

Esercizio 27/02/24

Traccia:

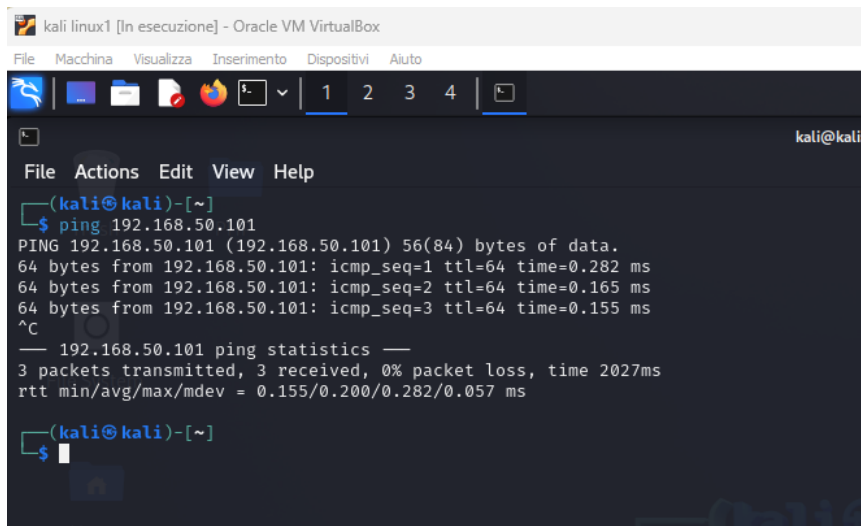
Configurate il vostro laboratorio virtuale per raggiungere la DVWA dalla macchina Kali Linux (l'attaccante). Assicuratevi che ci sia comunicazione tra le due macchine con il comando ping.

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.**

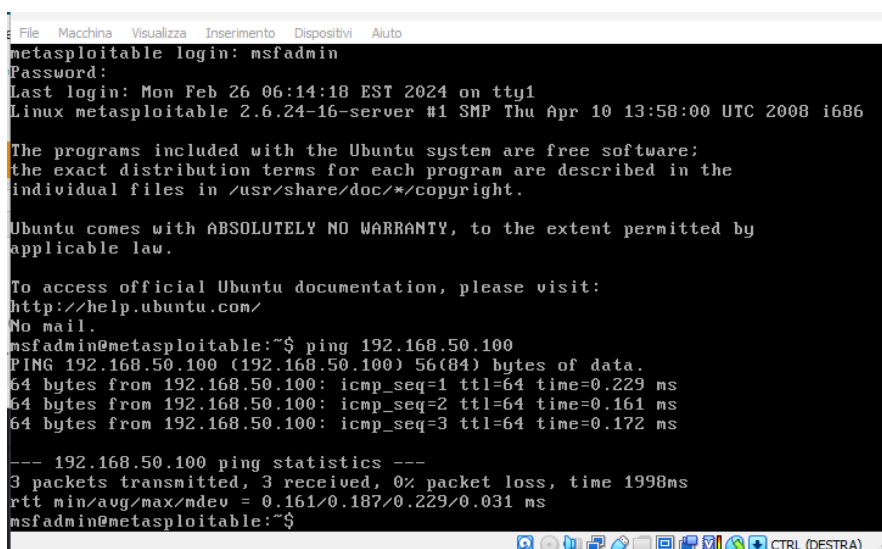
La soluzione riporta l'approccio utilizzato per le seguenti vulnerabilità:

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

Vediamo grazie al ping che le VM Kali e Metasploitable comunicano fra loro.



```
kali linux1 [In esecuzione] - Oracle VM VirtualBox
File Macchina Visualizza Inserimento Dispositivi Aiuto
kali@kali:~$ ping 192.168.50.101
PING 192.168.50.101 (192.168.50.101) 56(84) bytes of data.
64 bytes from 192.168.50.101: icmp_seq=1 ttl=64 time=0.282 ms
64 bytes from 192.168.50.101: icmp_seq=2 ttl=64 time=0.165 ms
64 bytes from 192.168.50.101: icmp_seq=3 ttl=64 time=0.155 ms
^C
--- 192.168.50.101 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2027ms
rtt min/avg/max/mdev = 0.155/0.200/0.282/0.057 ms
```



```
metasploitable login: msfadmin
Password:
Last login: Mon Feb 26 06:14:18 EST 2024 on tty1
Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To access official Ubuntu documentation, please visit:
http://help.ubuntu.com/
No mail.
msfadmin@metasploitable:~$ ping 192.168.50.100
PING 192.168.50.100 (192.168.50.100) 56(84) bytes of data.
64 bytes from 192.168.50.100: icmp_seq=1 ttl=64 time=0.229 ms
64 bytes from 192.168.50.100: icmp_seq=2 ttl=64 time=0.161 ms
64 bytes from 192.168.50.100: icmp_seq=3 ttl=64 time=0.172 ms
--- 192.168.50.100 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.161/0.187/0.229/0.031 ms
msfadmin@metasploitable:~$
```

Settiamo il livello di sicurezza della DVWA a “LOW”.

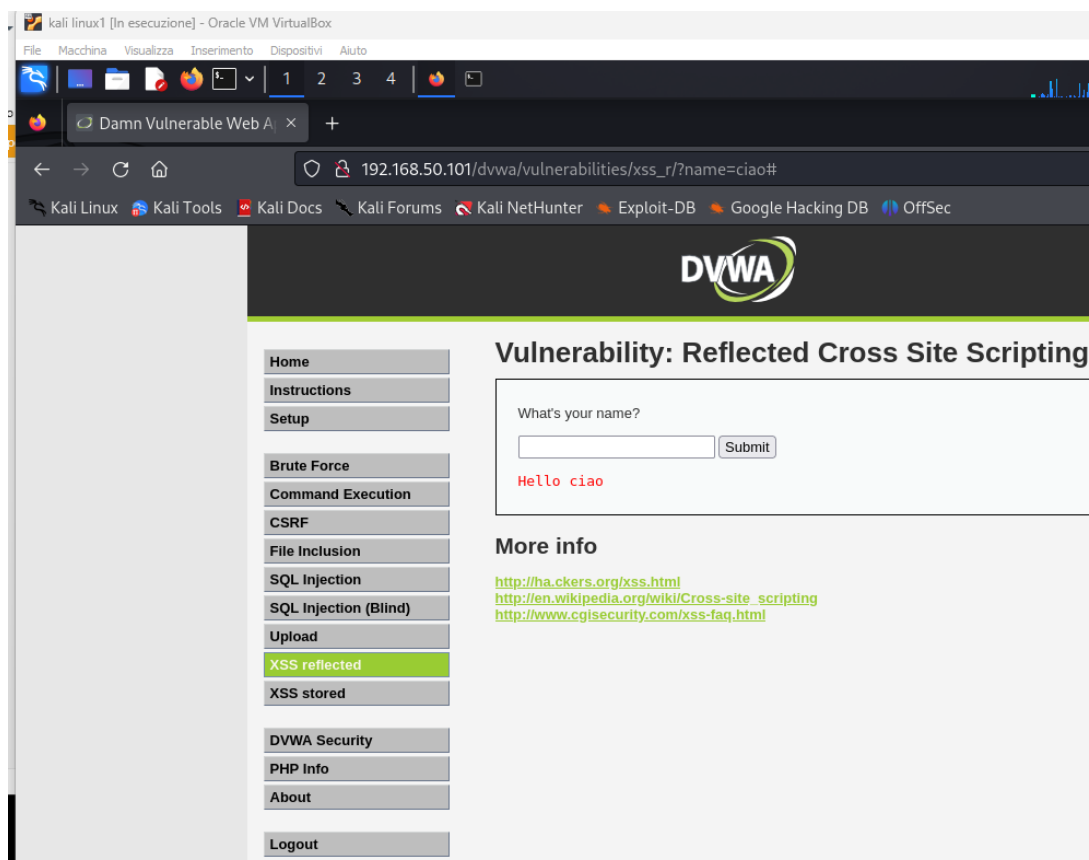


Iniziamo dalla vulnerabilità **XSS REFLECTED**.

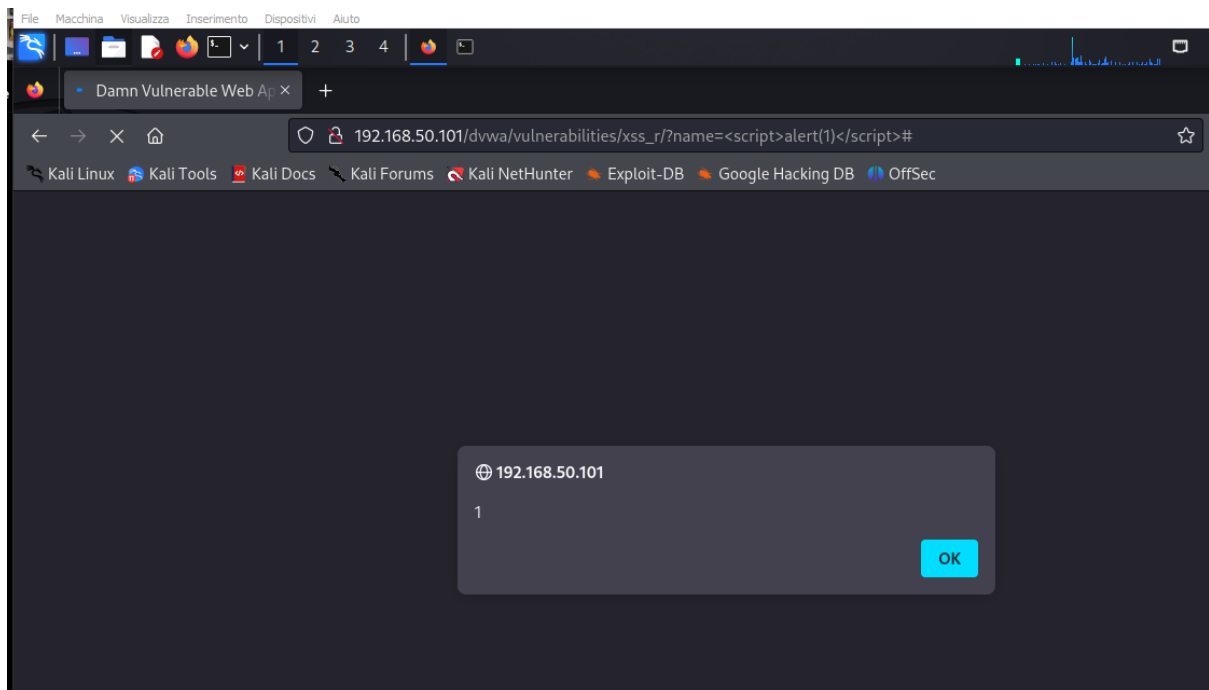
L'obiettivo del test è iniettare del codice javascript all'interno di una pagina web che stampa a video caratteri passati tramite un form.

La richiesta avviene tramite metodo **GET** e la variabile target si chiama “**name**”.

Analizzando il codice sorgente html possiamo notare che il valore inviato (*ciao*) viene stampato a video tra i tag.



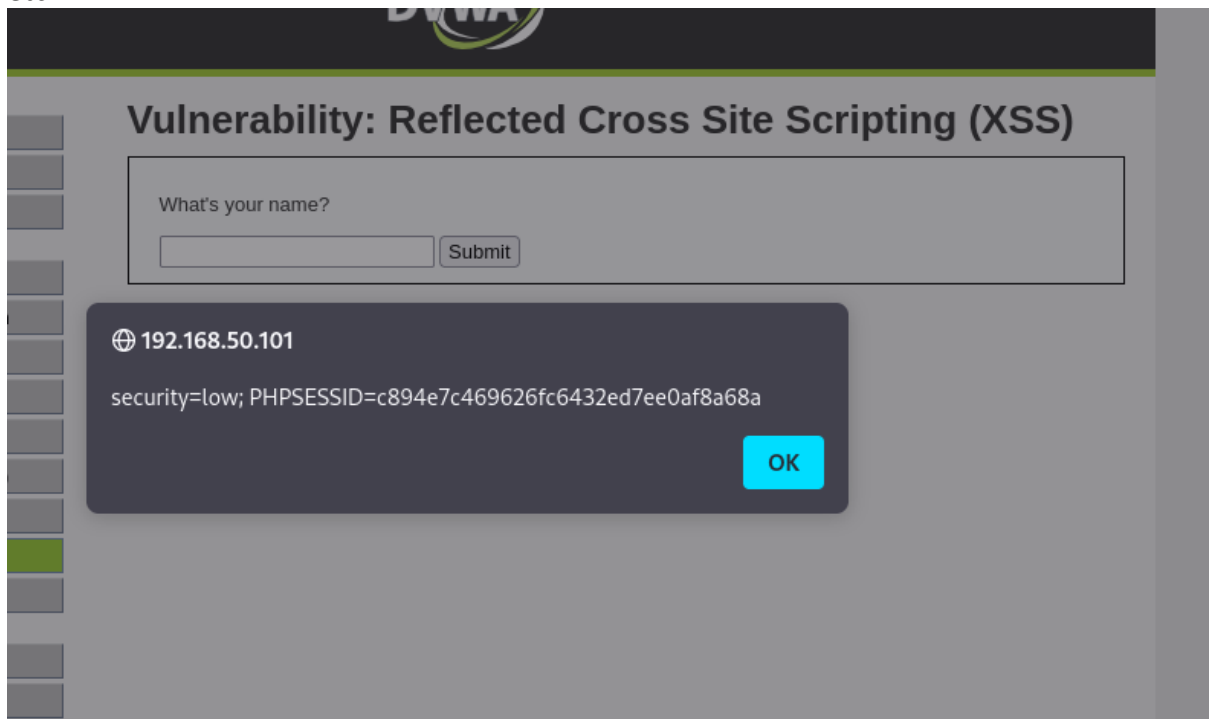
Provando ad inserire il classico payload “**alert(1)**” vediamo che ha funzionato subito:



Andando ad analizzare il codice sorgente possiamo notare che in realtà non è stato inserito alcun controllo in input sulla variabile `$_GET['__name']`:



Sfruttando questa vulnerabilità potremmo anche riuscire a ricavare i cookie, l'id di sessione etc.



Passiamo alla seconda vulnerabilità **SQL Injection**.

La vulnerabilità deriva dalla concatenazione diretta dell'input dell'utente nella query SQL senza una corretta sanificazione o parametrizzazione.

Andando a vedere il codice sorgente, la variabile \$id viene recuperata dall'input dell'utente senza alcuna convalida o sanificazione. Viene quindi direttamente concatenato nella stringa di query SQL:

```
<?php
if(isset($_GET['Submit'])){
    // Retrieve data

    $id = $_GET['id'];

    $getid = "SELECT first_name, last_name FROM users WHERE user_id = '$id'";
    $result = mysql_query($getid) or die('<pre>' . mysql_error() . '</pre>');

    $num = mysql_numrows($result);

    $i = 0;

    while ($i < $num) {

        $first = mysql_result($result,$i,"first_name");
        $last = mysql_result($result,$i,"last_name");

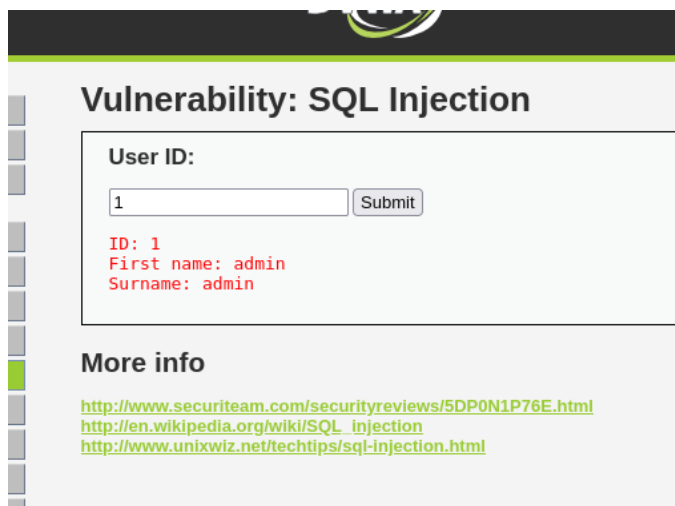
        echo '<pre>';
        echo 'ID: ' . $id . '<br>First name: ' . $first . '<br>Surname: ' . $last;
        echo '</pre>';

        $i++;
    }
}
?>
```

Questo consente a un utente malintenzionato di manipolare il valore di \$id ed immettere codice SQL dannoso, ciò potenzialmente può portare ad accessi non autorizzati, perdita di dati parziale o completa dei dati.

La SQL Injection è una tecnica utilizzata per attaccare le applicazioni basate sui database. Questo viene fatto includendo porzioni di istruzioni SQL in un campo di immissione nel tentativo di indurre il sito Web a passare un comando SQL malevolo appena formato al database (ad esempio, scaricare il contenuto del database all'attaccante). SQL injection è una tecnica di code injection che sfrutta una vulnerabilità di sicurezza nel software di un'applicazione. La vulnerabilità si verifica quando l'input dell'utente viene filtrato in modo errato per caratteri di escape letterali di stringa incorporati in istruzioni SQL o l'input dell'utente non è fortemente tipizzato ed eseguito in modo imprevisto. L'iniezione SQL è principalmente nota come vettore di attacco per siti Web, ma può essere utilizzata per attaccare qualsiasi tipo di database SQL.

Iniziamo con il primo input (1): sintatticamente corretto e semanticamente corretto; dunque ci si aspetta una risposta “corretta”.



Vulnerability: SQL Injection

User ID:

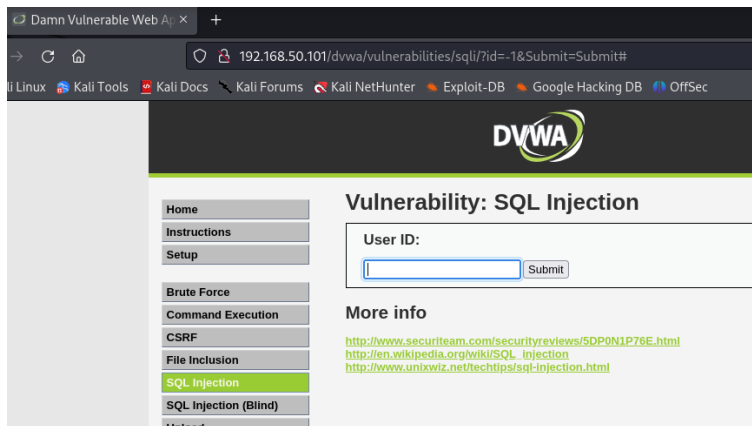
ID: 1
First name: admin
Surname: admin

More info

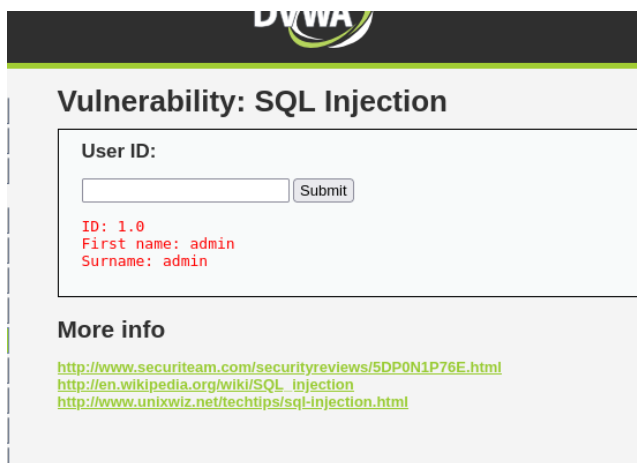
<http://www.securiteam.com/securityreviews/5DP0N1P76E.html>
http://en.wikipedia.org/wiki/SQL_injection
<http://www.unixwiz.net/techtips/sql-injection.html>

Immettiamo invece un input (-1); questo è: sintatticamente corretto, ma semanticamente incorretto (ID negativo).

Avremo una risposta nulla:



Immettiamo l'input (1.0): sintatticamente corretto e semanticamente corretto.
Un'espressione probabilmente equivalente a 1.



L'applicazione stampa direttamente l'input numerico (se valido) e converte un argomento double in intero. La riflessione dell'input (input reflection) è l'atto del server di includere (senza filtri) l'input di un utente nella risposta. Questo è ovviamente un comportamento negativo, perché permette ad un attaccante di vedere il risultato di un attacco, permette l'esecuzione di codice nel contesto del browser della vittima che naviga una pagina maliziosa.

Proviamo con un input (2-1): sintatticamente corretto e semanticamente corretto; si tratta di un'espressione equivalente a 1.

Vulnerability: SQL Injection

User ID:

ID: 2-1
First name: Gordon
Surname: Brown

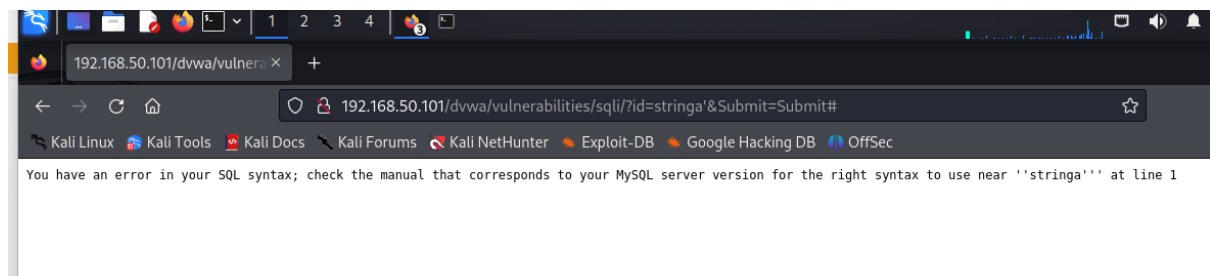
More info

<http://www.securiteam.com/securityreviews/5DP0N1P76E.html>
http://en.wikipedia.org/wiki/SQL_injection
<http://www.unixwiz.net/techtips/sql-injection.html>

L'applicazione riflette l'input numerico (se valido), estrae la parte numerica "2" e la converte in un intero.

Se, invece, immettiamo l'input (stringa'): questo è sintatticamente incorretto e semanticamente incorretto (ID negativo).

Otteniamo così un messaggio di errore del server SQL.



Il sito comunque ci dà delle informazioni: il server è MySQL, il parametro è inserito tra apici singoli, e l'errore avviene alla riga 1.

Possiamo provare ad iniettare un input che trasformi la query SQL in un'altra in grado di stampare tutte le righe della tabella. Il server SQL stampa tutte le righe della tabella se e solo se la clausola WHERE risultante dall'iniezione è sempre vera. Una **tautologia** è una condizione logica vera indipendentemente dall'input utente. L'esempio più classico di tautologia è il seguente:

```
SELECT f1, f2, f3
FROM table
WHERE f1 = 'v1' OR '1'='1';
```

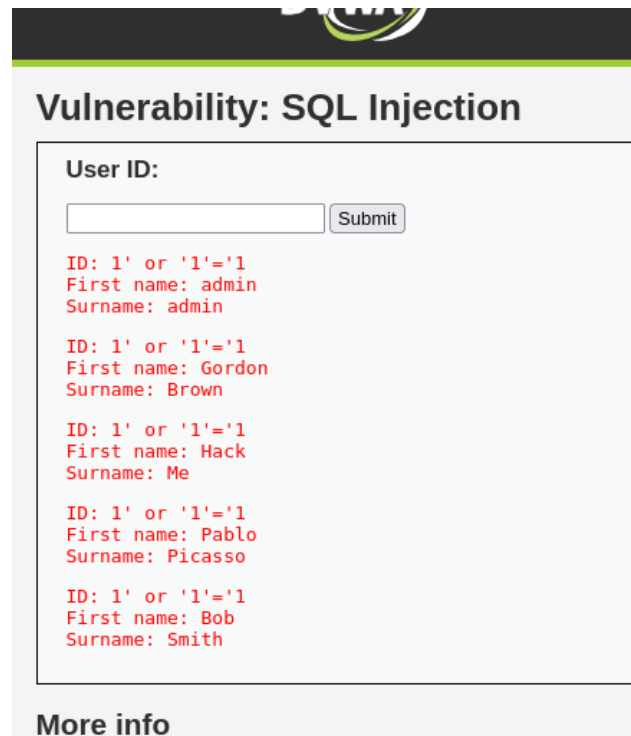
È possibile iniettare una tautologia immettendo questo tipo di input: 1' or '1'='1

La query risultante è la seguente:

SELECT f1, f2, f3

FROM table

WHERE f1 = '1' OR '1'='1';



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

More info

L'iniezione SQL basata su tautologia ha comunque diverse limitazioni: non permette di dedurre la struttura di una query SQL, non permette di selezionare altri campi rispetto a quelli presenti nella query SQL, non permette di eseguire comandi arbitrari SQL.