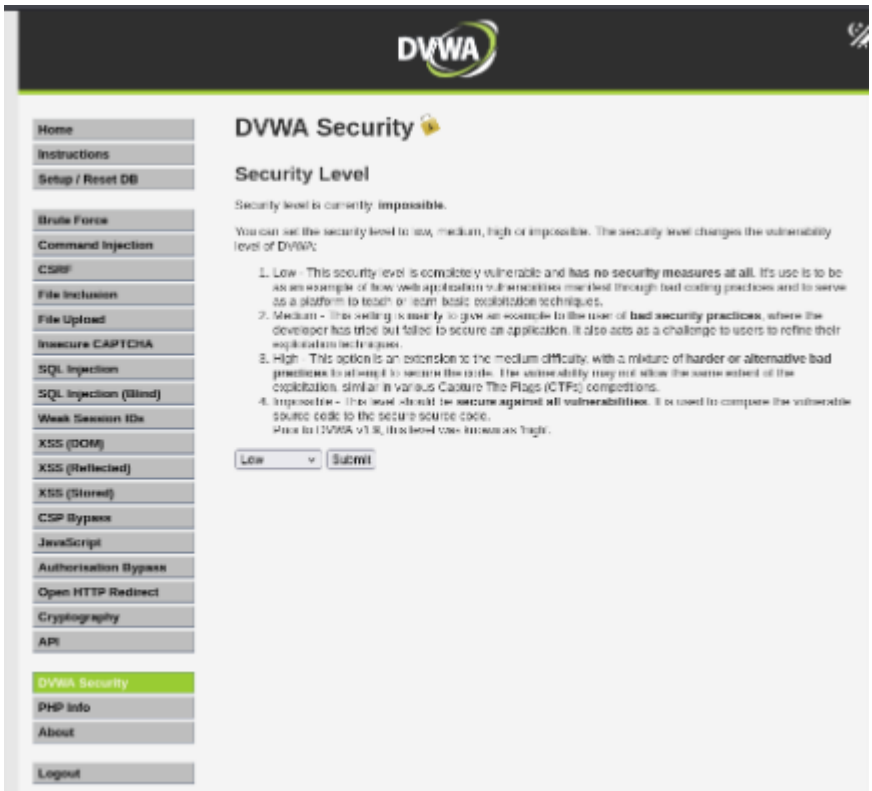
Argomento: Sfruttamento delle Vulnerabilità XSS e SQL Injection sulla DVWA

Obiettivi: Configurare il laboratorio virtuale per sfruttare con successo le vulnerabilità XSS e SQL Injection sulla Damn Vulnerable Web Application (DVWA).

Impostare il livello di sicurezza

Nel pannello DVWA:

- Vai su **DVWA Security**.
- Security Level: Low
- **Significato:** Nessuna protezione. Il codice è volutamente vulnerabile e facile da "bucare"



XSS reflected

XSS Reflected (Cross-Site Scripting Riflesso) è una vulnerabilità in cui un input malevolo (come uno script JavaScript) viene inviato tramite una richiesta HTTP (es. URL o form) e immediatamente "riflesso" nella risposta della pagina web, senza essere filtrato. Questo permette all'attaccante di eseguire codice nel browser della vittima, ad esempio mostrando un popup, rubando cookie o reindirizzando a siti dannosi.

È "riflesso" perché il payload non viene memorizzato, ma appare solo nella risposta immediata.

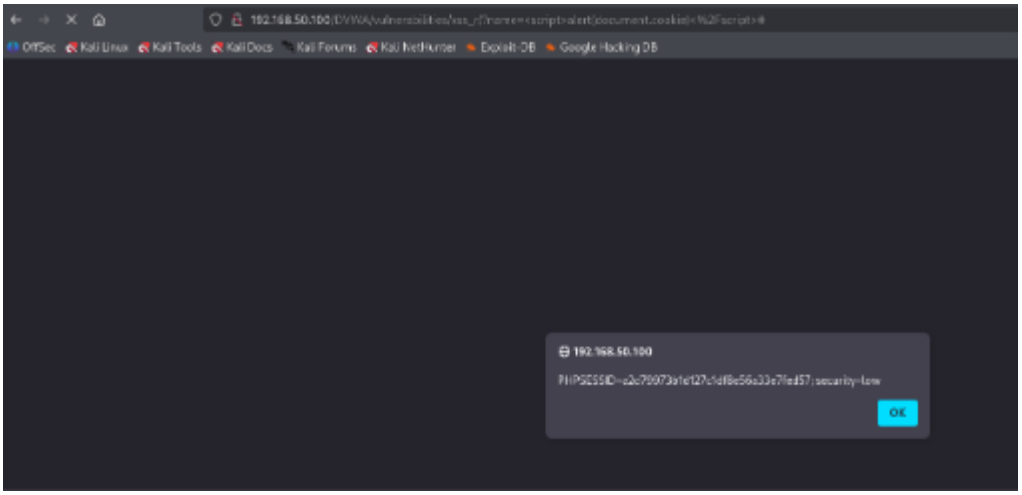
Esempio:

<script>alert(document.cookie)</script>

Ecco cosa significa, passo per passo:

- <script> è un tag HTML che serve a inserire codice JavaScript in una pagina web.
- alert(document.cookie) è un comando che mostra un popup con tutti i cookie associati alla pagina. I cookie possono contenere informazioni sensibili come token di sessione, dati di autenticazione o preferenze utente.

In breve: questo è un esempio di come un sito web possa essere **compromesso** se non filtra correttamente i contenuti inseriti dagli utenti.



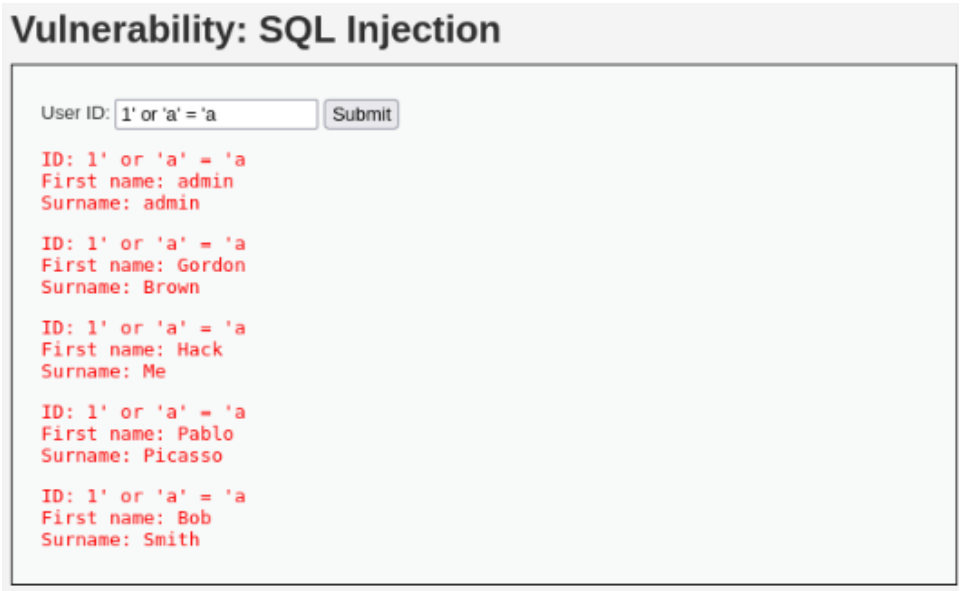
SQL Injection (non-blind)

Una **SQL Injection non-blind** è una tecnica in cui un attaccante inserisce codice SQL in un campo vulnerabile (come un form di login o una barra di ricerca) e **riceve una risposta visibile** dal database.

Questo permette all'attaccante di capire subito se l'attacco ha avuto successo.

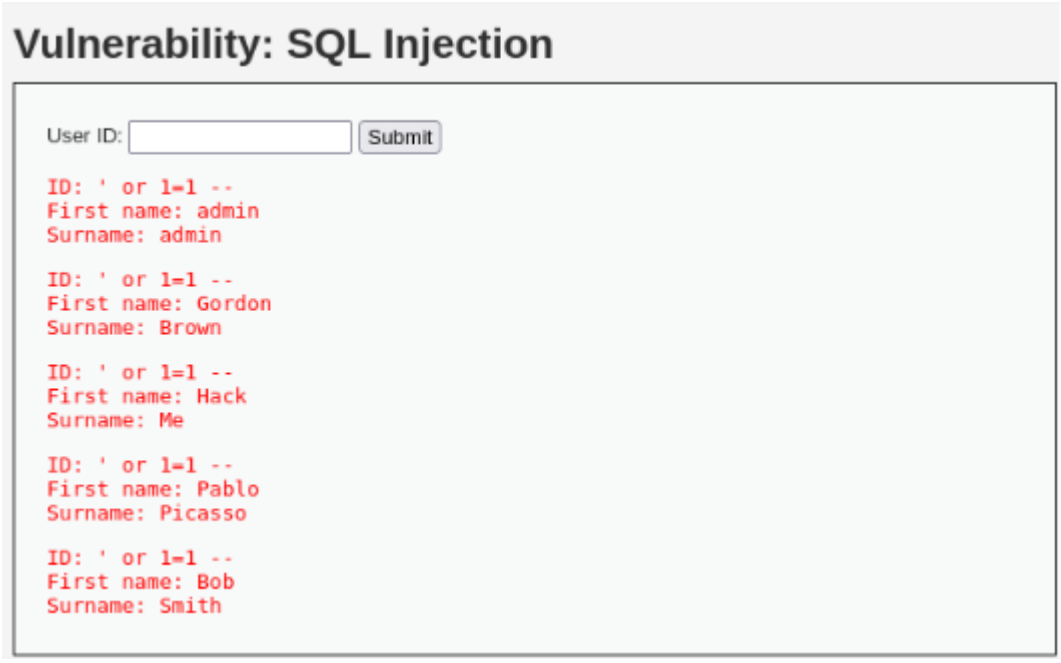
Esempio:

1. Inserendo **`1' or 'a' = 'a`**, si forza la condizione della query a essere sempre vera, permettendo l'accesso non autorizzato o la visualizzazione di dati che non dovrebbero essere accessibili. In parole povere: è come ingannare il sistema per ottenere una risposta che normalmente non verrebbe concessa.



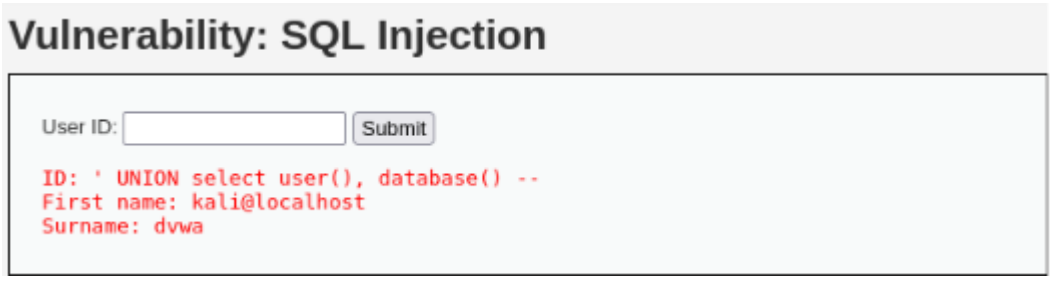
2. **`' or 1=1 --`** altera la logica della query in modo che la condizione risulti sempre vera (1=1 è sempre vero), mentre il `--` commenta il resto della query, impedendo l'esecuzione della parte che potrebbe bloccare l'accesso. Il risultato? Il sistema potrebbe concedere accessi o

mostrare dati che normalmente sarebbero protetti.

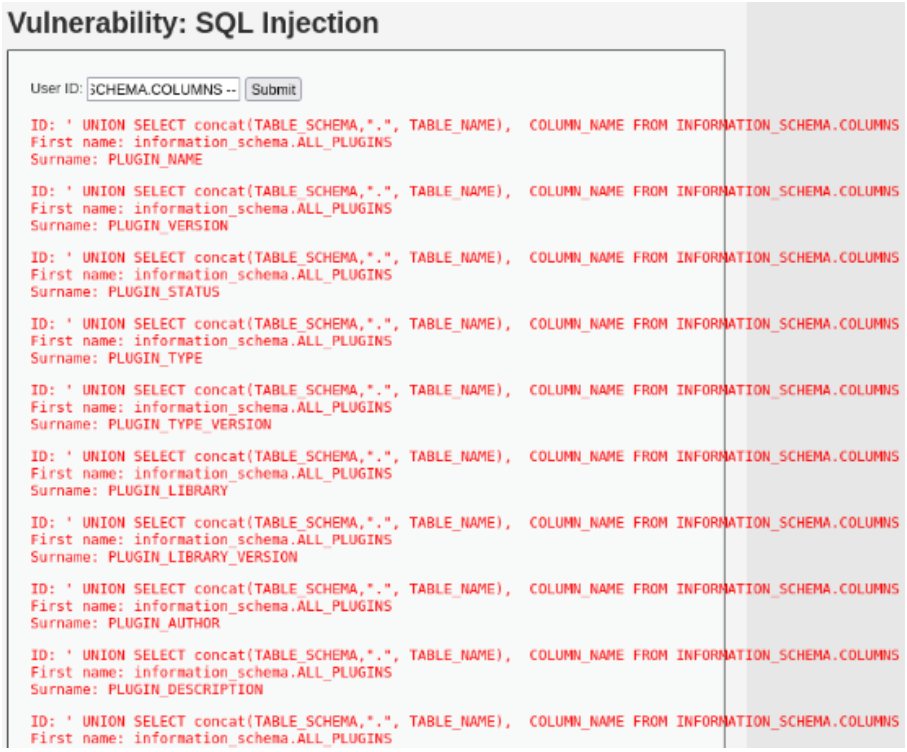


3. **UNION select user(), database() --**

   Quella stringa che hai scritto è un esempio avanzato di SQL injection, in cui si sfrutta l'operatore `UNION` per unire il risultato di una query legittima con un'altra costruita dall'attaccante. In questo caso, si cerca di ottenere informazioni come l'utente corrente del database (`user()`) e il nome del database in uso (`database()`).



4. **' UNION SELECT concat(TABLE_SCHEMA,".", TABLE_NAME), COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS --** cerca di esplorare la struttura interna del database. Con questa query, un attaccante sta provando a elencare tutti i nomi delle colonne (`COLUMN_NAME`) di tutte le tabelle (`TABLE_NAME`) presenti negli schemi (`TABLE_SCHEMA`) del database. In altre parole, è come cercare di mappare l'intero contenuto del database per trovare informazioni sensibili o punti deboli.



5. **' UNION SELECT concat(TABLE_SCHEMA,".", TABLE_NAME), COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS WHERE table_schema="dvwa" --**Questa istruzione SQL injection cerca di estrarre informazioni dettagliate dal database, concentrandosi specificamente sullo schema `"dvwa"`—che è spesso utilizzato come ambiente di test per dimostrazioni di vulnerabilità web.

   In particolare, la query sfrutta `UNION SELECT` per unire il risultato a una richiesta che concatena il nome dello schema e della tabella (`TABLE_SCHEMA.TABLE_NAME`) con il nome delle colonne (`COLUMN_NAME`). Il filtro `WHERE table_schema="dvwa"` limita l'output ai dati dell'applicazione target.

SQLMAP:

sqlmap è uno strumento open source usato nel penetration testing per automatizzare il rilevamento e lo sfruttamento delle vulnerabilità da SQL injection. È come un coltellino svizzero per chi analizza la sicurezza delle applicazioni web: potente, versatile e capace di interagire con una vasta gamma di database.

Con sqlmap puoi:

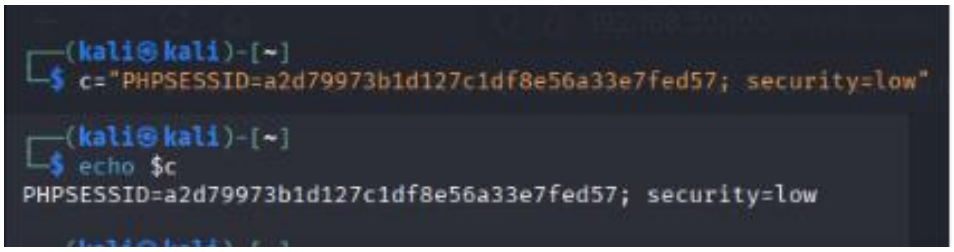
- Identificare e sfruttare diversi tipi di SQL injection (blind, error-based, time-based, ecc.).
- Estrarre dati da tabelle e colonne, anche in modo mirato.
- Eseguire comandi sul server se le condizioni lo permettono.
- Integrare con altri strumenti come Metasploit per escalation di privilegi.

Esempio:

1. Ho creato la variabile c che è una semplice definizione in Bash, dove assegni una stringa di cookie a una variabile. In questo caso:

   C= “PHPSESSID=a2d79973b1d127c1df8e56a33e7fed57; security=low”

   Serve a semplificare la scrittura dei comandi successivi. Invece di riscrivere i cookie ogni volta, usi $c per richiamare il contenuto.



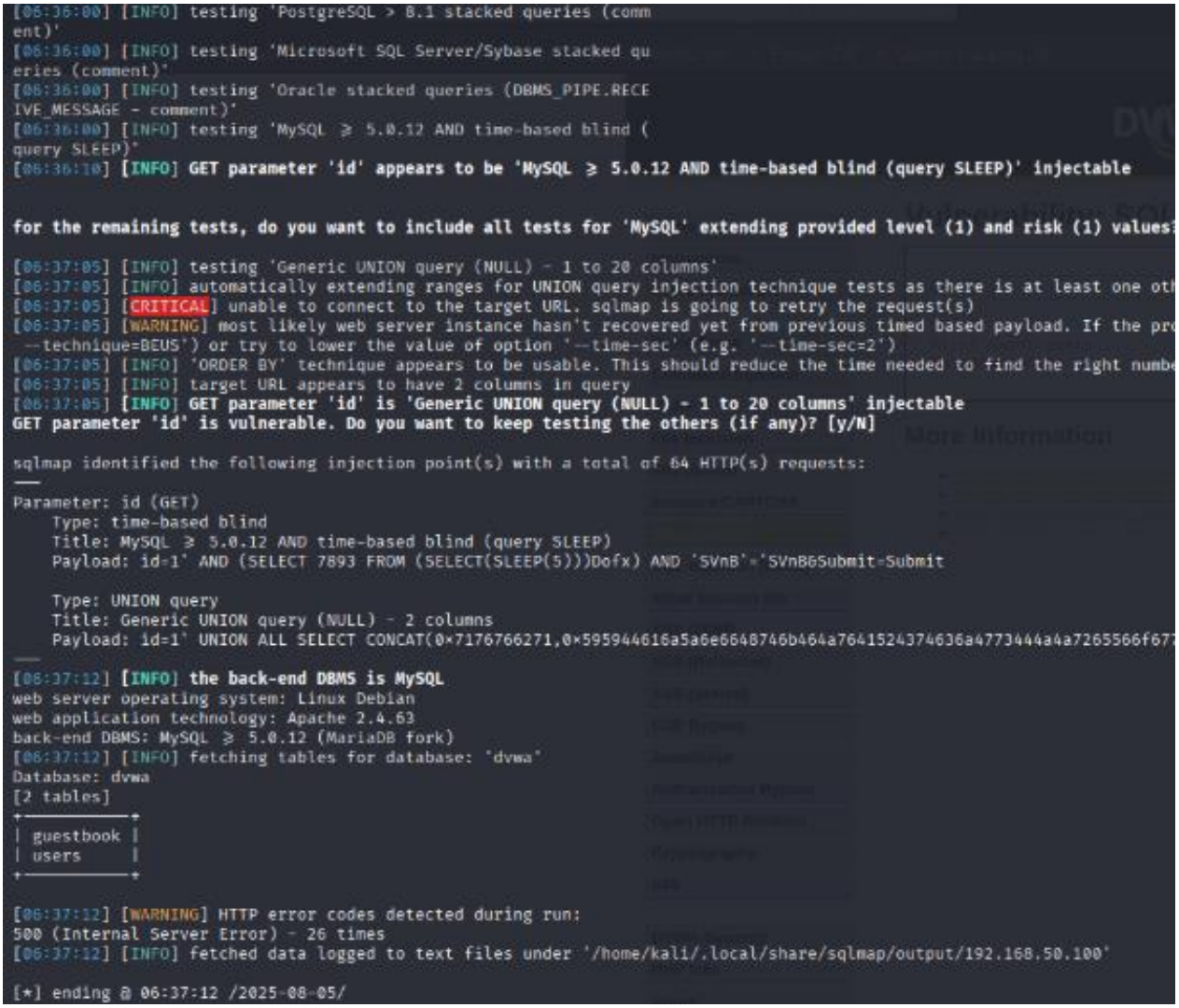
2. **Identificazione dei Database** Ho avviato una scansione iniziale con:

   sqlmap -u "http://192.168.50.101/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit" --cookie=$c --dbs

Grazie a `sqlmap`, il comando ha verificato se l'app è vulnerabile e ha richiesto l'elenco dei database esistenti.

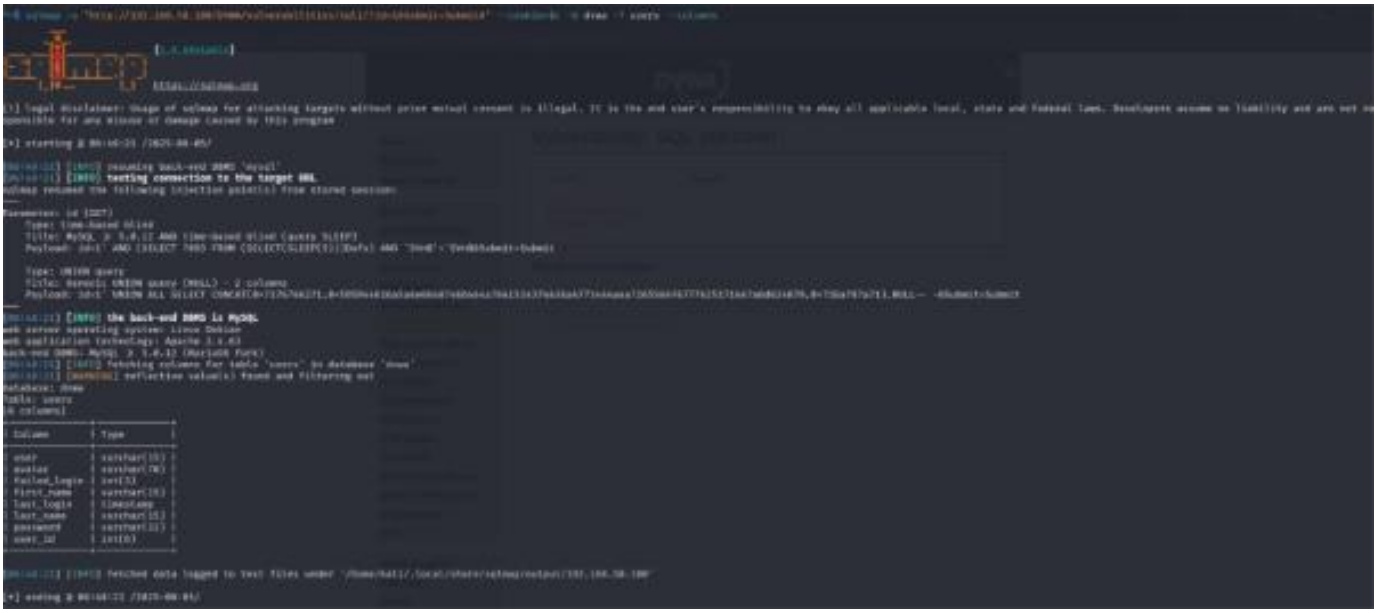
3. **Enumerazione Tabelle** Successivamente ho esplorato la struttura del database `dvwa`:

   sqlmap -u "http://192.168.50.100/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit" --cookie=$c -D dvwa –tables



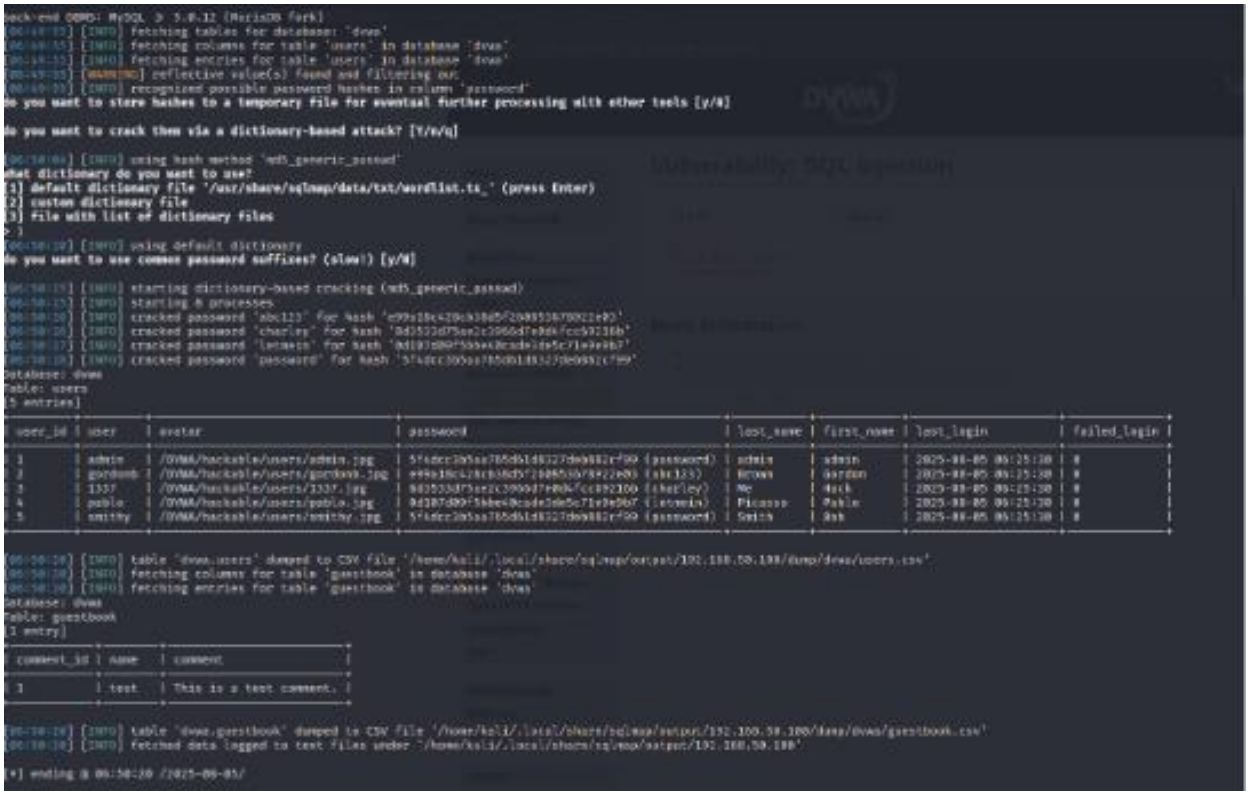
4. **Analisi della Tabella** `users` quella che, per convenzione, custodisce le informazioni degli account. Con questo passo, ho chiesto quali **colonne** contiene.

   sqlmap -u "http://192.168.50.100/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit" --cookie=$c -D dvwa -T users –columns



5. **Estrazione dei Dati, Infine**, Con il comando `--dump-all`, ho ordinato a sqlmap di prelevare tutto il contenuto del database dvwa. Essendo in ambiente di test e sicurezza impostata su "low", sqlmap ha avuto via libera. Mi ha restituito i dati della tabella `users`, incluse le credenziali, in forma di **hash criptati**.

   sqlmap -u "http://192.168.50.100/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit" --cookie=$c -D dvwa --dump-all



Conclusione dell'Esercizio: Sfruttamento delle Vulnerabilità XSS e SQL Injection su DVWA

Durante l'esercizio, è stato configurato correttamente un ambiente di laboratorio composto da una macchina attaccante Kali Linux e il target DVWA. Dopo aver verificato la comunicazione tra le due entità, si è proceduto con l'accesso all'applicazione vulnerabile e la configurazione del livello di sicurezza a **LOW**, condizione necessaria per simulare exploit realistici in ambiente controllato.

Successivamente, sono state identificate e sfruttate due tipologie di vulnerabilità:

- **XSS reflected**: È stata individuata una pagina vulnerabile che riflette input utente direttamente nell'output HTML, consentendo l'esecuzione di script malevoli nel browser della vittima. Questo tipo di attacco evidenzia i rischi legati all'assenza di validazione e sanitizzazione dell'input.
- **SQL Injection non blind**: È stata eseguita una serie di interrogazioni SQL malformate tramite tool come sqlmap, permettendo l'accesso al backend della DVWA, l'enumerazione delle tabelle e l'estrazione di dati sensibili dalla tabella users.

L’esercizio ha permesso di comprendere in modo pratico come queste vulnerabilità possano essere sfruttate e quanto sia fondamentale applicare **misure di sicurezza adeguate** come la sanitizzazione dell’input, la parametrizzazione delle query SQL e il monitoraggio delle risposte dell’applicazione.