1.1. Findings 001:

Cross-Site Scripting (XSS) Vulnerability

XSS is a class of web vulnerability where an attacker injects malicious script (usually JavaScript) into pages viewed by other users. When the victim's browser executes that script, the attacker can steal cookies, perform actions on behalf of the user, manipulate the page, or phish credentials. The injection is where the q parameter is inserted into the page output without proper escaping (e.g., server template or client innerHTML).

Description	The application is vulnerable to a Reflected Cross-Site Scripting (XSS) attack because the value of the q parameter is inserted into the response without proper output encoding or sanitization. An attacker can craft a malicious URL that executes arbitrary JavaScript in the victim's browser.
Impact	1. Theft of session cookies, tokens, or other sensitive data. 2. Execution of arbitrary JavaScript code in the victim's browser.
URL	http://192.168.254.5:5000/product/3?q=<script>alert("xss")</script>
HTTP Request	GET /product/3?q=%3Cscript%3Ealert(%22xss%22)%3C/script%3E HTTP/1.1 Host: 192.168.254.5:5000 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 Cookie: session=eyJ1c2VyX2lkIjo2fQ.aMUzuA.ZpuEwIgsp_aO9I9EinxfYrX_PDE Upgrade-Insecure-Requests: 1 Priority: u=0, i
CVSS Score	6.1
Vector String	CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:C/C:L/I:L/A:N
Status	unpatched

Evidence:

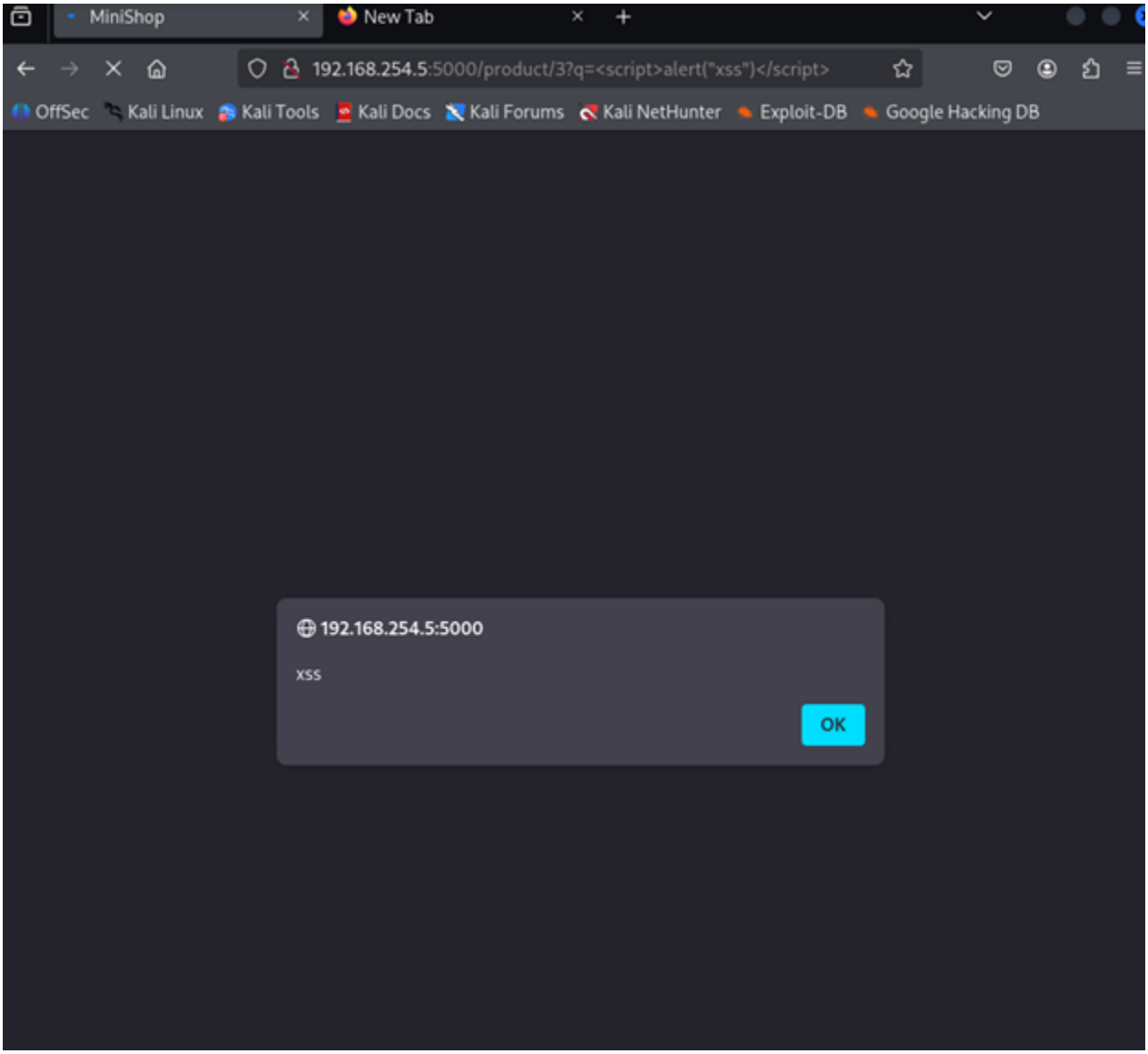


Figure 1: Proof of concept for a reflected XSS vulnerability, demonstrating successful code execution with a pop-up alert.

Remedies:

1. Output Encoding / Escaping: Always escape user-supplied input before rendering it in the HTML response (e.g., use htmlspecialchars() in PHP, escape() in Python/Flask, or template engines with auto-escaping).
2. Use Safe DOM APIs: In client-side code, avoid innerHTML, document.write(), or eval() with untrusted data; instead, use textContent or setAttribute.

1.2. Findings 002:

CSRF (Cross-Site Request Forgery) vulnerability

CSRF (Cross-Site Request Forgery) is a web security vulnerability that tricks an authenticated user into performing an unwanted action on a web application. It works by exploiting the trust a web application has in a user's browser. Unlike other attacks that target the user's credentials, CSRF focuses on forcing the user to submit a forged request without their knowledge or consent.

Description	The application is vulnerable to **Cross-Site Request Forgery (CSRF)**. An attacker can trick an authenticated user into unknowingly submitting unauthorized requests (e.g., changing account settings, transferring funds, or performing administrative actions) by embedding malicious requests into a web page, email, or script. The application does not implement proper CSRF protections such as anti-CSRF tokens or same-site cookie policies.
Impact	1. Unauthorized actions can be performed on behalf of an authenticated user. 2. Compromise of user accounts, including password or email changes.

URL	http://192.168.254.5:5000/checkout
HTTP Request	POST /checkout HTTP/1.1 Host: 192.168.254.5:5000 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 Content-Type: application/x-www-form-urlencoded Content-Length: 24 Origin: http://192.168.254.5:5000 Connection: keep-alive Referer: http://192.168.254.5:5000/checkout Cookie: session=eyJjYXJ0Ijp7fSwidXNlcl9pZCI6N30.aMVImA.8Bd1sIfvYv9A3dPMVc6ZhHv9ghI Upgrade-Insecure-Requests: 1 Priority: u=0, i address=fvfd&phone=rfewg
CVSS Score	6.5
Vector String	CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:N/I:H/A:N
Status	unpatched

Evidence:

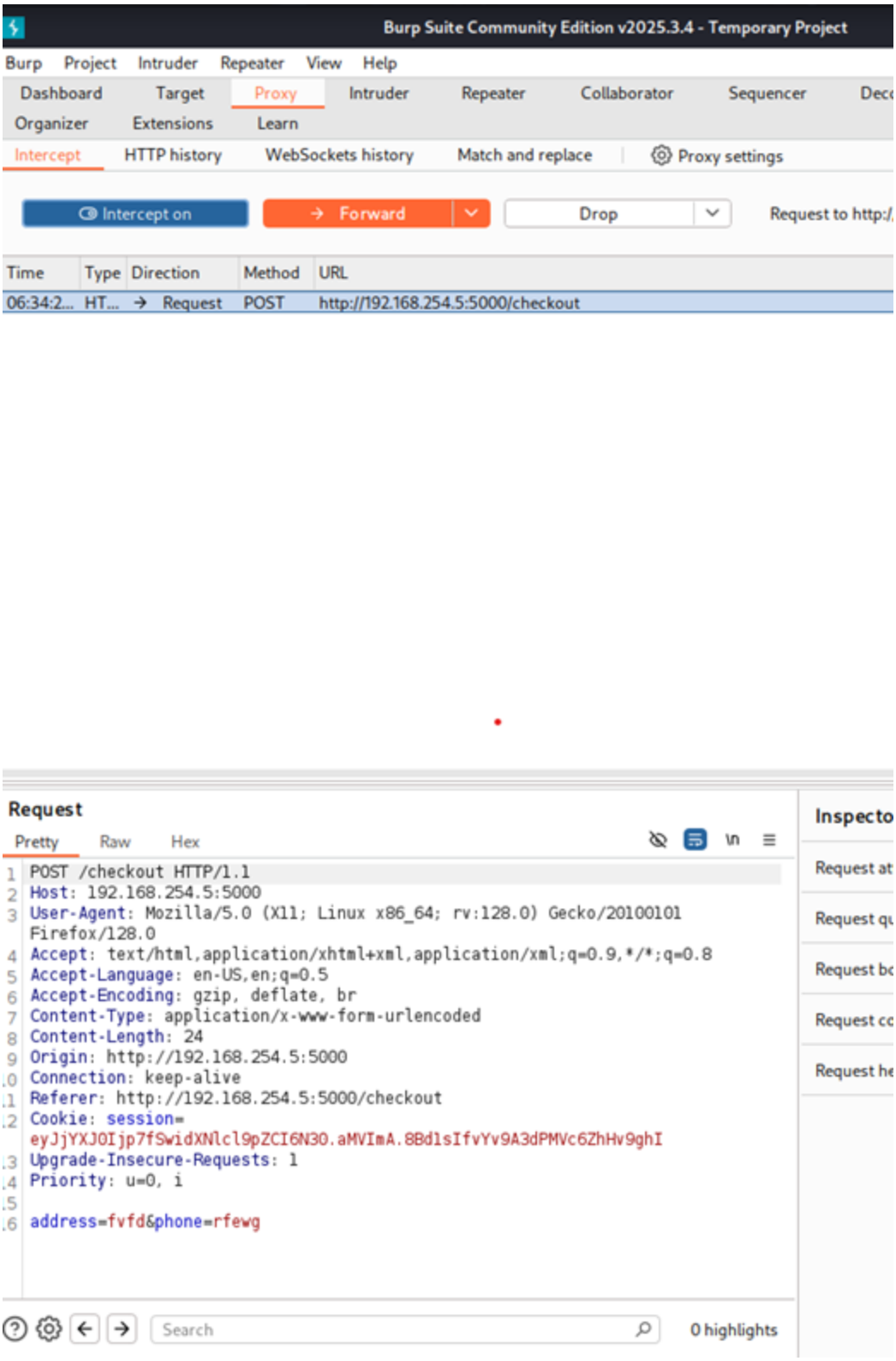


Figure 2: A Burp Suite proxy intercepting a POST request to a /checkout endpoint, showing the request's details and session cookie.

n

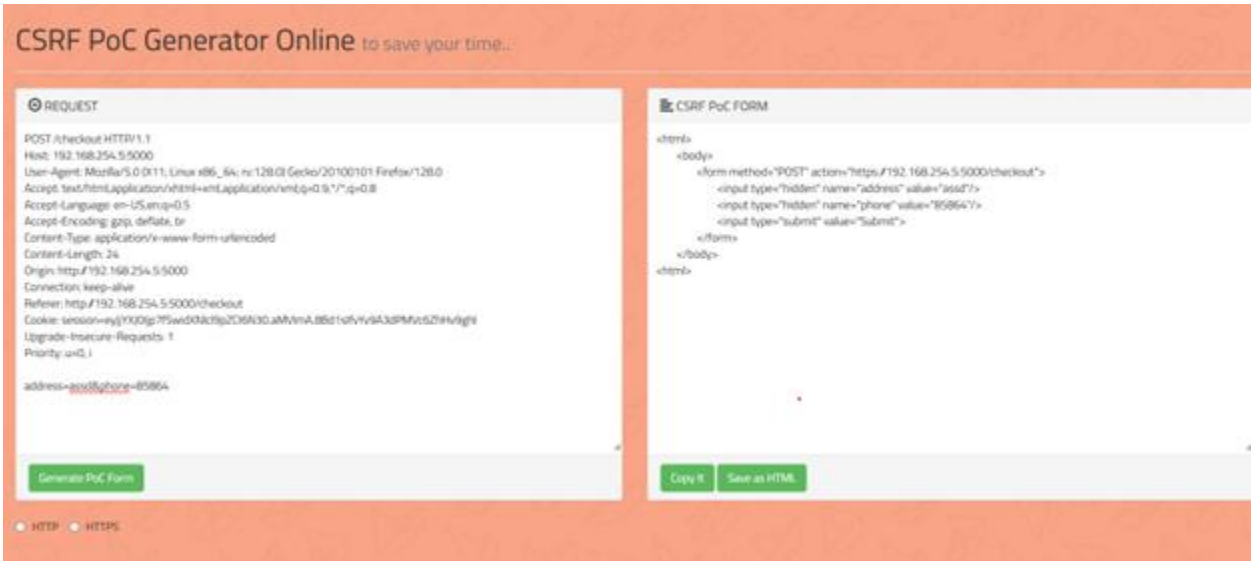


Figure 3 :A CSRF PoC generator tool creating the HTML form to perform a forged POST request.

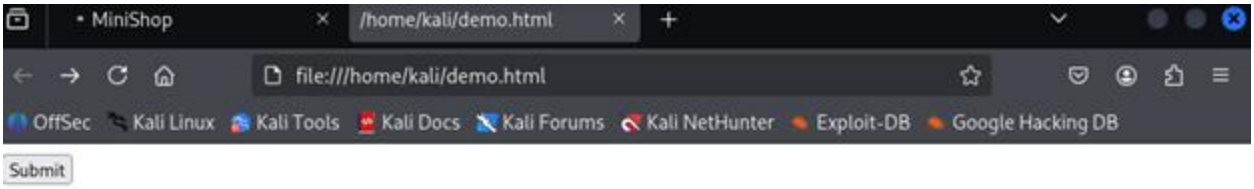


Figure 4: A CSRF form with only a submit button, designed to execute a malicious request when clicked.

Remedies:

1. **CSRF Tokens**: Implement anti-CSRF tokens that are unique, unpredictable, and validated on each sensitive request.
2. **SameSite Cookies**: Configure session cookies with SameSite=strict or SameSite=lax to prevent cross-origin request usage.

1.3. Findings 003:

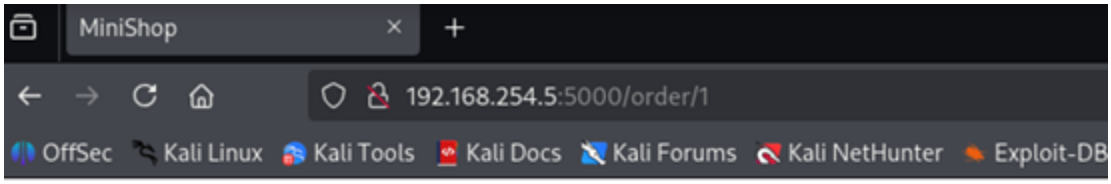
Insecure Direct Object Reference (IDOR) vulnerability

Insecure Direct Object Reference (IDOR) is a type of access control vulnerability where an application exposes internal object identifiers (such as database IDs, filenames, or account numbers) without proper authorization checks. Attackers can manipulate these identifiers to gain unauthorized access to data or perform actions on resources they do not own.

Description	The application is vulnerable to IDOR because it allows direct access to objects (e.g., /product/3, /user/6) based solely on predictable identifiers without verifying the user's authorization. By modifying these identifiers, an attacker can view or manipulate resources belonging to other users.
Impact	1. Unauthorized access to sensitive information (e.g., personal details, financial records). 2. Unauthorized modification or deletion of other users' data.

URL	http://192.168.254.5:5000/order/1
HTTP Request	POST /order/1 HTTP/1.1 Host: 192.168.254.5:5000 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 Cookie: session=eyJjYXJ0Ijp7fSwidXNlcl9pZCI6N30.aMVLrA.uYCLUHXqHBlPkaGxDULQcVOYPQM Upgrade-Insecure-Requests: 1 Priority: u=0, i Content-Type: application/x-www-form-urlencoded Content-Length: 0
CVSS Score	7.5
Vector String	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:N
Status	unpatched

Evidence:



Home | Cart | Orders | Returns | Profile | Logout

Order #1

Product: Red Sneakers

User: alice

Address: 1 Alice St

Phone: 111-1111

Date: 2025-09-12T08:10:32.604631

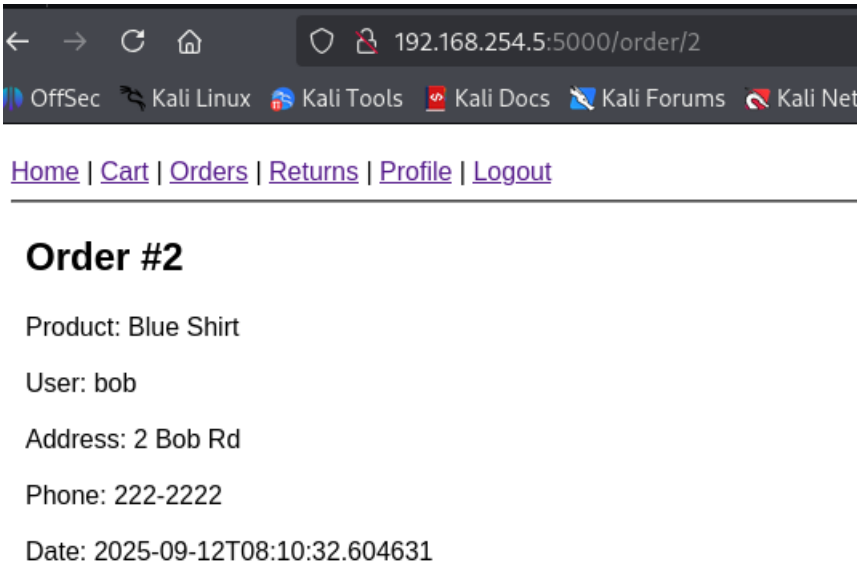


Figure 5: Evidence of an Insecure Direct Object Reference (IDOR) vulnerability, where a user can view another user's order details by simply changing the id parameter in the URL from 1 to 2

Remedies:

1. **Implement Authorization Checks**: Always verify that the current user has permission to access or modify the requested object.
2. **Use Indirect References**: Replace direct IDs with randomized or mapped values (e.g., UUIDs, opaque references).

1.4. Findings 004:

SQL Injection (SQLi) Vulnerability

SQL Injection (SQLi) is a critical web application vulnerability that occurs when user input is unsafely concatenated into SQL queries. Attackers can exploit this flaw to read, modify, or delete database data, bypass authentication, or even take full control of the backend system. Tools like sqlmap automate the detection and exploitation of SQL injection vulnerabilities.

Description	The application is vulnerable to SQL Injection because it fails to properly sanitize and parameterize user input in database queries. This allows attackers to inject arbitrary SQL code through parameters, which is then executed by the backend database. Testing with sqlmap confirmed the injection point and database information disclosure.
Impact	1. Unauthorized access to sensitive data (usernames, passwords, financial records). 2. Modification or deletion of database entries.
URL	http://192.168.254.5:5000/order?id=1
HTTP Request	GET /order?id=1 HTTP/1.1 Host: 192.168.254.5:5000 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 Cookie: session=eyJjYXJ0Ijp7fSwidXNlcl9pZCI6N30.aMVLrA.uYCLUHXqHBlPkaGxDULQcVOYPQM

	Upgrade-Insecure-Requests: 1 Priority: u=0, i
CVSS Score	9.1
Vector String	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:L
Status	unpatched

Evidence:

Figure 6 :A screenshot from sqlmap demonstrating successful data exfiltration of the orders table from a SQL database.

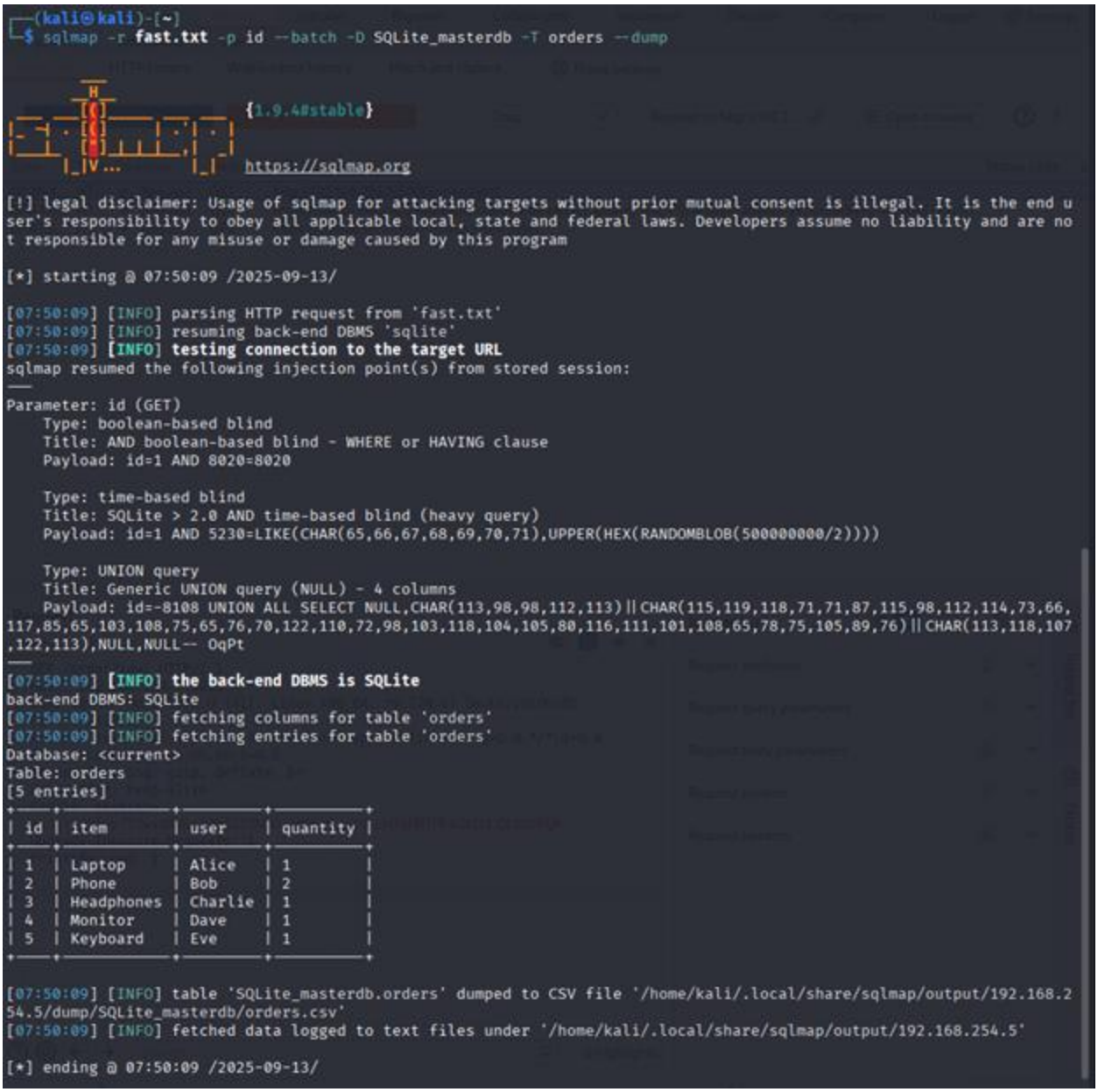


Figure 7: A screenshot from sqlmap identifying and dumping the tables present in a vulnerable SQL database.

Remedies:

1. **Use Parameterized Queries (Prepared Statements)**: Always use parameterized SQL queries or stored procedures instead of string concatenation.
2. **Input Validation**: Apply strict whitelisting for expected input formats (numbers, dates, enums).

1.5. Findings 005:

Local File Disclosure vulnerability

Local File Disclosure (LFD), also called Local File Inclusion (LFI) in some contexts, is a vulnerability where an application improperly handles user-supplied file paths. Attackers can manipulate input to access sensitive files on the server, such as configuration files, password files, or application source code.

Description	The application is vulnerable to Local File Disclosure because it directly uses user-controlled input to access server-side files without proper validation or sanitization. An attacker can modify file path parameters to read arbitrary local files on the server.
Impact	1. Exposure of sensitive system files (e.g., /etc/passwd, configuration files). 2. Disclosure of application source code or credentials.
URL	http://192.168.254.5:5000/hidden/creds.txt
HTTP Request	GET /hidden/creds.txt HTTP/1.1 Host: 192.168.254.5:5000 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 Upgrade-Insecure-Requests: 1 Priority: u=0, i
CVSS Score	7.0
Vector String	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:N
Status	unpatched

Evidence:

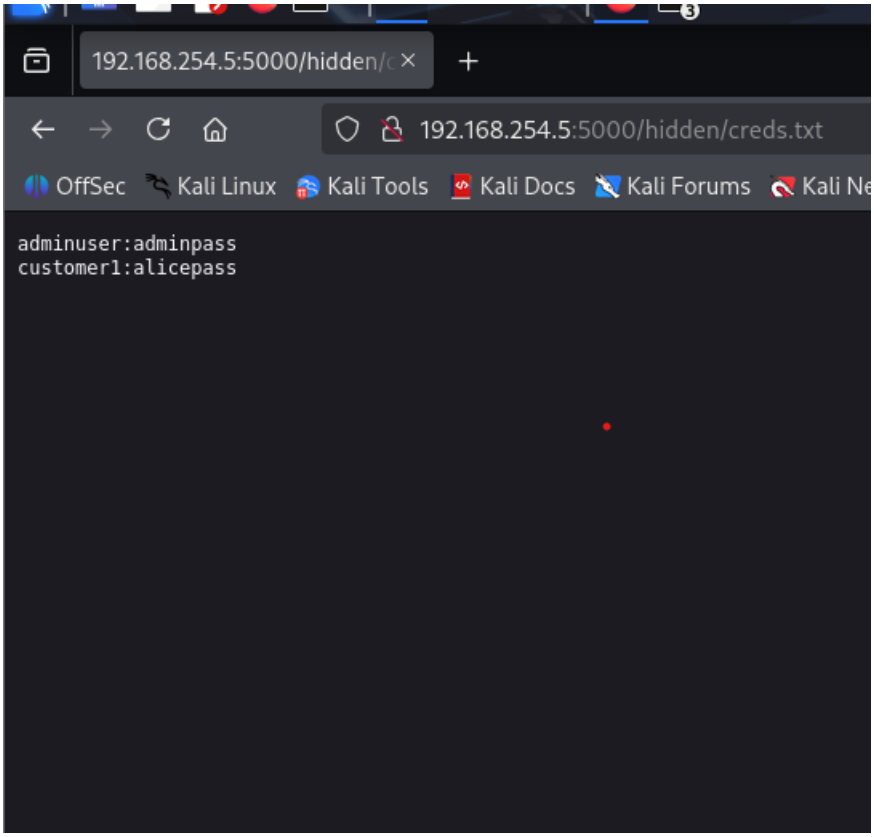


Figure 8: Proof of concept for a Local File Disclosure (LFD) vulnerability, showing a sensitive file (creds.txt) accessed directly via a URL

Remedies:

1. **Input Validation / Whitelisting**: Allow only specific files or directories to be accessed.
2. **Avoid Direct File Inclusion**: Never use user input directly in file system operations.