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

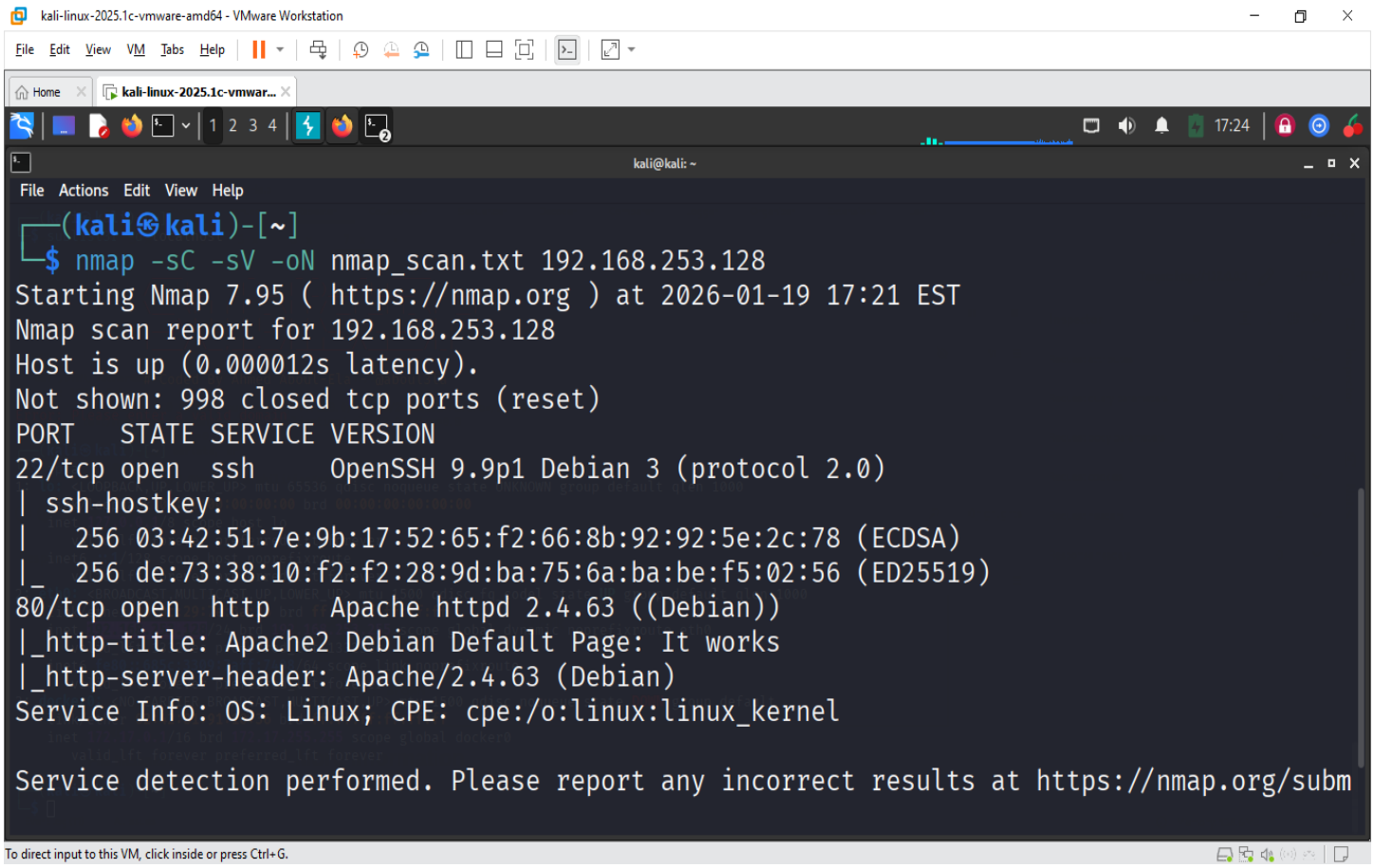
Objective: To enumerate services and subdomains to identify attack vectors.

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).



➢ 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.

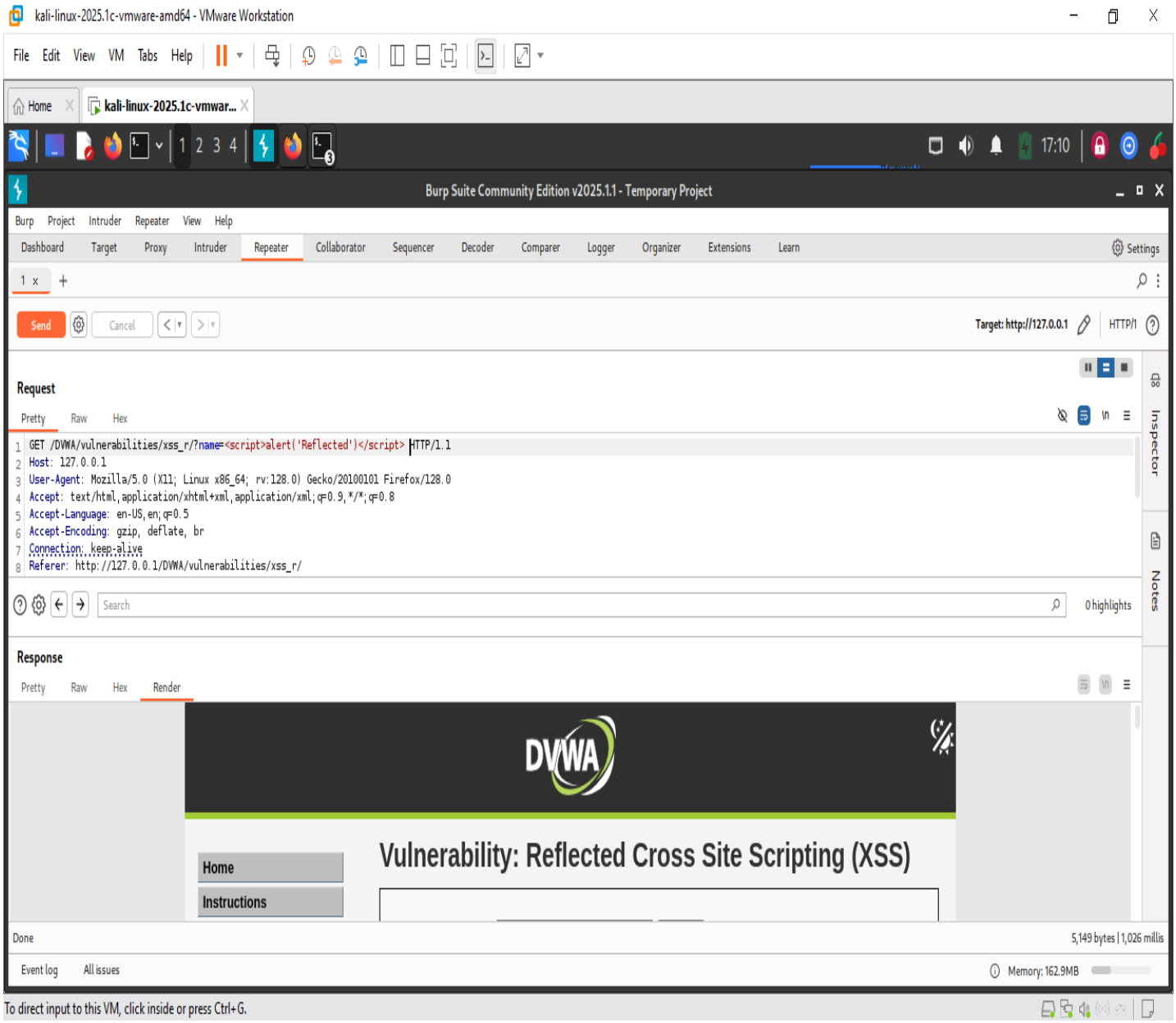
Location: /vulnerabilities/xss_r/

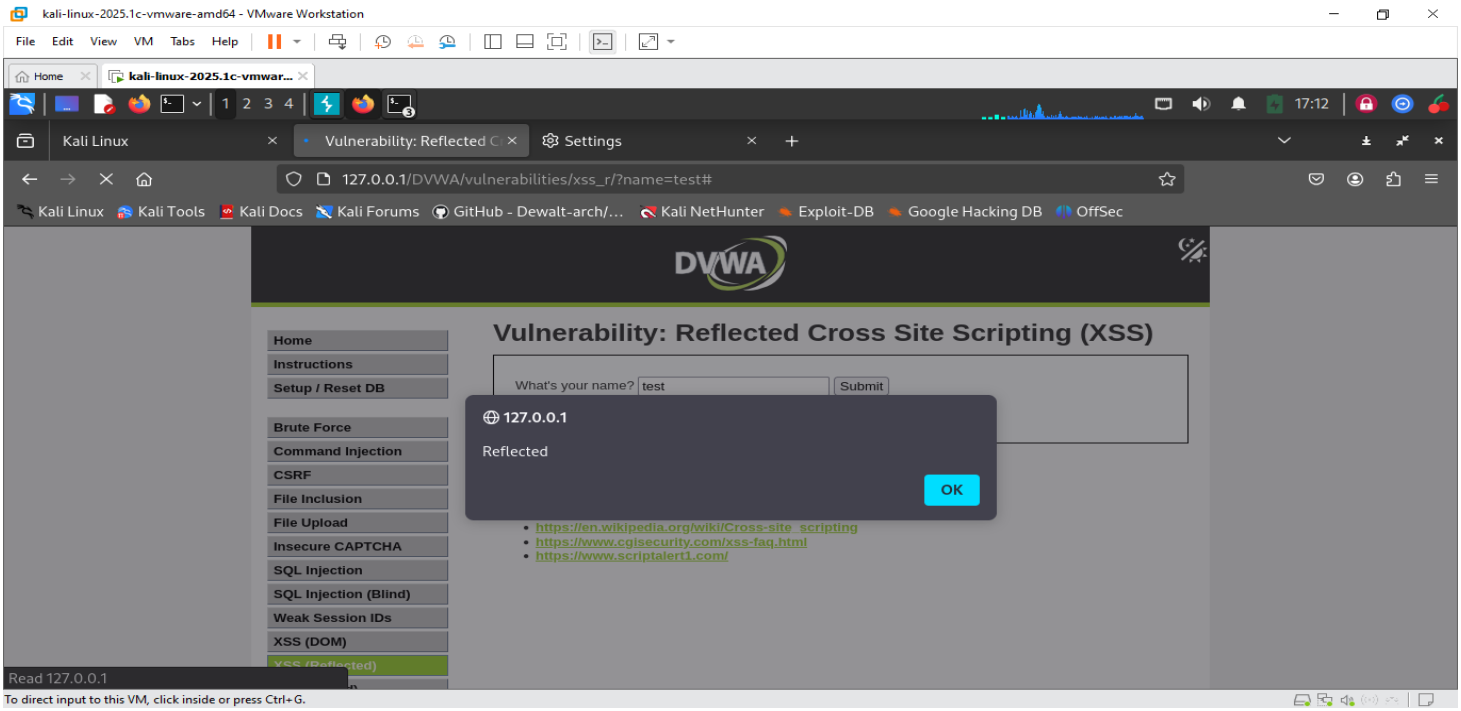
Payload: <script>alert('Reflected XSS')</script>

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:





[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:

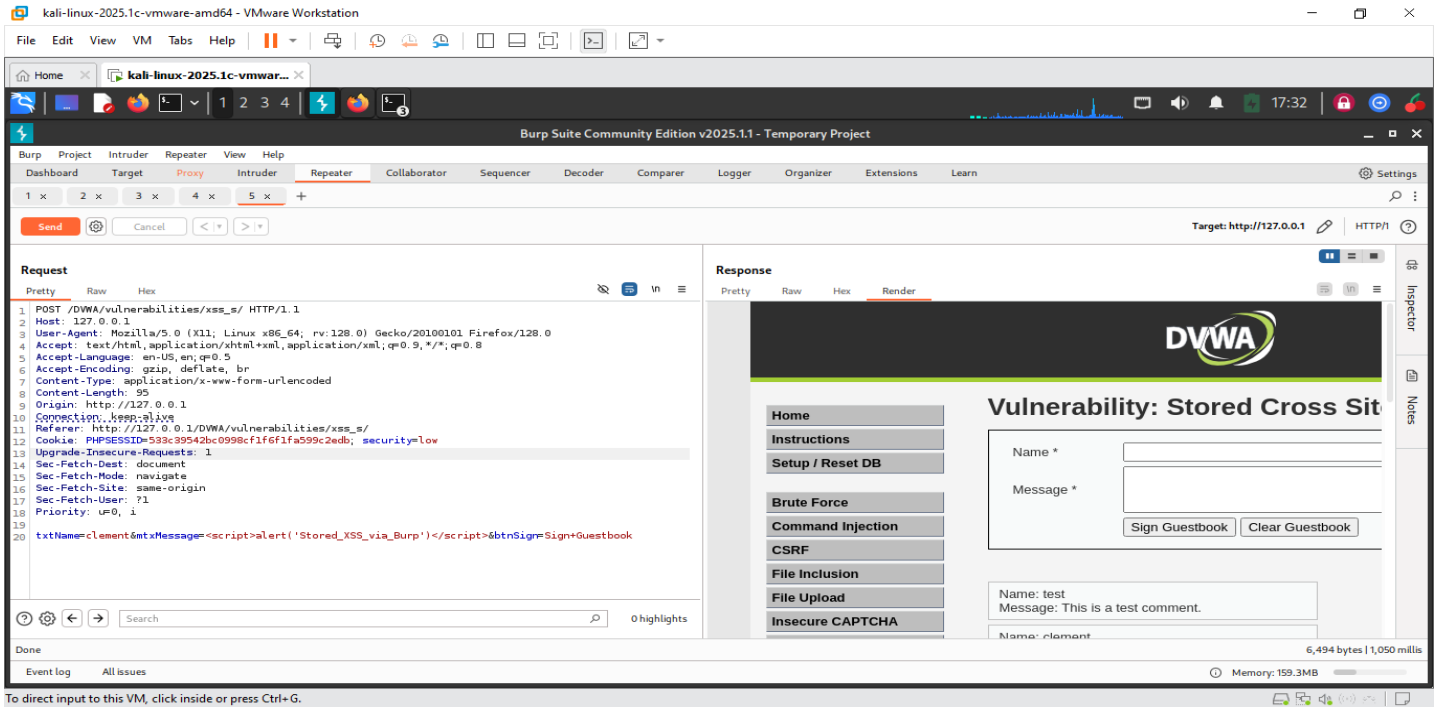
1. I entered a standard name and message.

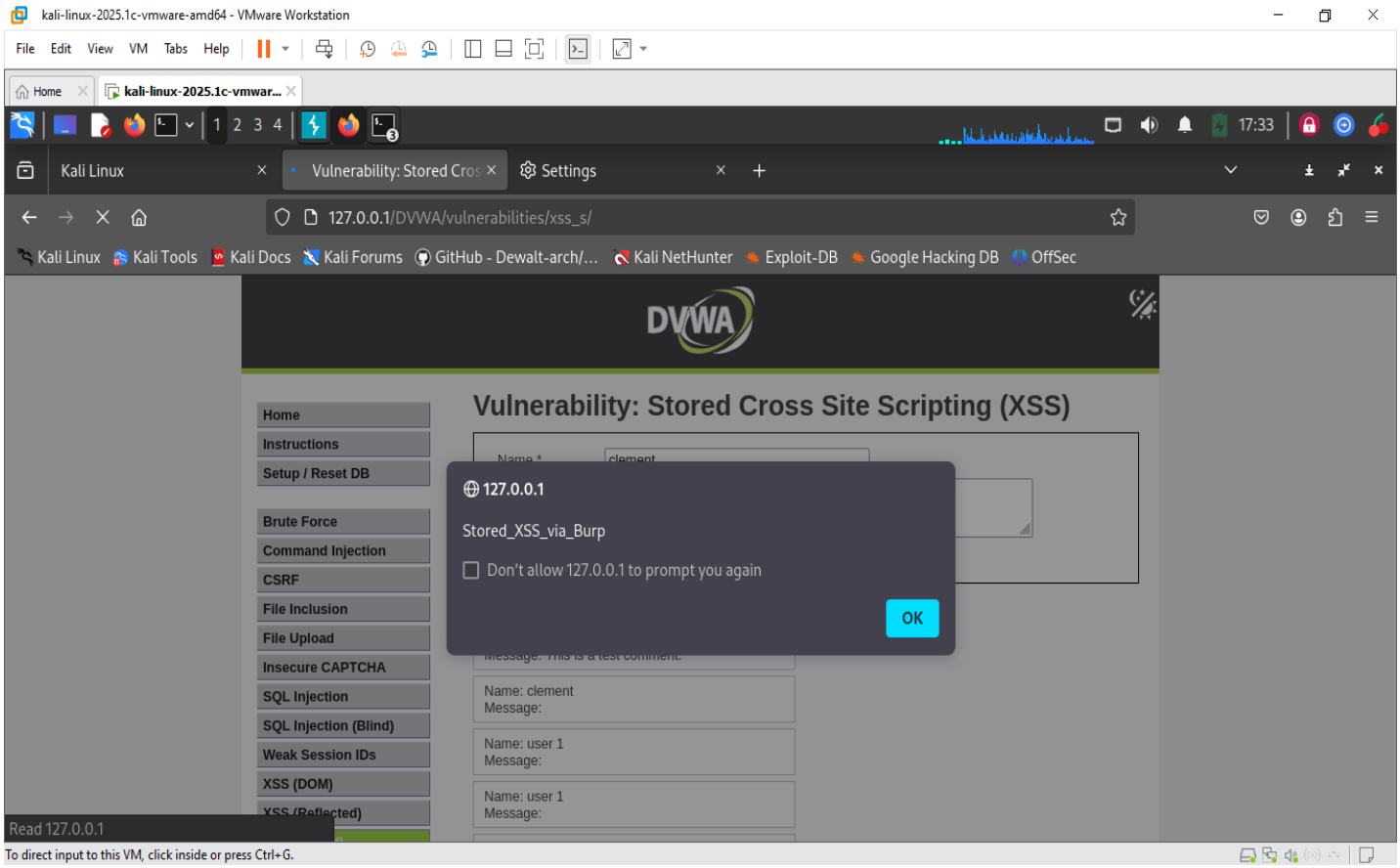
2. I intercepted the POST request in Burp Suite.

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:





- **DOM-Based XSS**

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

Location: /vulnerabilities/xss_d/

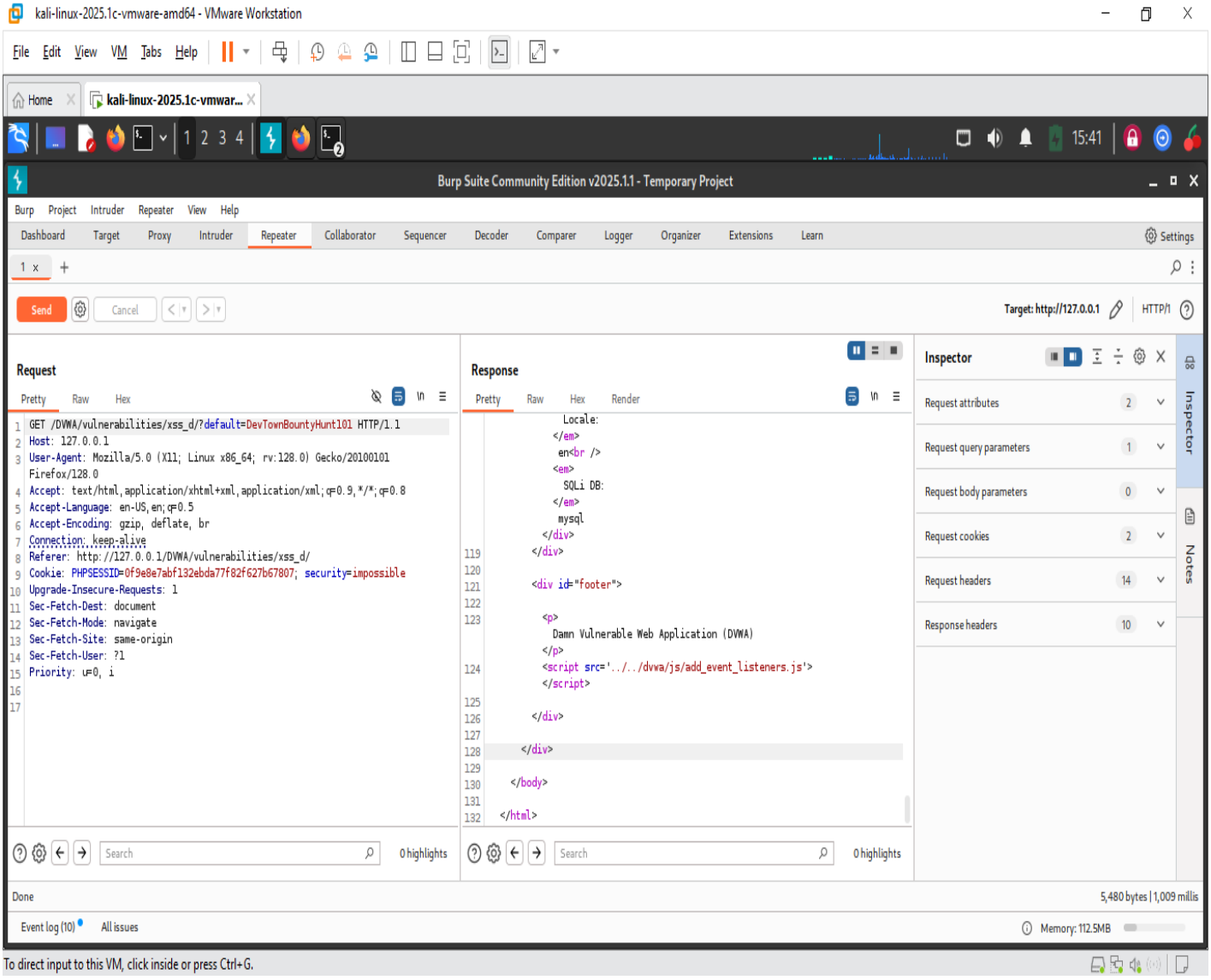
Payload: ?default=<script>alert('DOM')</script>

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.

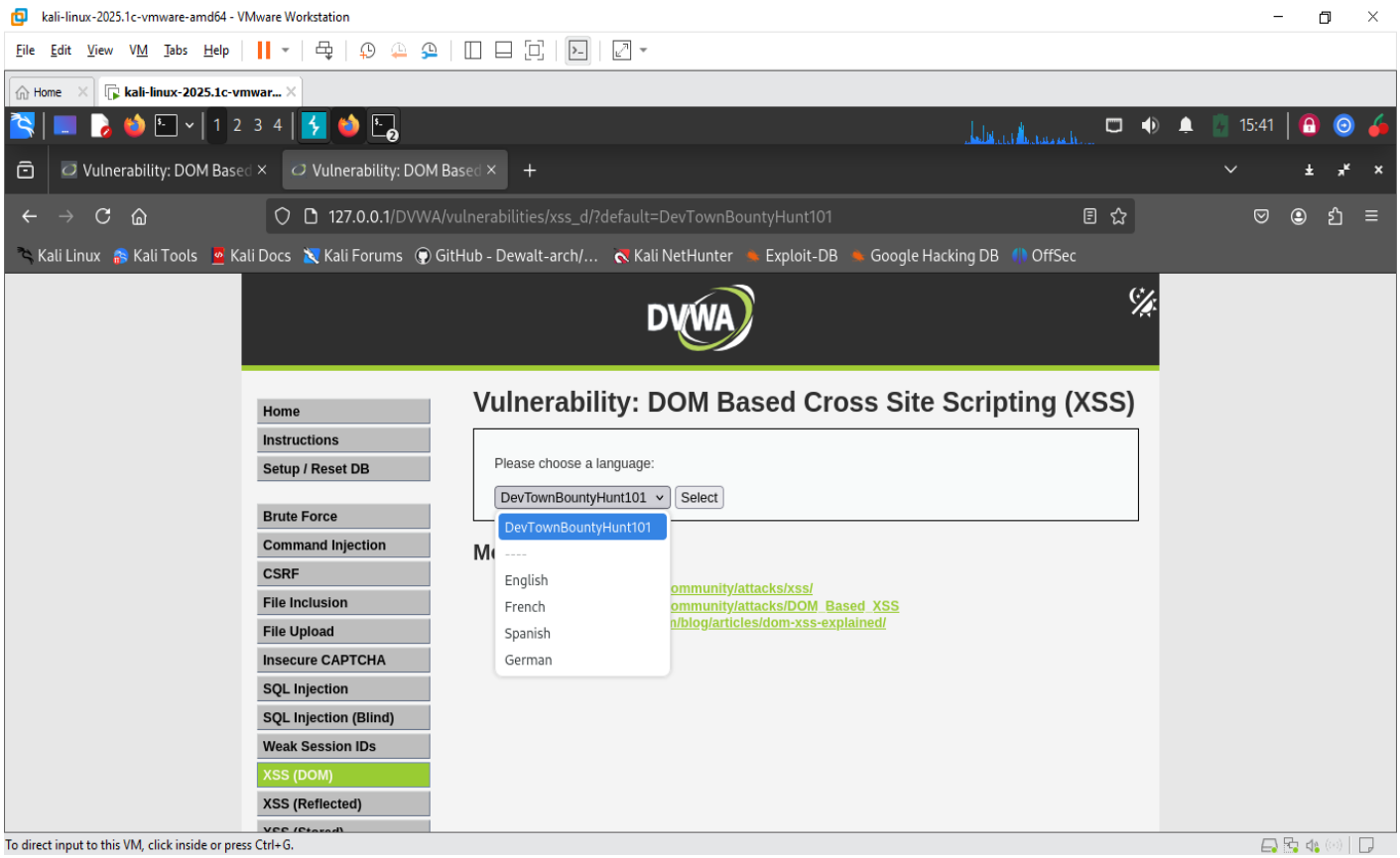
Proof of Concept:


kali-linux-2025.1c-vmware-amd64 - VMware Workstation
File Edit View VM Tabs Help
Home
kali-linux-2025.1c-vmwar...
15:41
Burp Suite Community Edition v2025.1.1 - Temporary Project
Burp Project Intruder Repeater View Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn
Settings
1 x
Send Cancel
Target: http://127.0.0.1
HTTP/1
Request
Pretty Raw Hex
GET /DVWA/vulnerabilities/xss_d/?default=DevTownBountyHunt101 HTTP/1.1
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_d/
Cookie: PHPSESSID=0f9e8e7abf132ebda77f82f627b67807; security=impossible
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Priority: u=0, i
Search
0 highlights
Response
Pretty Raw Hex Render
Locale:
en
SQLi DB:
mysql

Damn Vulnerable Web Application (DVWA)

<script src='../../dvwa/js/add_event_listeners.js'>
</script>

</body>
</html>
Search
0 highlights
Inspector
Request attributes 2
Request query parameters 1
Request body parameters 0
Request cookies 2
Request headers 14
Response headers 10
Inspector
Notes
Done
5,480 bytes | 1,009 millis
Event log (10)
All issues
Memory: 112.5MB
To direct input to this VM, click inside or press Ctrl+G.



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

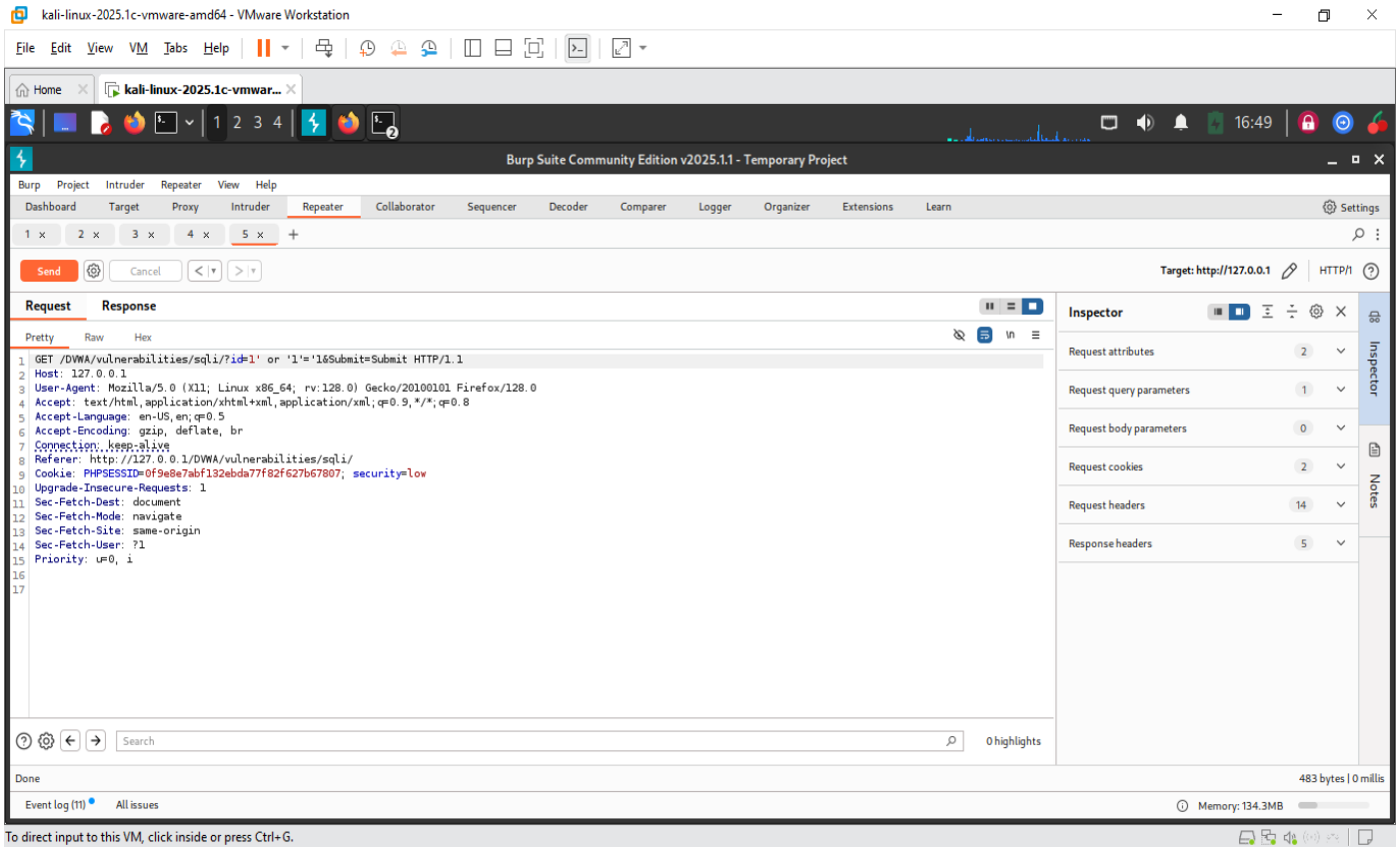
Steps to Reproduce:

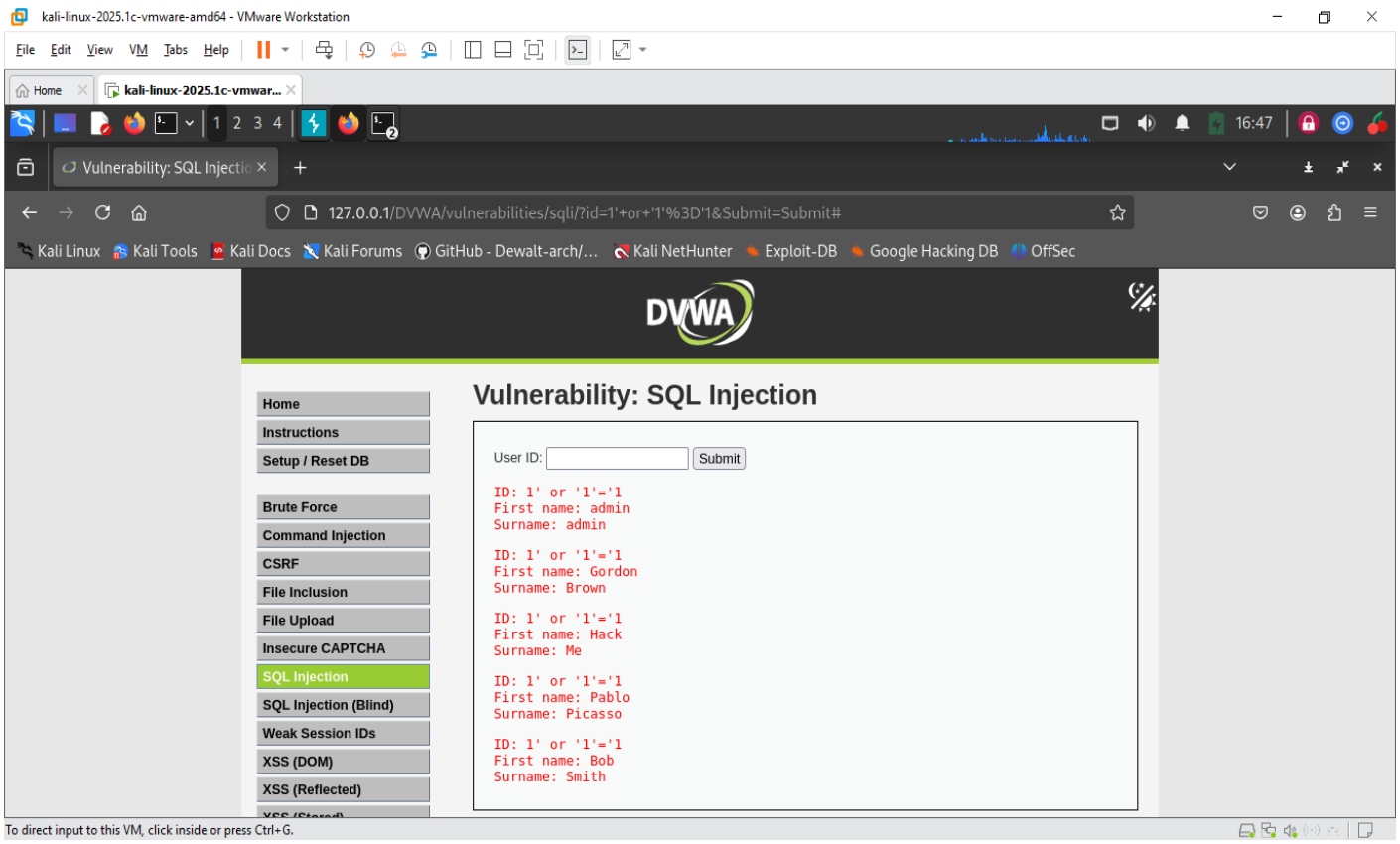
1. I captured the search request in Burp Suite.

2. I sent to Repeater and injected the payload.

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

Proof of Concept:





➢ 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.

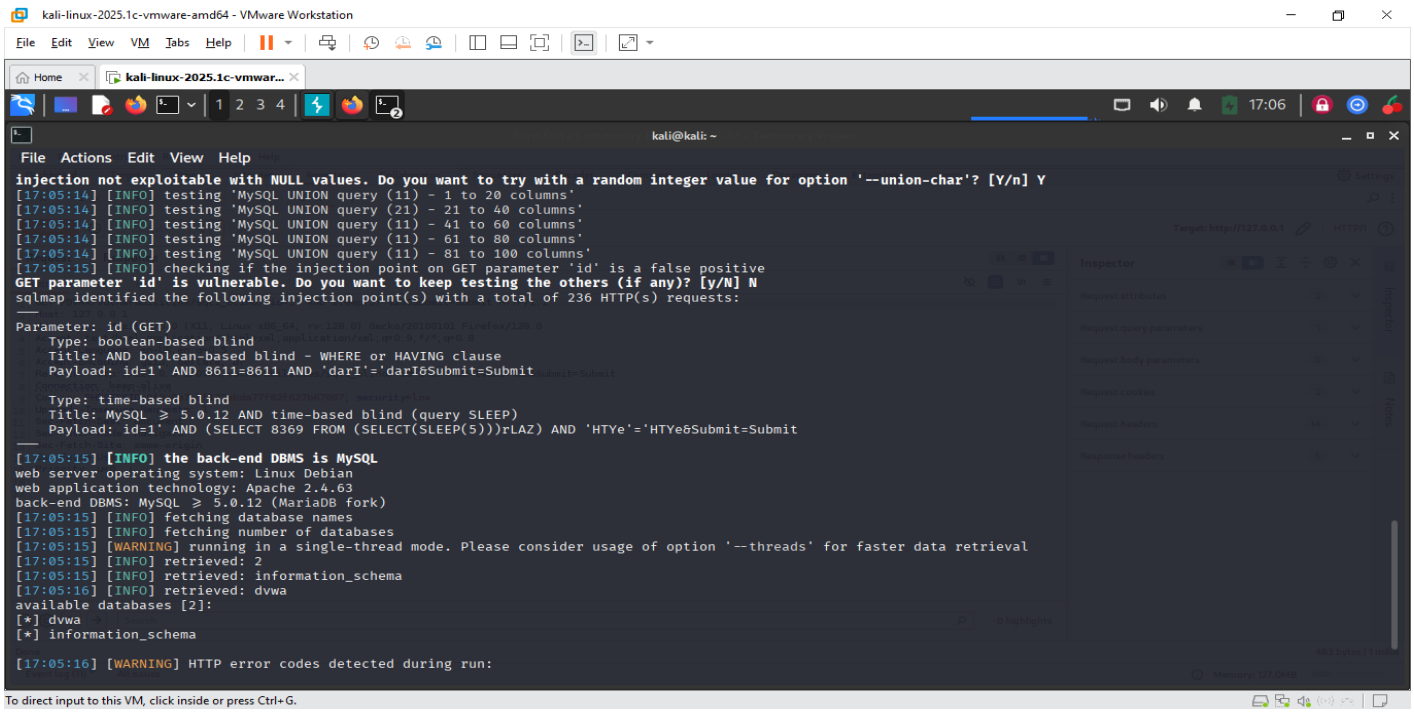
Location: /vulnerabilities/sqli_blind/

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:



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/sqli/ (Simulated IDOR scenario)

Steps to Reproduce:

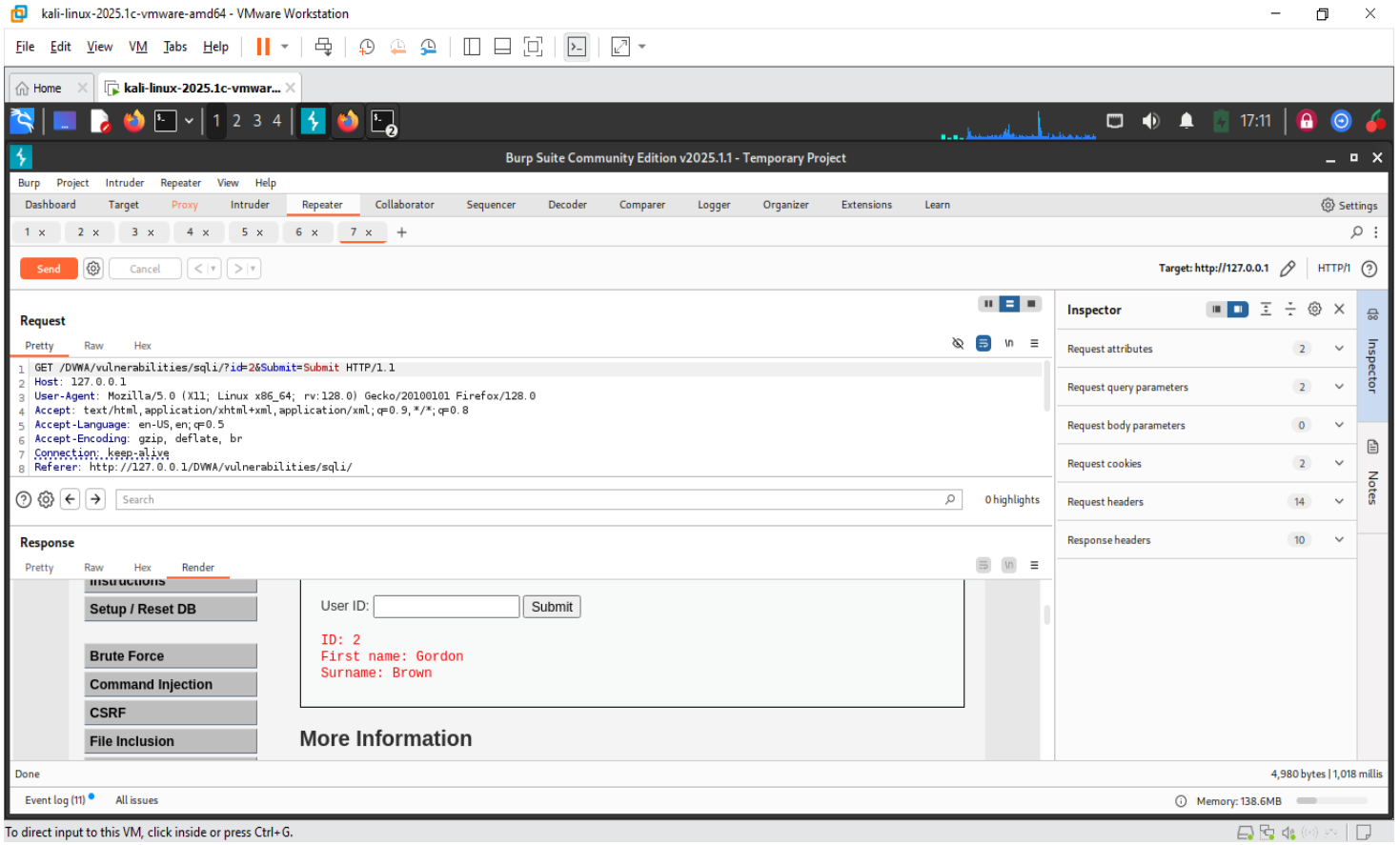
1. I logged in as a standard user (ID 1).

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:



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.