

REPORT ESERCITAZIONE: Web Application Pentesting (XSS & SQLi)

Studente: Rocco Paolo Caccamo

Corso: Cybersecurity Specialist

Data: 13 Gennaio 2026

Target: DVWA (Metasploitable 2)

Obiettivo: Identificazione e sfruttamento di vulnerabilità Cross-Site Scripting (Reflected) e SQL Injection.

1. Executive Summary

L'attività laboratoriale ha avuto lo scopo di analizzare la sicurezza della *Damn Vulnerable Web Application* (DVWA) configurata con livello di sicurezza "Low". Sono state testate con successo due classi di vulnerabilità:

1. **Reflected XSS:** È stata dimostrata la possibilità di iniettare ed eseguire codice JavaScript arbitrario nel browser della vittima.
2. **SQL Injection:** È stato possibile manipolare le query al database backend tramite l'iniezione di operatori logici, evidenziando una mancata sanitizzazione degli input.

2. Configurazione del Laboratorio (Setup)

Come da requisiti preliminari, è stata verificata la connettività tra la macchina attaccante (Kali Linux) e il server target (Metasploitable).

- **Verifica Connettività (Kali -> Target):** Il ping verso **192.168.50.10** ha confermato che il target è raggiungibile.

```

kali@kali: ~
Session Actions Edit View Help
(kali@kali)~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:1f:b7:23 brd ff:ff:ff:ff:ff:ff
    inet 192.168.50.12/24 brd 192.168.50.255 scope global noprefixroute eth0
        valid_lft forever preferred_lft forever
    inet6 fe80::d09:76f5:93bf:8caf/64 scope link noprefixroute
        valid_lft forever preferred_lft forever

(kali@kali)~$ ping 192.168.50.10
PING 192.168.50.10 (192.168.50.10) 56(84) bytes of data:
64 bytes from 192.168.50.10: icmp_seq=1 ttl=64 time=0.154 ms
64 bytes from 192.168.50.10: icmp_seq=2 ttl=64 time=0.181 ms
64 bytes from 192.168.50.10: icmp_seq=3 ttl=64 time=0.134 ms
^C
--- 192.168.50.10 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2035ms
rtt min/avg/max/mdev = 0.134/0.156/0.181/0.019 ms
(kali@kali)~$

```

(Figura 1: Test di connettività ICMP dalla macchina attaccante)

- **Verifica Bidirezionale (Target -> Kali):** Per garantire la stabilità di eventuali reverse shell o connessioni in uscita, è stato verificato il ping anche dalla macchina Metasploitable verso Kali (**192.168.50.12**).

```

metasploitable login: msfadmin
Password:
Last login: Mon Jan 12 10:06:13 EST 2026 on tty1
Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To access official Ubuntu documentation, please visit:
http://help.ubuntu.com/
No mail.
msfadmin@metasploitable:~$ ping 192.168.50.12
PING 192.168.50.12 (192.168.50.12) 56(84) bytes of data:
64 bytes from 192.168.50.12: icmp_seq=1 ttl=64 time=10.1 ms
64 bytes from 192.168.50.12: icmp_seq=2 ttl=64 time=0.201 ms
64 bytes from 192.168.50.12: icmp_seq=3 ttl=64 time=0.357 ms
--- 192.168.50.12 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 0.201/3.584/10.194/4.674 ms
msfadmin@metasploitable:~$ _

```

- (Figura 2: Conferma della comunicazione di rete dalla macchina ospite)

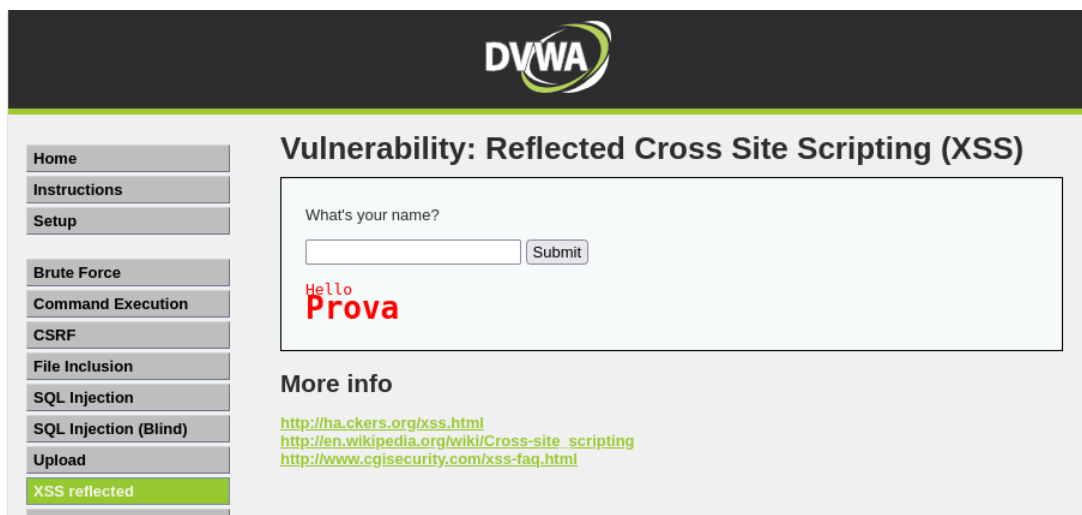
3. Analisi Vulnerabilità: Reflected XSS

La vulnerabilità "Cross-Site Scripting Reflected" si verifica quando l'input dell'utente viene riflesso immediatamente nella risposta del server senza adeguata validazione.

3.1 Identificazione del Reflection Point

Navigando nella sezione "XSS Reflected" della DVWA, è stato identificato un campo di input che chiede "What's your name?". Inserendo una stringa innocua in HTML ("

#

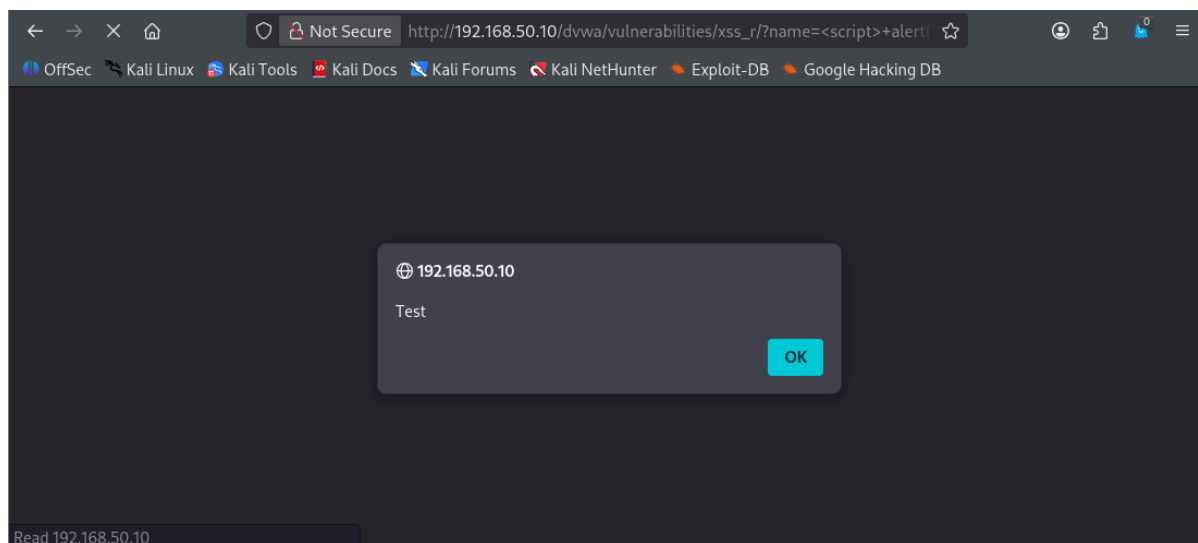


(Figura 3: Verifica del punto di riflessione dell'input utente nell'HTML della pagina)

3.2 Exploitation (Proof of Concept)

Per dimostrare l'esecuzione di codice arbitrario, è stato iniettato un payload JavaScript standard: `<script>alert('Test')</script>`

L'applicazione non ha filtrato i tag `<script>`, portando il browser a interpretare ed eseguire il codice, visualizzando un popup di avviso. Questo conferma la vulnerabilità e la possibilità di attacchi più gravi come il furto di cookie di sessione.



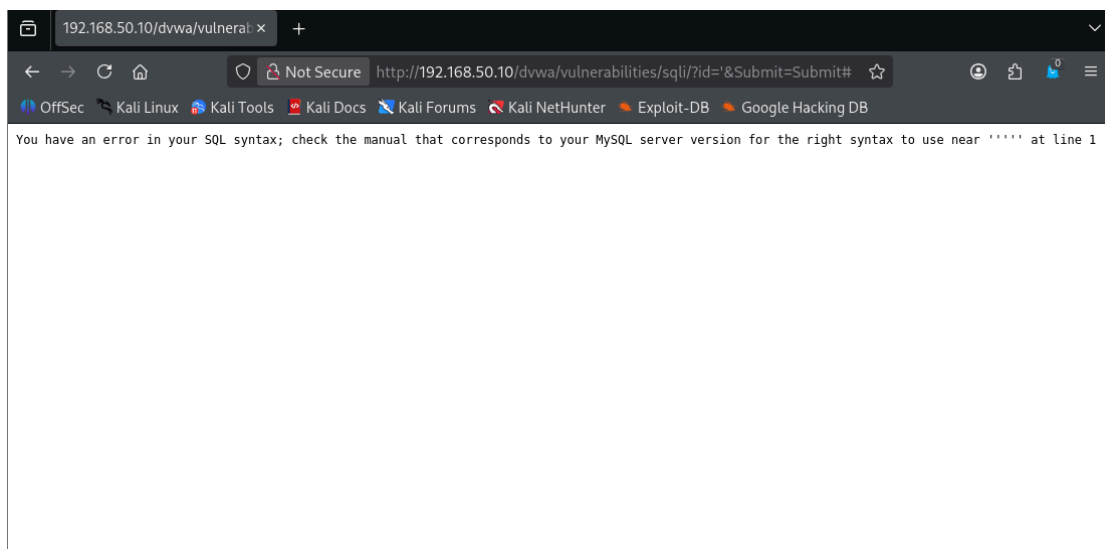
(Figura 4: Esecuzione del payload JavaScript, evidenza del successo dell'attacco XSS)

4. Analisi Vulnerabilità: SQL Injection

La SQL Injection (SQLi) permette a un attaccante di interferire con le query che l'applicazione fa al suo database.

4.1 Rilevamento (Error Based)

Per testare la robustezza del campo "User ID", è stato inserito un singolo apice ' . Questo carattere è un delimitatore di stringa in SQL. L'applicazione ha restituito un errore di sintassi MySQL ("You have an error in your SQL syntax..."). Questo errore è indicativo: dimostra che l'input utente viene concatenato direttamente nella query senza sanitizzazione, rompendo la struttura logica della richiesta.



(Figura 5: L'errore di sintassi SQL conferma che l'input utente viene interpretato dal DBMS)

4.2 Exploitation (Boolean Based / Tautology)

Identificata la vulnerabilità, è stato costruito un payload basato su una tautologia (una condizione sempre vera) per manipolare la clausola **WHERE** della query originale.

Payload Utilizzato: ' OR '1'='1 #

Analisi del Payload:

- ' : Chiude il campo ID originale.
- OR : Operatore logico.
- '1'='1 : Condizione sempre vera (Tautologia).
- # : Carattere di commento in MySQL, che ignora tutto il resto della query originale per evitare errori di sintassi.

La query risultante diventa logicamente equivalente a "Seleziona tutto se l'ID è vuoto OPPURE se 1 è uguale a 1", costringendo il database a restituire tutti i record presenti nella tabella utenti.



(Figura 6: Inserimento del payload SQLi progettato per bypassare i controlli e scaricare il database)

5. Remediation (Soluzioni Consigliate)

Per mettere in sicurezza l'applicazione (Blue Team), si raccomandano le seguenti azioni:

1. Prevenzione XSS:

- **Output Encoding:** Codificare i caratteri speciali (come `<`, `>`, `&`, `"`) in entità HTML prima di visualizzarli nel browser (es. usando `htmlspecialchars()` in PHP).
- **Input Sanitization:** Validare rigorosamente i dati in ingresso.

2. Prevenzione SQL Injection:

- **Prepared Statements:** Utilizzare query parametrizzate. Questo assicura che il database tratti l'input dell'utente sempre come dati e mai come codice eseguibile, neutralizzando l'iniezione di apici o operatori logici.
- **Input Validation:** Assicurarsi che se il campo richiede un ID numerico, vengano accettati solo numeri interi.