

## REPORT S3/L2

### Exploit DVWA - XSS e SQL injection

#### Traccia

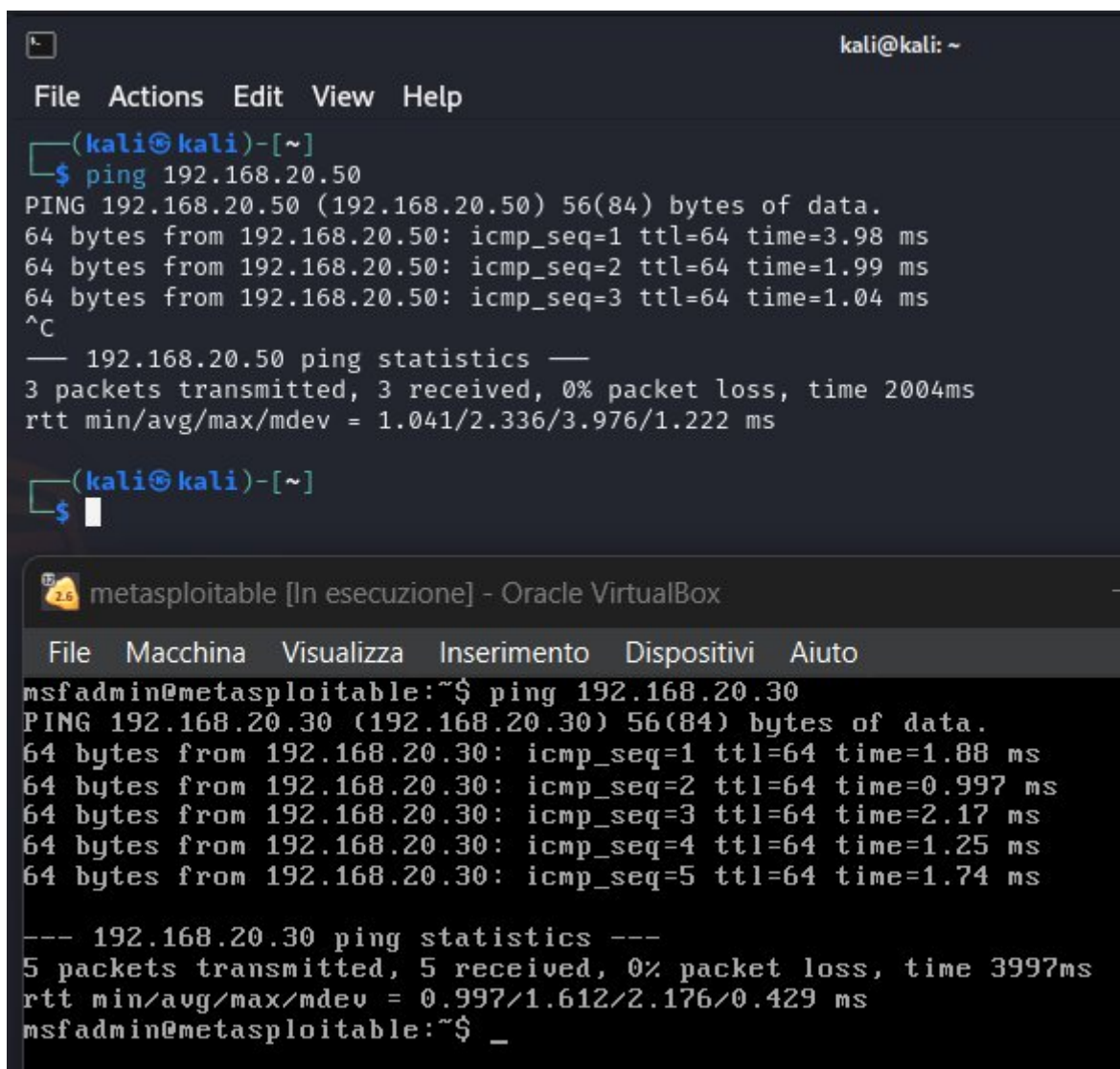
Sfruttamento delle Vulnerabilità XSS e SQL Injection sulla DVWA.

#### Svolgimento

Per procedere con l'esecuzione dell'esercizio configuriamo le macchine di Kali e Meta in modo che possano comunicare tra di loro. Assegniamo alle macchine i seguenti indirizzi IP:

- Kali: 192.168.20.30
- Meta: 192.168.20.50

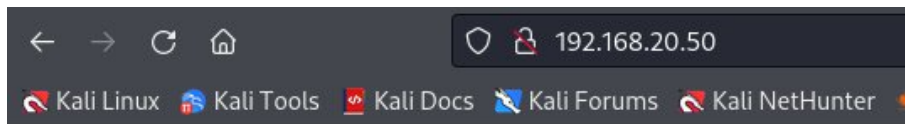
Eseguendo un semplice ping da una macchina all'altra e viceversa verificiamo che la comunicazione tra le due sia attiva.



```
kali@kali: ~  
File Actions Edit View Help  
(kali@kali)-[~]  
$ ping 192.168.20.50  
PING 192.168.20.50 (192.168.20.50) 56(84) bytes of data.  
64 bytes from 192.168.20.50: icmp_seq=1 ttl=64 time=3.98 ms  
64 bytes from 192.168.20.50: icmp_seq=2 ttl=64 time=1.99 ms  
64 bytes from 192.168.20.50: icmp_seq=3 ttl=64 time=1.04 ms  
^C  
--- 192.168.20.50 ping statistics ---  
3 packets transmitted, 3 received, 0% packet loss, time 2004ms  
rtt min/avg/max/mdev = 1.041/2.336/3.976/1.222 ms  
  
(kali@kali)-[~]  
$
```

```
metasploitable [In esecuzione] - Oracle VirtualBox  
File Macchina Visualizza Inserimento Dispositivi Aiuto  
msfadmin@metasploitable:~$ ping 192.168.20.30  
PING 192.168.20.30 (192.168.20.30) 56(84) bytes of data.  
64 bytes from 192.168.20.30: icmp_seq=1 ttl=64 time=1.88 ms  
64 bytes from 192.168.20.30: icmp_seq=2 ttl=64 time=0.997 ms  
64 bytes from 192.168.20.30: icmp_seq=3 ttl=64 time=2.17 ms  
64 bytes from 192.168.20.30: icmp_seq=4 ttl=64 time=1.25 ms  
64 bytes from 192.168.20.30: icmp_seq=5 ttl=64 time=1.74 ms  
  
--- 192.168.20.30 ping statistics ---  
5 packets transmitted, 5 received, 0% packet loss, time 3997ms  
rtt min/avg/max/mdev = 0.997/1.612/2.176/0.429 ms  
msfadmin@metasploitable:~$ _
```

Ci colleghiamo alla macchina di Meta tramite il browser di Kali, navighiamo verso la DVWA ricordandoci di mettere la sicurezza a livello LOW.

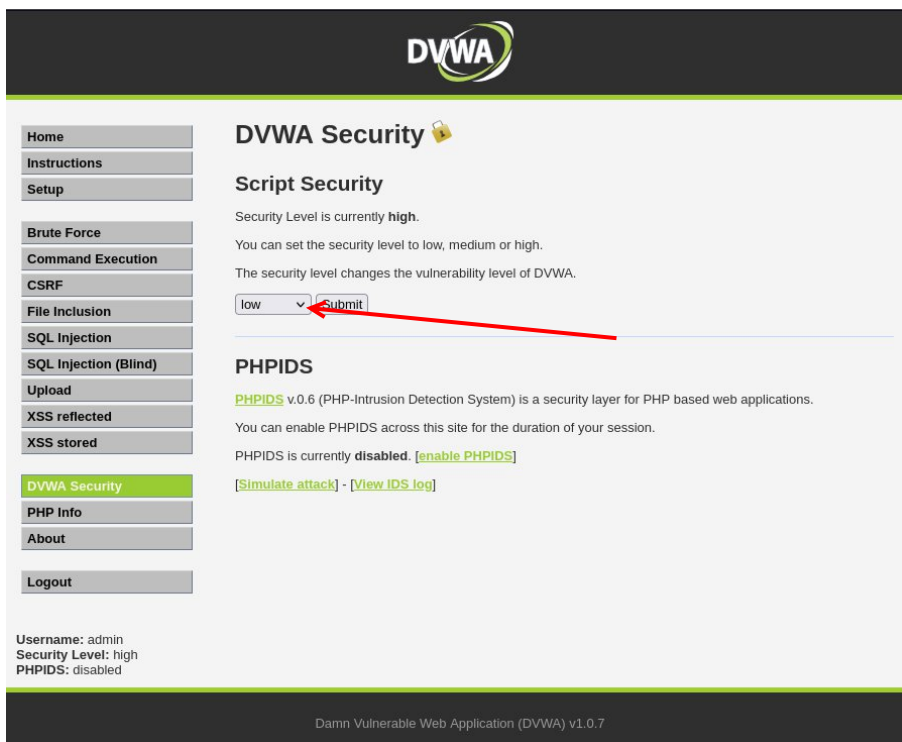


Warning: Never expose this VM to an untrusted network!

Contact: msfdev[at]metasploit.com

Login with msfadmin/msfadmin to get started

- [TWiki](#)
- [phpMyAdmin](#)
- [Mutillidae](#)
- [DVWA](#)
- [WebDAV](#)



## Vulnerabilità XSS reflected

Negli attacchi cross-site scripting (XSS) un codice dannoso viene inserito in siti Web altrimenti attendibili. Di conseguenza, gli script dannosi possono accedere a tutti i cookie, i token di sessioni o altre informazioni sensibili conservate dal browser e utilizzate in quel sito.

Ora dalla scheda *XSS reflected* facciamo alcune prove per vedere se il campo di inserimento delle stringhe è vulnerabile a questo tipo di attacco. Proviamo ad inserire:

1. `<i>Alberto</i>`
2. `<script>alert('XSS Attack!');</script>`

E vediamo come il campo riflette esattamente quello che ci aspettiamo, ovvero la stringa Alberto scritta in corsivo e un pop-up con la scritta XSS Attack. Questo ci conferma la vulnerabilità del campo.

### Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?

Submit

Hello *Alberto*

🌐 192.168.20.50

XSS Attack!

OK

Sfruttiamo l'attacco XSS reflected per recuperare i cookie. Ci basta modificare la riga di codice del secondo esempio in questa: `<script>alert(document.cookie);</script>` per ottenere:

**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  
Logout

## Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?

Hello

192.168.20.50

security=low; PHPSESSID=0f74d66c7a23e6689e22f263c0d082f5

OK

## Vulnerabilità SQL injection

L'SQL injection è una tecnica di hacking che, sfruttando alcuni errori nella programmazione di pagine HTML, consente di inserire ed eseguire codice non previsto all'interno di applicazioni web che interrogano un database.

Per testare questo attacco sulla DVWA di Meta andiamo nell'apposita scheda e proviamo prima di tutto a capire come funziona il campo di inserimento e cosa ci restituisce.

### Vulnerability: SQL Injection

User ID:

ID: 1  
First name: admin  
Surname: admin

Se inseriamo il numero 1 vediamo che ci vengono restituite le informazioni del primo elemento della tabella del database.

Così come troviamo una risposta, anche se di errore, nel momento in cui inseriamo il carattere speciale '.

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

Questo ci fa concludere che il sito / webApp è vulnerabile all'attacco in questione, pertanto prepariamo la stringa da inserire nel campo per farci restituire l'intera tabella.

Scrivendo: 1' OR '1'='1 nel campo User ID otteniamo tutta la tabella con nomi e cognomi:

### Vulnerability: SQL Injection

User ID:

ID: 1' OR '1'='1  
First name: admin  
Surname: admin

ID: 1' OR '1'='1  
First name: Gordon  
Surname: Brown

ID: 1' OR '1'='1  
First name: Hack  
Surname: Me

ID: 1' OR '1'='1  
First name: Pablo  
Surname: Picasso

ID: 1' OR '1'='1  
First name: Bob  
Surname: Smith

Questo perchè avendo passato quell'input è come se avessimo composto la seguente query:

```
SELECT first_name, surname FROM Tabella WHERE id='1' OR '1'='1'
```

Ovviamente l'id 1 esiste e  $1=1$  è una condizione sempre vera, pertanto il risultato della query sarà sempre verificato su ogni riga della tabella.