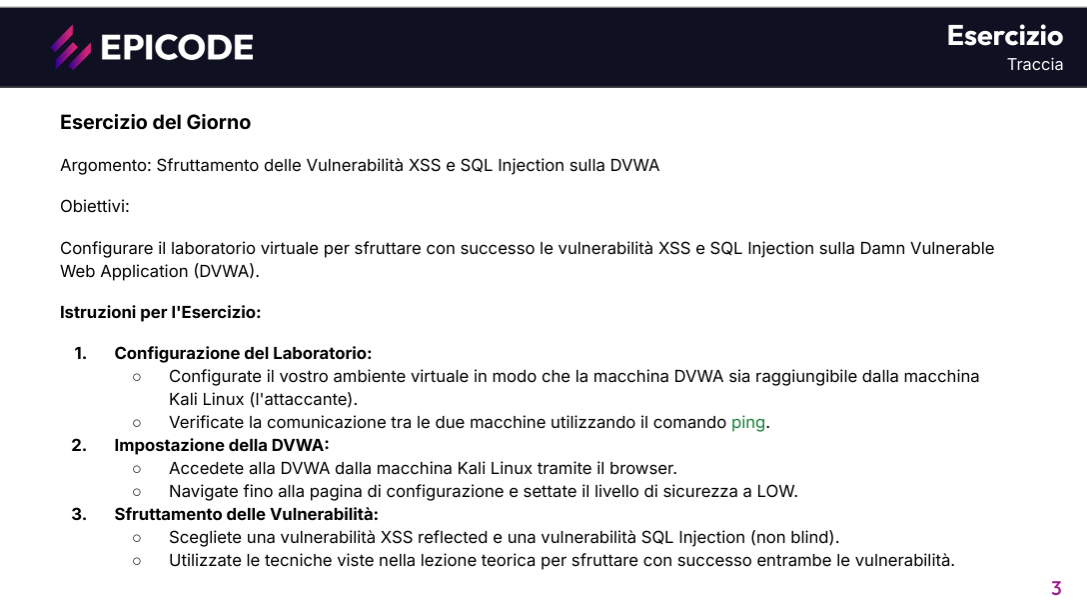
**S6L2**

**XSSeSQL injection**

**Marco Falchi**

**Consegna**

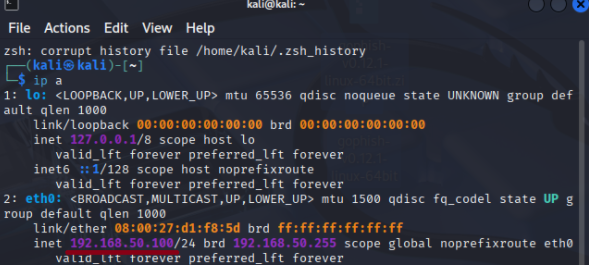


**Svolgimento**

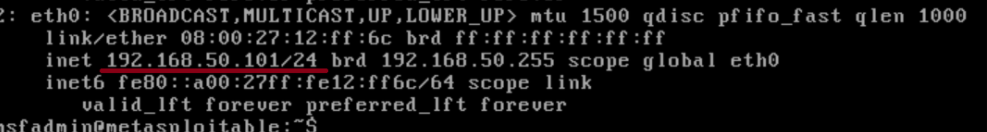
Configurazione del laboratorio:

Ho iniziato lo svolgimento dell’esercizio configurando le macchine virtuali che sono andato ad utilizzare.   
Nello specifico ho impostato Kali Linux e Metasploitable2 sulla **stessa “rete interna”**.

Per far sì che le macchine comunicassero ho impostato in maniera manuale gli indirizzi IP.



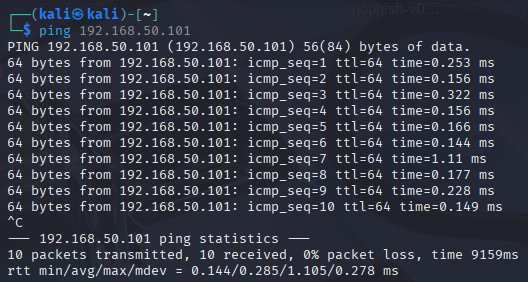
192.168.50.100 per la macchina kali linux

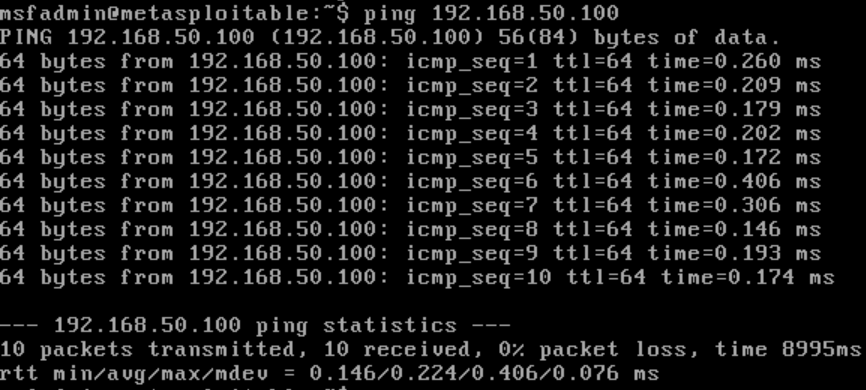


192.168.50.101 per la macchina kali linux

**Verifica delle comunicazioni**

Per verificare l’effettiva connessione fra le due macchine ho eseguito dei test di ping da ambe le parti





Notiamo quindi che le macchine comunicano correttamente senza perdita di pacchetti.

**Configurazione DVWA**



Come da consegna ho impostato la sicurezza su LOW

**Sfruttamento delle vulnerabilità**

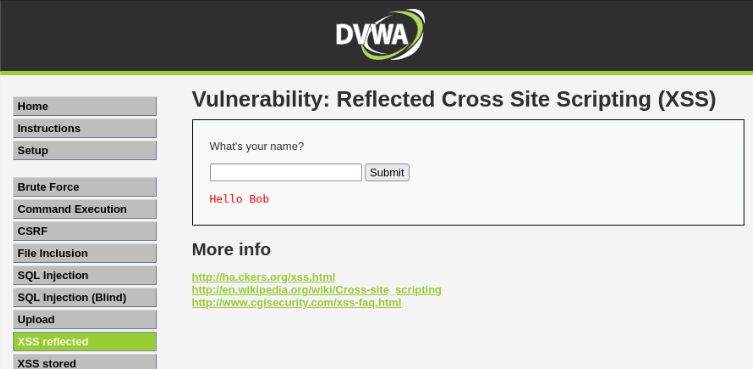
**Vulnerabilità XSS**

Mi sono quindi spostato nella sezione dedicata alle vulnerabilità XSS (cross-site

scripting). Queste vulnerabilità si verificano quando un’applicazione utilizza un

input proveniente dall'utente senza filtrarlo per generare il contenuto da mostrare.

Per prima cosa ho verificato l’output inserendo come input il nome di prova Bob



Come si può vedere, ci viene restituito un messaggio di benvenuto che riporta il

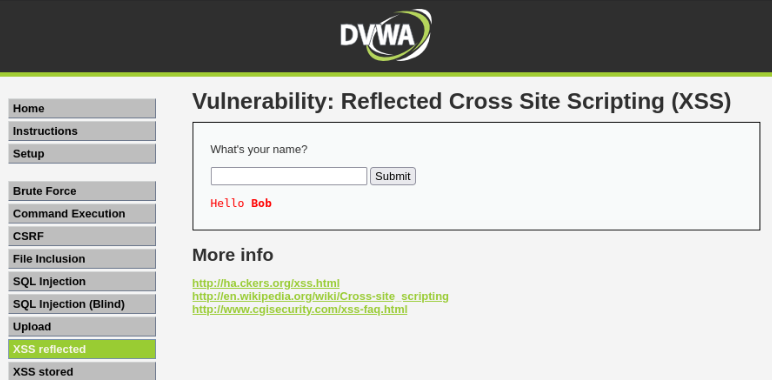
nome passato in input. Osservando l’URL possiamo notare l’intera query

?name=Bob#.

A questo punto ho provato a modificare l’input fornito per testare se venisse

eseguito come codice. Ad esempio ho provato ad inserire nuovamente Il nome Bob ma in grassetto tramite i tag HTML <b> e </b>.

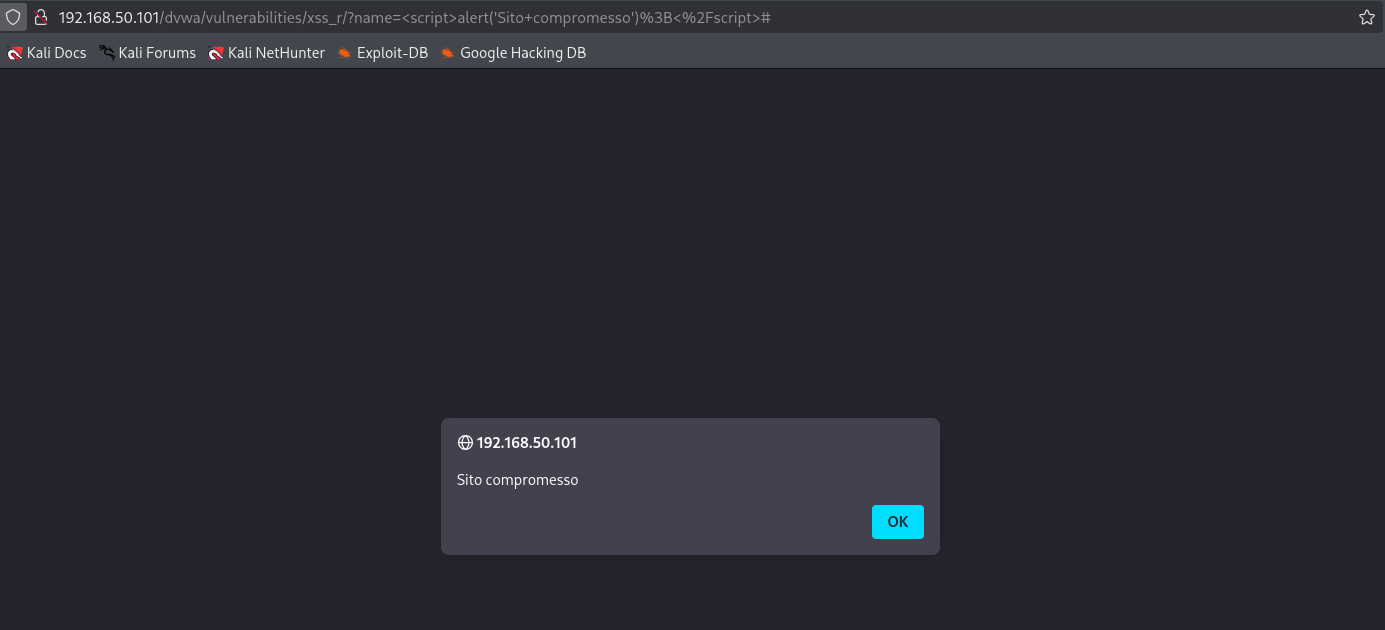
Come mostra l’immagine che segue, l’input viene preso ed eseguito come se fosse uno script, fornendoci in output il nome in grassetto.



Ho poi eseguito uno script che permette di far apparire una finestra

di messaggio all’utente inserendo il testo “Sito compromesso” tramite

<script>alert(‘Sito compromesso’);</script>

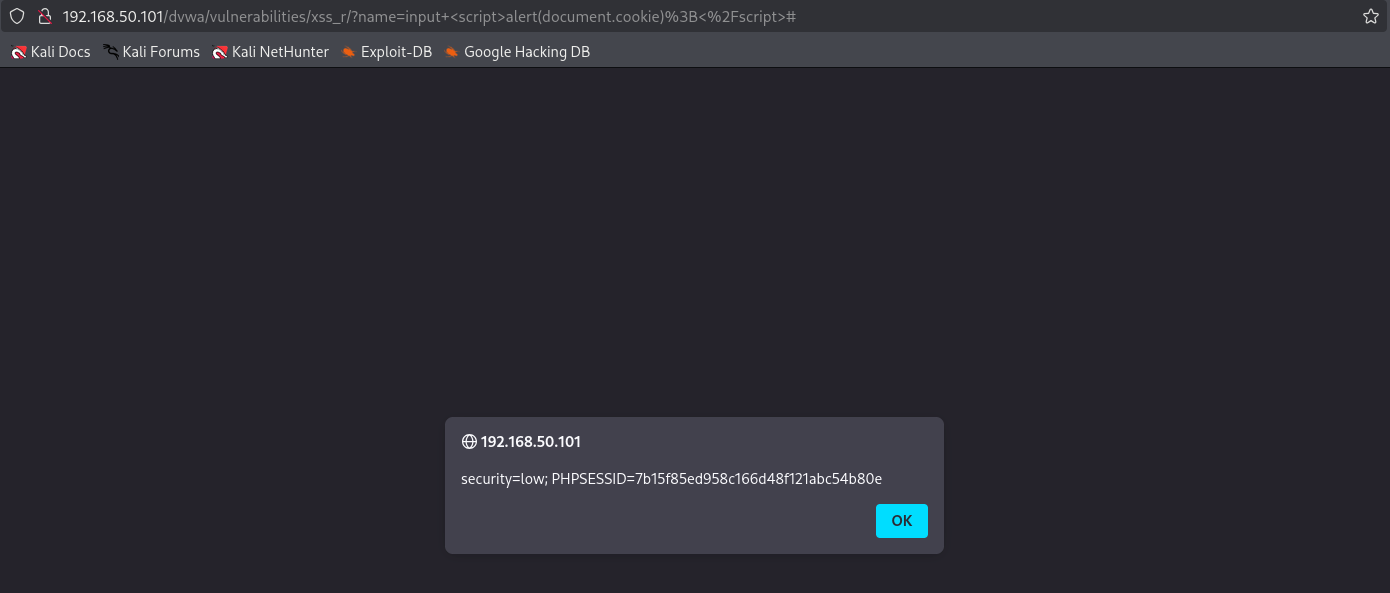


Questo è solo un messaggio innocuo ma possiamo utilizzare questa

tecnica per visualizzare il codice di sessione PHP, detto anche token di sessione, ovvero il codice univoco utilizzato da PHP per tenere traccia dell’utente durante la navigazione in modo che questo non debba ogni volta reinserire la password a ogni passaggio o cambio pagina.

Ho dunque inserito nel campo di input <script>alert(document.cookie);</script>

ottenendo in output il codice di sessione.



Il comando document.cookie restituisce una stringa contenente tutti i cookie

associati al dominio della pagina corrente che potremmo utilizzare per compiere ulteriori attacchi.

**Vulnerabilità SQL injection**

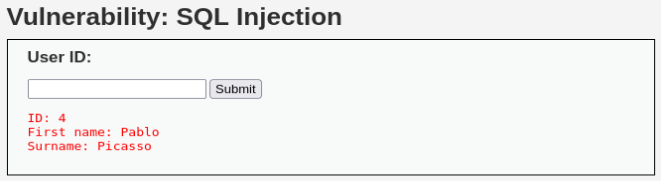
Mi sono spostato poi nella sezione SQL injection della DVWA per testare la

vulnerabilità delle applicazioni web che, non testando e filtrando l’input dell’utente, permette di eseguire comandi sui database di backend.

Ho provato ad inserire diversi input, se inseriamo ad esempio un numero

compreso tra 1 e 5 otteniamo risposta nome e cognome dell’id inserito:

ID 4:



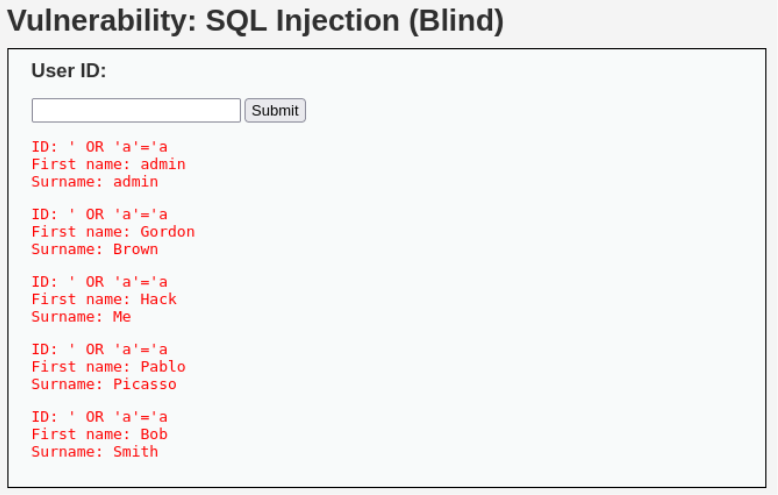
SQL utilizza dei comandi per gestire i database di backend.

Ho ipotizzato dunque che esistesse una query **SELECT FirstName,Surname FROM tables WHERE id='4’.**

* Ho quindi provato ad inserire un apice come input e viene mostrato un errore.
* Ho provato poi ad inserire le classiche query che vengono usate per bypassare i controlli come ad esempio **' OR 'a'='a** ovvero un **valore sempre uguale a True**.

In questo modo, il programma restituisce in output tutto il database con nome e

cognome degli utenti elencati in table.



Dato che i database contengono le password degli utenti, ho provato a concatenare il comando precedente con un un’altra query tramite **UNION** inserendo la query

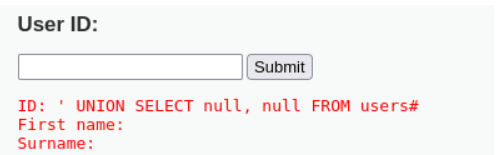
**’ UNION SELECT null FROM users#**

ottenendo come risposta dal server:

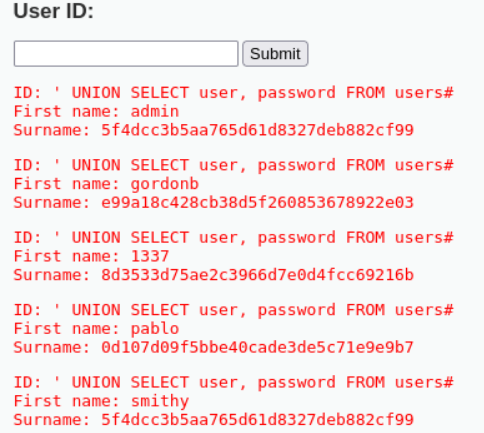


Questo messaggio indica che la tabella selezionata è composta da un numero maggiore di colonne, dunque ho provato il comando

**’ UNION SELECT null, null FROM users#**



Individuato quindi il numero di colonne, ossia **2** ho sostituito i valori null con username, password non ottenendo riscontro, quindi ho provato con user, password ottenendo stavolta l’intera lista delle credenziali presenti nel database:



Invece di andare a tentativi come ho fatto io si poteva fare in casi più complessi anche gli script:

**1' UNION SELECT null, table\_name FROM information\_schema.tables WHERE table\_schema=database() –**

**Che serve ad elencare le tabelle**

E

**1' UNION SELECT null, column\_name FROM information\_schema.columns WHERE table\_name='users' –**

**Che esamina le colonne**

**Conclusione:**

Questo esercizio mi ha insegnato come un piccolo errore di programmazione possa compromettere la sicurezza di un’intera infrastruttura, in questo caso di dvwa ci troviamo in un ambiente di simulazione con sicurezza bassa, quindi fatto apposta per essere “bucato” ma in casi reali mi fa pensare quanto bisogna essere scrupolosi e attenti.