PW VAPT Assessment & Report

Name : ISSAN PANDA

Email : issanpanda@gmail.com

Cross-Site Scripting (XSS)

[VULN1] Basic Reflected

→ Vulnerability Name & Description

The application reflects user-controlled input from the `q` parameter directly into the HTML response without proper sanitization or encoding. This allows an attacker to inject and execute arbitrary JavaScript code in the victim's browser. The vulnerability results in a Reflected Cross-Site Scripting (XSS) condition.

→ Proof of Concept (PoC)

- User input from q parameter is reflected unsanitized into the HTML output, triggering JavaScript execution.
- Alert popup displayed.

→ Impact Analysis

- Execute arbitrary JavaScript
- Steal session cookies
- Impersonate users
- Modify site content
- Redirect victims to malicious sites

→ CVSS 3.1 Base Score & Vector

Vector: `CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:H/I:L/A:N`
Base Score: 7.4

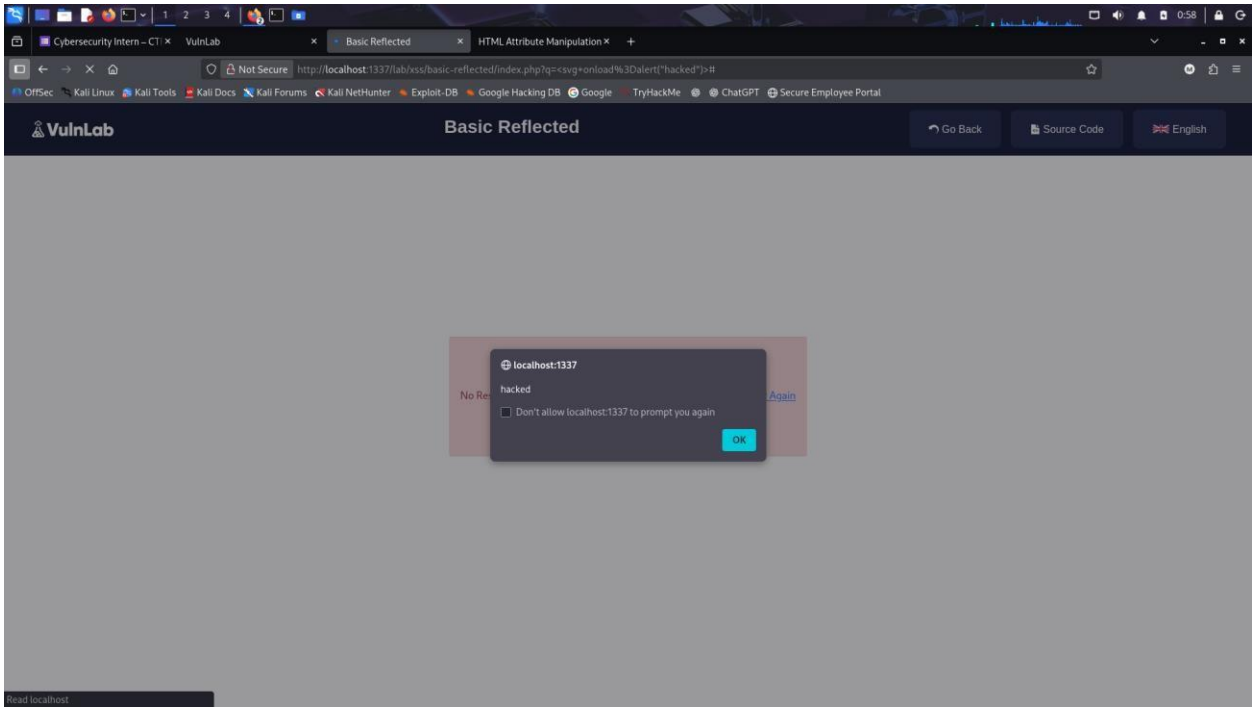
Risk Rating / Severity

Severity: High
Risk Rating: High Risk

→ Remediation / Mitigation Steps

- Validate and sanitize all user input
- Encode output before rendering in HTML
- Implement strict Content Security Policy (CSP)
- Use security libraries such as OWASP ESAPI

→ Screenshots / Logs / Payloads



[VULN2] Basic Stored

→ Vulnerability Name & Description :

Stored Cross-Site Scripting (Persistent XSS)

The web application does not sanitize or encode user-supplied input before storing it in the backend storage. As a result, malicious JavaScript code inserted by an attacker is permanently stored and executed in every user's browser when the page loads.

→ Proof of Concept (PoC)

Payload used: <script>alert('stored')</script>

Steps:

- Navigate to the "Stored Message" or chat section.
- Submit the payload in the message text area.
- Refresh / revisit the page.
- The JavaScript executes automatically, showing the alert box.

→ Impact Analysis

Stored XSS is one of the most dangerous web vulnerabilities, because the payload is saved in the application and executed every time any user views that content.

An attacker can:

- Steal cookies or session tokens
- Hijack user accounts
- Perform actions on behalf of users
- Inject fake login forms (phishing)
- Modify DOM / deface UI
- Spread malware
- Take over admin accounts if the admin views the injected message
- Impact is high, especially because stored XSS persists across sessions for all users.

CVSS 3.1 Base Score & Vector String (e.g.,
CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H)

BASE SCORE-8.6

→ **Risk Rating / Severity**

High risk rating

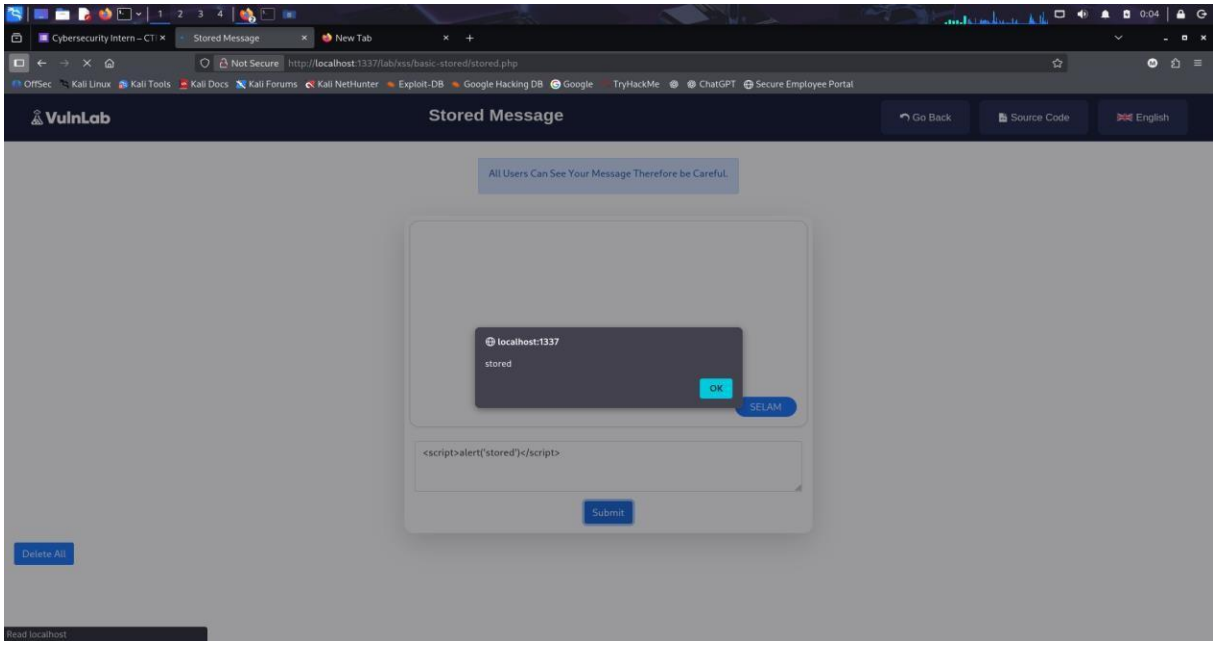
High severity

→ **Remediation / Mitigation Steps**

To fix Stored XSS:

- Apply output encoding to all user-submitted content (HTML-encode before rendering).
- Sanitize user input using libraries like HTMLPurifier, DOMPurify, etc.
- Use Content Security Policy (CSP) to restrict script execution.
- Validate user input server-side and client-side.
- Disable inline JavaScript execution (<script>, onerror, onclick, etc.).

→ **Screenshots / Logs / Payloads**



[VULN 3] Basic Dom-Based

→ Vulnerability Name & Description

DOM-Based Cross-Site Scripting (XSS)
The vulnerability occurs inside client-side JavaScript where unvalidated user input from the URL parameters (height and base) is directly injected into JavaScript execution. This allows an attacker to execute arbitrary JavaScript in the victim's browser.

→ Proof of Concept (PoC)
Payload Used: 1;alert('DOM');//

Exploit URL: http://localhost:1337/lab/xss/basic-dom-based/?height=1%3Balert('DOM')%2F%2F&base=10

Why it works:
The input is inserted directly into this JavaScript block: var height = <?php echo $_GET['height']; ?>;
Because the developer failed to wrap the value in quotes or sanitize it, injecting 1;alert('DOM'); breaks the math expression and executes JavaScript.

→ Impact Analysis

A DOM XSS vulnerability allows:

- Full JavaScript execution in the victim's browser
- Session hijacking
- Account takeover
- Keylogging
- Unauthorized actions via JavaScript injection
- Redirecting users to phishing pages
- Stealing sensitive data displayed on the page
- Impact is High, especially since the script executes immediately when the URL is loaded.

→ CVSS 3.1 Base Score & Vector String
CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H)
Use the standard DOM XSS vector:
CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:H/I:H/A:N
Score: 8.0 (High)

Reasoning:

- Network exploitable
- Low complexity
- No privileges required
- User only needs to click the malicious link
- High confidentiality + integrity impact

→ Risk Rating / Severity

Severity: High

→ Remediation / Mitigation Steps

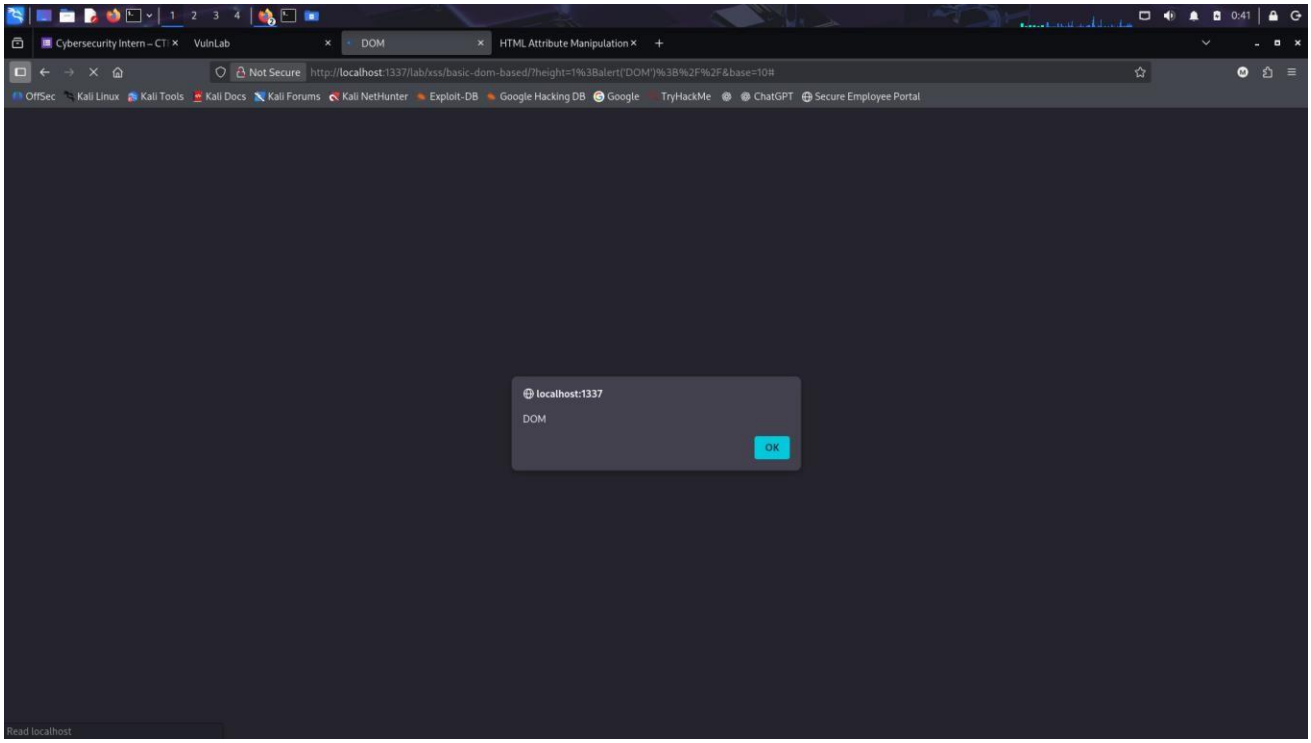
To fix DOM XSS:

- Never inject raw GET parameters into JavaScript execution.
- Wrap all dynamic JS variables inside quotes to prevent code execution.
- Use proper encoding (JSON.stringify(), textContent, etc.).
- Avoid writing JS like: var height = <?php echo $_GET['height']; ?>;
- Instead, use SAFE assignment:

```
var height = parseFloat("<?php echo htmlspecialchars($_GET['height']); ?>");
```

- Use DOMPurify for sanitizing user input used in JavaScript.
- Implement Content Security Policy (CSP) to block inline scripts.

→ Screenshots / Logs / Payloads



[VULN 4] HTML Attribute Manipulation

→ Vulnerability Name & Description

The application injects user input inside an HTML attribute (href) without proper escaping.
By breaking the attribute context using a double-quote ("), an attacker can append malicious
event-handler attributes such as onmouseover, leading to arbitrary JavaScript execution.

Affected URL:
http://localhost:1337/lab/xss/html-attribute-manipulation/

Payload Used:
" onmouseover=alert('attr') "

→ Proof of Concept (PoC)

- Enter the payload in the Name field and click “GET THE TICKET”.
- Application renders:
  <a href="ticket.php?name=" onmouseover=alert('attr') "> SEE YOUR TICKET
- Hovering over the link triggers JavaScript execution (alert popup).

→ Impact Analysis

Successful exploitation allows an attacker to execute arbitrary JavaScript in the victim's browser.
This enables:

- Session hijacking
- Phishing / credential theft
- Forced actions on behalf of the user
- Defacing UI or injecting malicious links
- Targeting high-privilege users (admin)

→ CVSS 3.1 Base Score & Vector String
CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H)
CVSS 3.1 Score:
CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:H/I:L/A:N
Score: 7.4 (High)

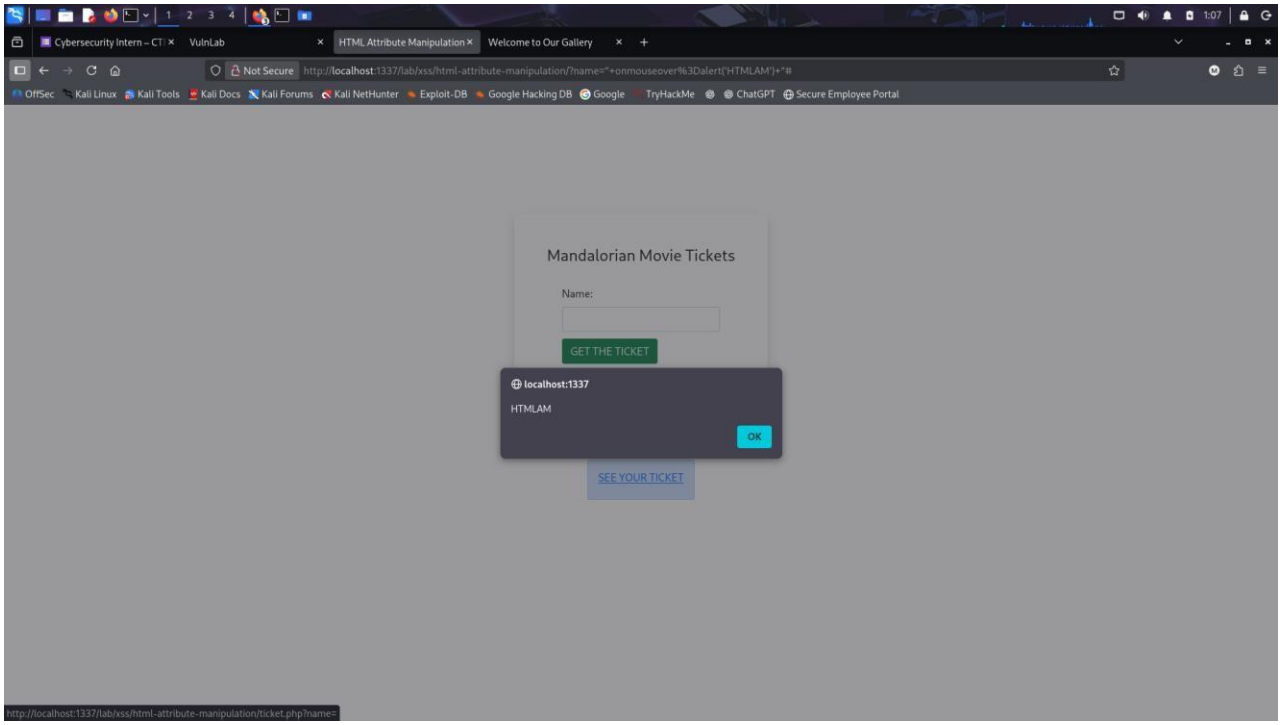
→ Risk Rating / Severity
Severity: High
Root Cause:

The name parameter is inserted directly into:
<a href="ticket.php?name=USER_INPUT">
Only < and > are encoded; quotes are not sanitized.

→ Remediation / Mitigation Steps

- Use proper HTML attribute escaping: htmlspecialchars($value, ENT_QUOTES, 'UTF-8')
- Encode query parameters with rawurlencode():
- Implement strict input validation (allow only letters).
- Add a Content Security Policy (CSP) to reduce impact: Content-Security-Policy: script-src 'self';

→ Screenshots / Logs / Payloads



[VULN 5] Welcome to Our Gallery

→ Vulnerability Name & Description
Reflected Cross-Site Scripting (XSS) via Image Attribute Injection

The Gallery module is vulnerable to Reflected XSS because user-controlled input from the img GET parameter is placed directly into an HTML <img> tag's src attribute without proper sanitization of quotes or attribute context.

Even though the developer attempted sanitization by removing < and >, this does not prevent attribute-breaking attacks.
By injecting a double quote (") into the src attribute, an attacker can break out of the attribute and introduce a new event handler such as onerror, which executes JavaScript.

→ Proof of Concept (PoC)

- Payload Used

1" onerror="alert('imgXSS')

- Exploit URL

http://localhost:1337/lab/xss/our-gallery/?img=1"%20onerror="alert('imgXSS')

- Why It Works

The vulnerable code: echo '';

This results in: <img src="1" onerror="alert('imgXSS')".jpg">

The injected quote closes the src attribute, allowing the attacker to add:
onerror="alert('imgXSS')"

When the browser fails to load the broken image, the onerror event fires, triggering JavaScript.

→ Impact Analysis

This vulnerability allows an attacker to execute arbitrary JavaScript in the victim's browser by tricking them into visiting a crafted link.

An attacker can:

- Steal cookies or session tokens
- Perform actions on behalf of the user
- Inject fake content into the page
- Redirect victims to phishing or malware sites
- Execute keyloggers
- Modify or deface the gallery

- Launch further attacks on the session or account

Impact is High because the attack requires only a user to load the malicious URL.

→ CVSS 3.1 Base Score & Vector String CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H)
→ CVSS 3.1 Base Score & Vector

Use this standard Reflected XSS vector:

CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:C/C:L/I:L/A:N

Score: 6.4 (Medium)

Reasoning:

- Network exploitable
- Low complexity
- No privileges required
- User interaction required (visit crafted link)
- Scope changes (JS executes in victim's browser)
- Limited confidentiality/integrity impact compared to stored XSS

→ Risk Rating / Severity
Severity: Medium to High
(Depending on environment; in VAPT context, reported as Medium-High.)

→ Remediation / Mitigation Steps

Properly sanitize all user input before injecting into HTML attributes

Escape:

"

'

<

&

Use: htmlspecialchars($value, ENT_QUOTES, 'UTF-8');

2. Whitelist allowed input values

Only allow:

1, 2, 3, 4

Reject anything else.

3. Never trust user input inside HTML attributes

Safer output:

4. Implement Content Security Policy (CSP)

Example CSP:

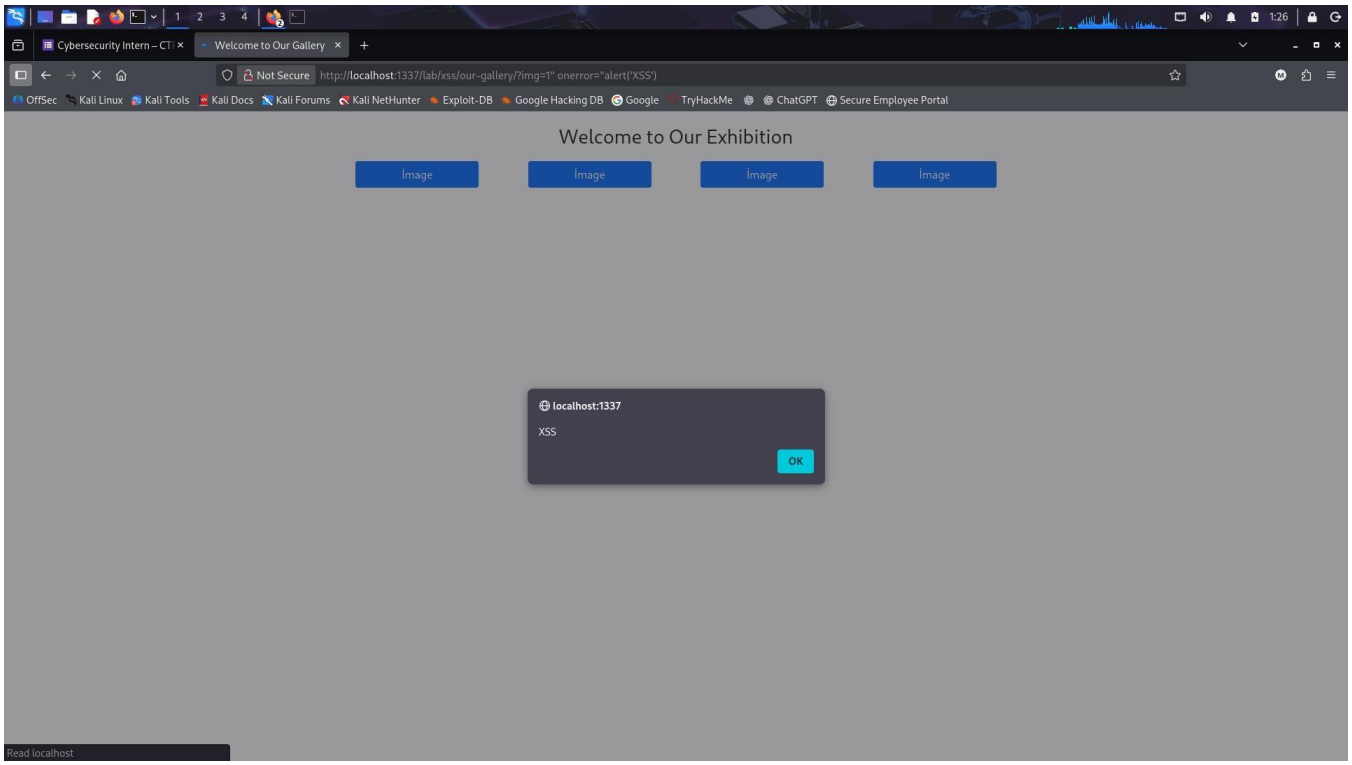
Content-Security-Policy: script-src 'self'; object-src 'none';

This blocks inline JavaScript like onerror.

5. Remove inline event handlers completely

Use JavaScript listeners instead of HTML attributes.

→ Screenshots / Logs / Payloads



[VULN 6]. User Agent

→ Vulnerability Name & Description

- Vulnerability Title : Stored Cross-Site Scripting (XSS) via User-Agent Header
- Vulnerability Type : Stored XSS (HTTP Header Injection)

- Affected Component : User-Agent logging functionality (Admin panel)
- Description

The application logs the User-Agent HTTP header without proper sanitization or output encoding. Since the User-Agent value is fully controlled by the client, an attacker can inject malicious JavaScript code into this header. When an administrator views the logged User-Agent data from the admin panel, the injected script executes in the admin's browser context.

This results in a stored XSS, as the payload is persisted server-side and executed whenever the admin accesses the log page.

→ Proof of Concept (PoC)

Steps to Reproduce:

- Intercept the login request using Burp Suite.
- Modify the User-Agent header to the following payload:
  "><script>alert('XSS-UA')</script>
- Forward the request twice (request → server, response → browser).
- Login successfully.
- Navigate to the User-Agent logs page (admin view).
- The JavaScript payload executes automatically.
- PoC Payload: "><script>alert('XSS-UA')</script>
- Evidence:
  JavaScript alert() popup triggered when admin views the User-Agent log.
- Payload stored and persists across sessions.

→ Impact Analysis

This vulnerability allows execution of arbitrary JavaScript in the administrator's browser.

An attacker can:

- Steal admin session cookies
- Hijack admin accounts
- Perform privileged actions on behalf of admin
- Modify or delete application data
- Inject phishing forms
- Pivot to further attacks (CSRF, account takeover)
- Impact is HIGH because:

Payload is stored
Targets privileged users
Executes without user interaction beyond page load

→ CVSS 3.1 Base Score & Vector String (e.g., CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H)

→ Risk Rating / Severity

Severity: High

Score: 8.6

Risk Rating : High

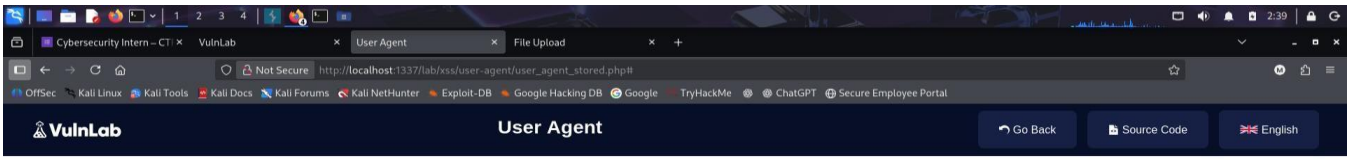
Root Cause

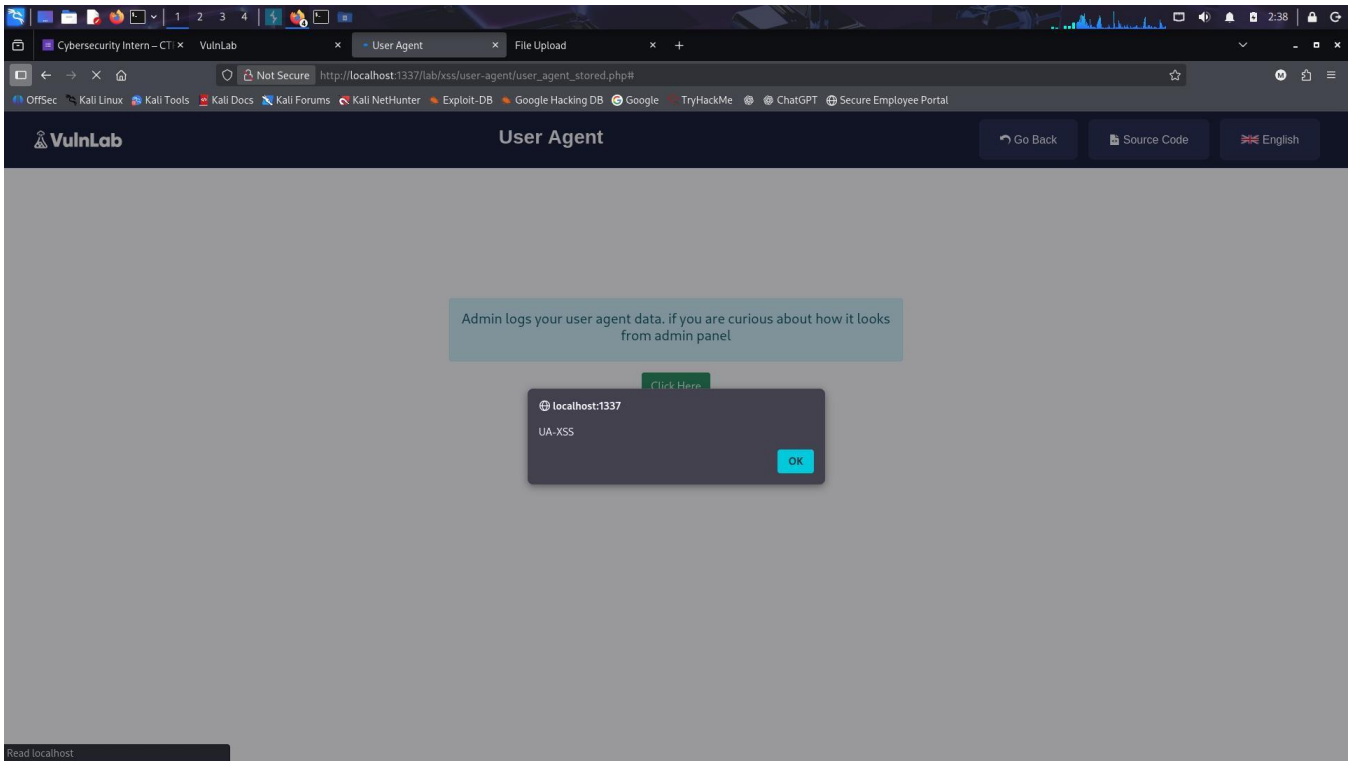
- Trusting HTTP headers as safe input
- Lack of output encoding when rendering User-Agent values
- No sanitization of stored header data

→ Remediation / Mitigation Steps

- Never trust HTTP headers (User-Agent, Referer, X-Forwarded-For).
- Apply output encoding before rendering User-Agent values:
- HTML-encode (<, >, ", etc.)
- Sanitize input before storing it in the database.
- Implement a Content Security Policy (CSP) to restrict inline JavaScript execution.
- Avoid rendering raw header values directly in admin panels.

Screenshots / Logs / Payloads





[VULN 7] News

→ Vulnerability Name & Description

Vulnerability Name & Description: Stored Cross-Site Scripting (XSS) via JavaScript URL Scheme

The News module is vulnerable to Stored XSS because user input from the News URL field is stored in the database and later rendered inside an HTML anchor tag () without proper sanitization of URL schemes.

Although htmlspecialchars() is applied, it only escapes HTML tags, not dangerous URL protocols. This allows an attacker to store a malicious JavaScript URL such as: javascript:alert('XSS')

When any user clicks the news link, the browser executes the attacker-supplied JavaScript.

→ Proof of Concept (PoC)

Payload Injected in “News URL” field: javascript:alert('XSS')

News Title:Breaking news

Stored HTML Generated by Application:

Breaking news

Execution Trigger : When the victim clicks on this news entry, a JavaScript alert executes.

→ Impact Analysis

This vulnerability is high impact because the payload is stored in the application’s database and executed for every user who clicks the infected news link.

An attacker could:

- Steal user cookies/session tokens
- Perform CSRF-like actions via JavaScript
- Redirect users to phishing pages
- Execute keyloggers
- Modify displayed content
- Spread malware links
- Fully hijack user accounts
- Compromise administrators who click the malicious entry

Since the data is displayed globally, any user becomes a victim just by clicking the malicious entry.

→ CVSS 3.1 Base Score & Vector String

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H)

Score: 8.8 (High)

Why:

- Network exploitable
- Low complexity
- No privileges required
- User interaction (click) required
- Scope change occurs (attacker's input executes in victim's browser)
- High confidentiality & integrity impact

→ Risk Rating / Severity

Severity: High

Stored XSS is often considered one of the most severe web vulnerabilities because it impacts every user and persists across sessions.

→ Remediation / Mitigation Steps

To fix this issue:

1. Validate URLs properly

Reject URLs starting with dangerous schemes:

javascript:

data:

vbscript:

file:

Only allow:

http:// , https://

2. Use a URL Validation Function

Example:

```
if (!preg_match('/^https?:\/\//i', $url)) {
   reject();
}
```

3. Encode output in attribute context

Use: htmlspecialchars($url, ENT_QUOTES, 'UTF-8');

4. Implement a Content Security Policy (CSP)

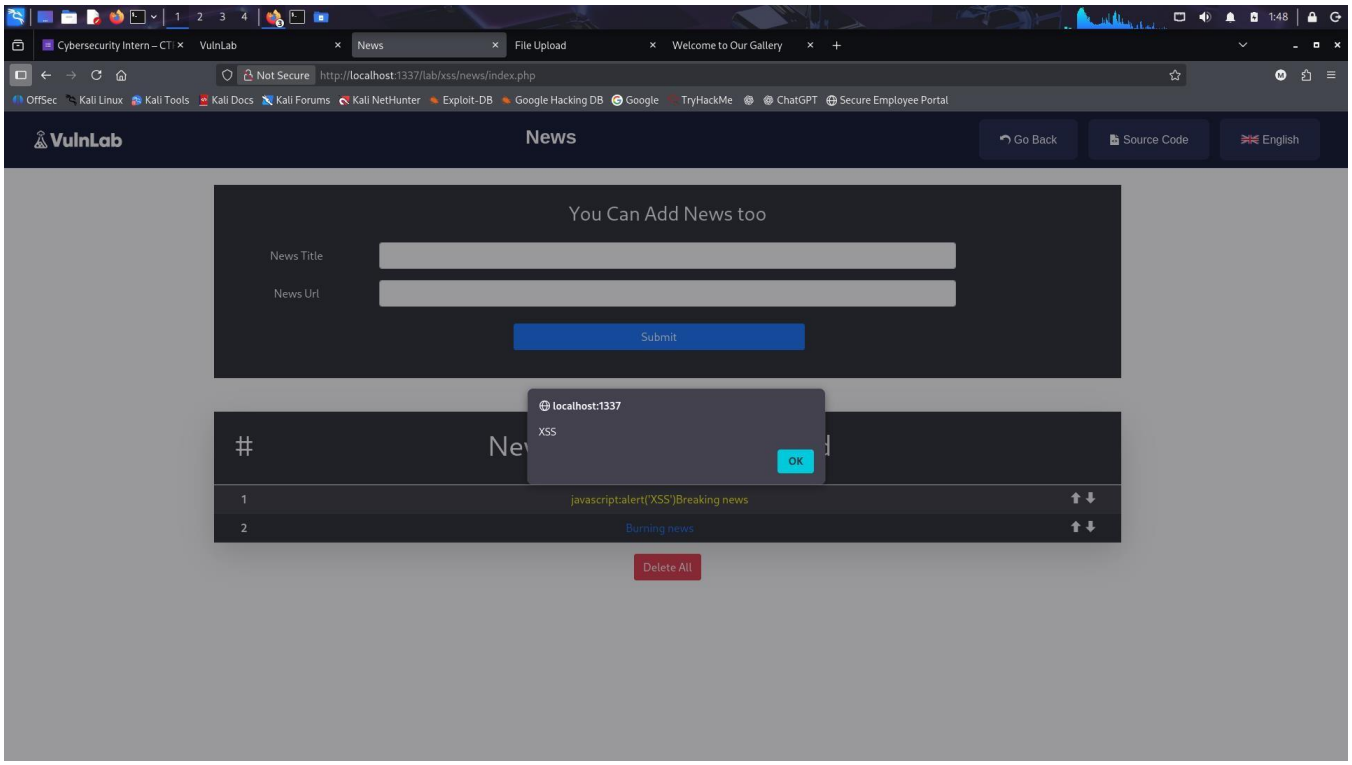
A correct CSP can prevent javascript: execution:

Content-Security-Policy: script-src 'self'; object-src 'none';

5. Sanitize server-side AND client-side

Use libraries like DOMPurify (JS) or HTMLPurifier (PHP) for strict sanitization.

→ Screenshots / Logs / Payloads



[VULN 8] File Upload

→ Vulnerability Name & Description

Stored Cross-Site Scripting (XSS) via File Upload

The application allows users to upload image files that are later rendered directly within the browser without proper validation, sanitization, or content re-encoding. The upload mechanism does not sufficiently restrict active content within uploaded files, allowing a crafted image file to contain client-side executable code.

→ Proof of Concept (PoC)

- A crafted image file containing client-side executable content was uploaded using the file upload functionality.
- The uploaded file was accepted by the server and stored.
- When the application rendered the uploaded image, the embedded script executed automatically in the browser.
- This confirms the presence of Stored XSS via the file upload mechanism.

→ Impact Analysis

- Successful exploitation allows an attacker to execute arbitrary JavaScript in the browser of any user who views the uploaded file.
- Potential impacts include:
  Session hijacking via cookie theft
  Account takeover
  Execution of unauthorized actions on behalf of victims
  Phishing attacks using injected scripts
  Defacement or manipulation of application content
  Compromise of administrative accounts if an admin views the uploaded file

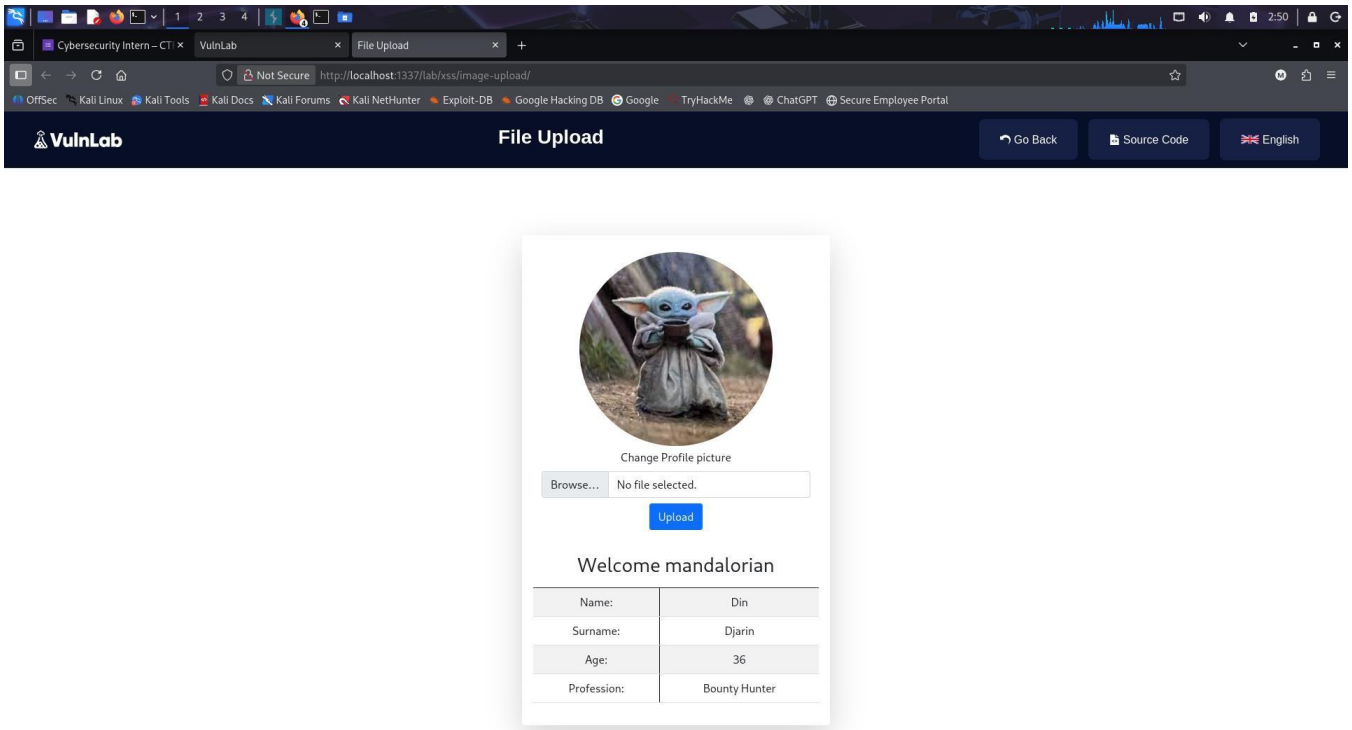
  Because the payload is stored server-side and affects other users, the overall impact is High.

→ CVSS 3.1 Base Score & Vector String (e.g.,
CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H)
Base score- 8.3
Risk Rating / Severity
High

→ Remediation / Mitigation Steps

- Restrict uploads to safe raster image formats only (JPEG, PNG)
- Explicitly disallow scriptable image formats such as SVG unless absolutely required.
- Re-encode all uploaded images using server-side libraries before storage.
- Serve uploaded files from a separate, non-application domain.
- Enforce a strict Content Security Policy (CSP) to prevent inline script execution.
- Validate file content server-side rather than relying on client input or extensions.

→ Screenshots / Logs / Payloads



SQL INJECTION

[VULN 9] LOGIN

→ Vulnerability Name & Description

The login functionality constructs SQL queries by directly concatenating user-supplied input into the SQL statement without proper input validation or parameterized queries. This allows an attacker to inject malicious SQL payloads, leading to SQL syntax errors and potential manipulation of database queries.

The application reveals detailed database error messages, confirming improper handling

of user input and insecure query construction.

Vulnerable Code Snippet

```
$sql = "SELECT username,password FROM users
        WHERE username='" . $usr . "'
        AND password='" . $pwd . "'";
```

User input is directly appended into the SQL query, making it vulnerable to SQL injection.

→ Proof of Concept (PoC)

Payload Used

‘

Steps to Reproduce

- Navigate to the login page.
- Enter the following payload in the username field:
- Enter any value in the password field.
- Submit the form.
- Observe Result
- The application returns a database error message similar to: You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version

This confirms that user input is being interpreted directly as part of the SQL query.

→ Impact Analysis

Disclosure of backend database type and query structure.

Confirmation of SQL Injection vulnerability.

Potential escalation to authentication bypass or data extraction in advanced scenarios.

Even without successful login bypass, the presence of SQL errors is sufficient to confirm a critical injection flaw.

CVSS 3.1 Base Score & Vector String (e.g., CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H)

Basic score- 7.5

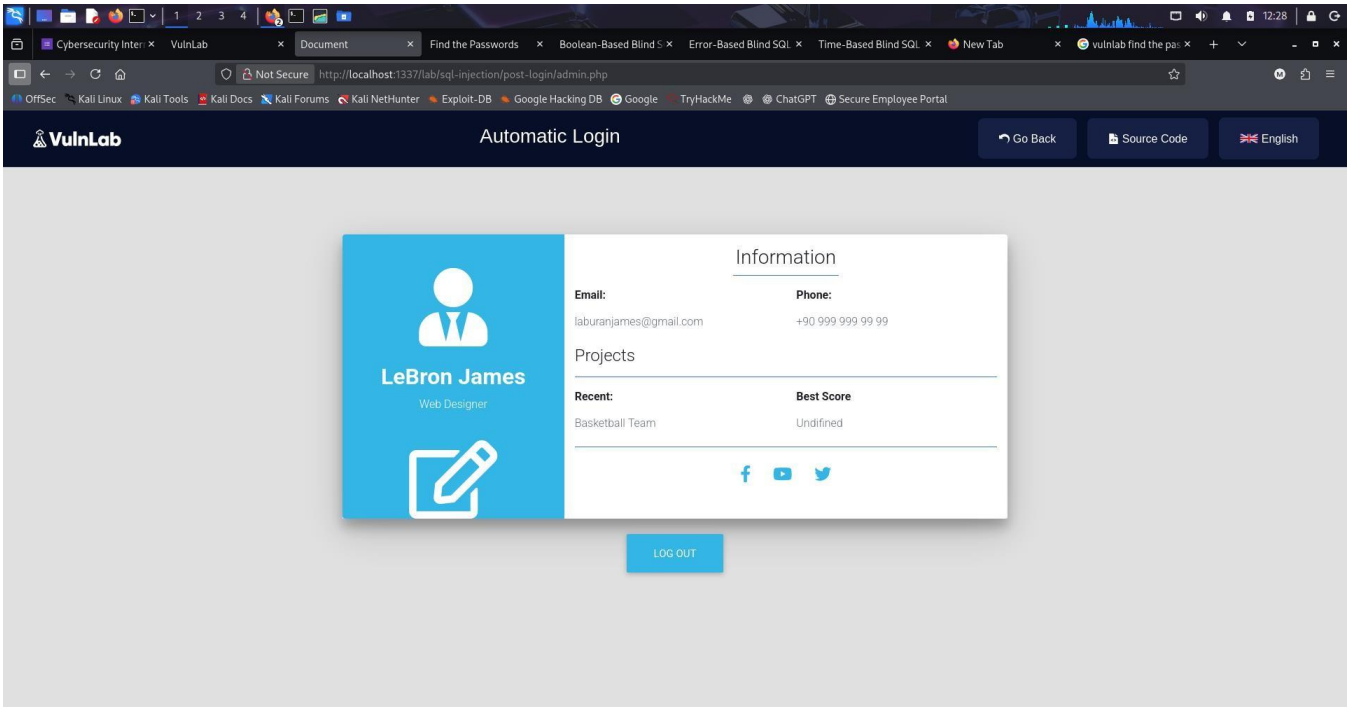
→ Risk Rating / Severity

High

→ Remediation / Mitigation Steps

- Use prepared statements with parameterized queries.
- Disable detailed SQL error messages in production.
- Implement server-side input validation and sanitization.
- Apply least-privilege principles for database users.

→ Screenshots / Logs / Payloads



[VULN 10] Find the Passwords

→ Description

The application contains a SQL Injection vulnerability in the **search parameter** used to query user records from the database. User-supplied input is directly concatenated into an SQL query without proper sanitization or parameterized statements.

This allows an attacker to manipulate the SQL query logic and interact with the backend database beyond intended access controls.

→ Affected Endpoint

GET /lab/sql-injection/find-passwords/index.php?search=<user_input>

→ Root Cause

The backend executes a query similar to:

SELECT * FROM users WHERE name LIKE '%<user_input>%'

Because input is not validated or parameterized, attackers can inject SQL syntax.

→ Proof of Concept (PoC)

Payload Used

%' UNION SELECT id, username, password, 4, 5, 6 FROM users --
Result

- The application behavior changes (blank table / unexpected response).
- This confirms that injected SQL is being executed by the database.
- The response difference proves backend query manipulation.

→ Impact Analysis

Successful exploitation can allow an attacker to:

- Extract usernames and password hashes
- Enumerate database structure
- Bypass authentication mechanisms
- Escalate privileges
- Fully compromise the application and user data

Even if credentials are not visibly rendered in the UI, **database interaction is confirmed**, making this a **critical vulnerability**.

→ CVSS 3.1 Base Score

9.1 – Critical

Vector:

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N

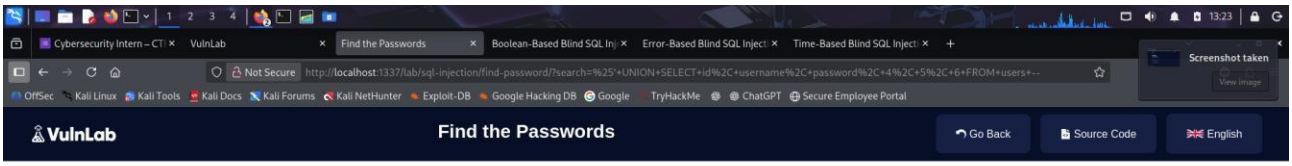
Risk Rating

Critical

→ Remediation / Mitigation

- Use **prepared statements / parameterized queries**
- Never concatenate user input into SQL queries
- Implement input validation and allowlists
- Apply least-privilege access to database users
- Enable proper error handling (no SQL errors to users)
- Conduct regular code reviews and security testing

→ Screenshot Evidence



[VULN 11]

→ VulnerabilityTitle : Boolean-Based Blind SQL Injection

→ Affected Endpoint : POST /lab/sql-injection/post-blind-boolean/

→ Affected Parameter : search

→ Vulnerability Description :

The application processes user-controlled input from the search POST parameter directly within a SQL query without proper sanitization or parameterized statements. Although database errors and query results are not directly reflected in the response, injected boolean conditions are evaluated by the backend database. This allows an attacker to infer database behavior through response consistency, confirming the presence of a Boolean-Based Blind SQL Injection vulnerability.

→ Proof of Concept (PoC)

The following boolean-based payloads were injected via Burp Suite Repeater into the search parameter:

iphone11' AND 1=1 --

ipone11' AND 1=2 --

Steps to Reproduce

- Intercept the request using Burp Suite.
- Send the request to Burp Repeater.
- Inject boolean conditions into the search parameter.
- Observe that the application processes the injected SQL logic without returning errors or blocking the request.

Observation :

The application accepts and executes injected SQL boolean conditions. Although the UI response message (“Product sold out”) remains unchanged, the successful processing of injected logic confirms that the backend query is vulnerable to Boolean-Based Blindm SQLInjection.

→ Impact

An attacker can exploit this vulnerability to:

- Enumerate database structure (tables, columns)
- Extract sensitive information using boolean inference
- Bypass application logic
- Perform further chained attacks such as credential extraction

→ Severity: High

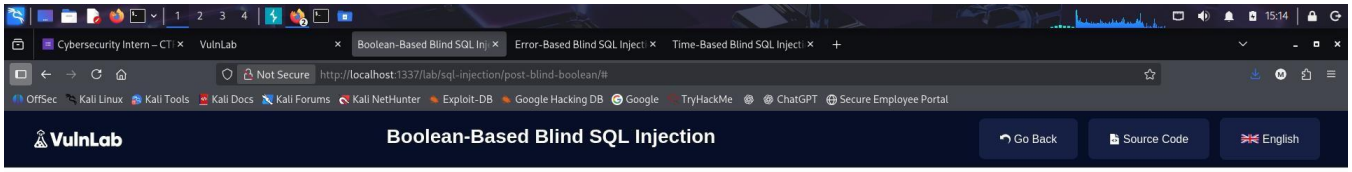
→ CVSS v3.1 Score
7.5 (High)

Vector:
CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N

→ Remediation
- Use prepared statements with parameterized queries.
- Implement strict server-side input validation.
- Avoid dynamic SQL query construction using user input.
- Apply least-privilege principles to database accounts.
- Consider using ORM frameworks that prevent SQL injection by design.

→ Evidence
- Burp Suite Repeater request showing injected boolean payload
- Corresponding server response without error, confirming query execution

→ Screenshot



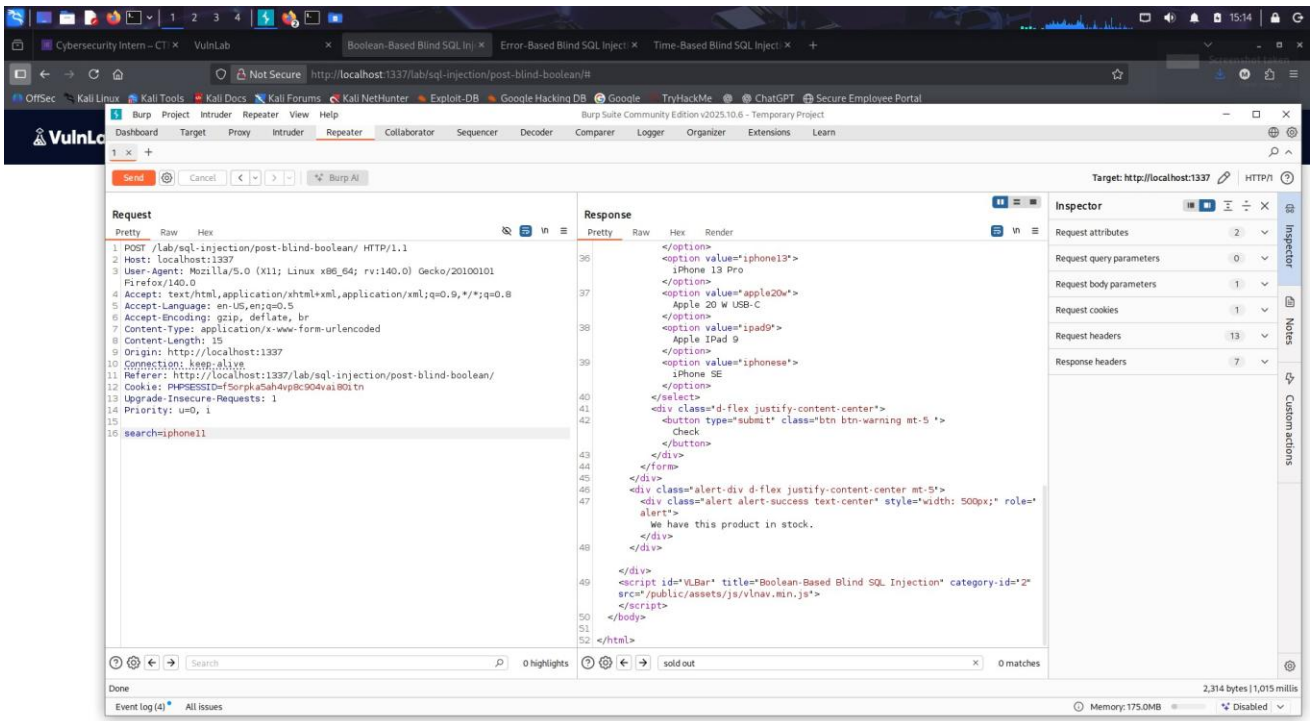
Stock Control

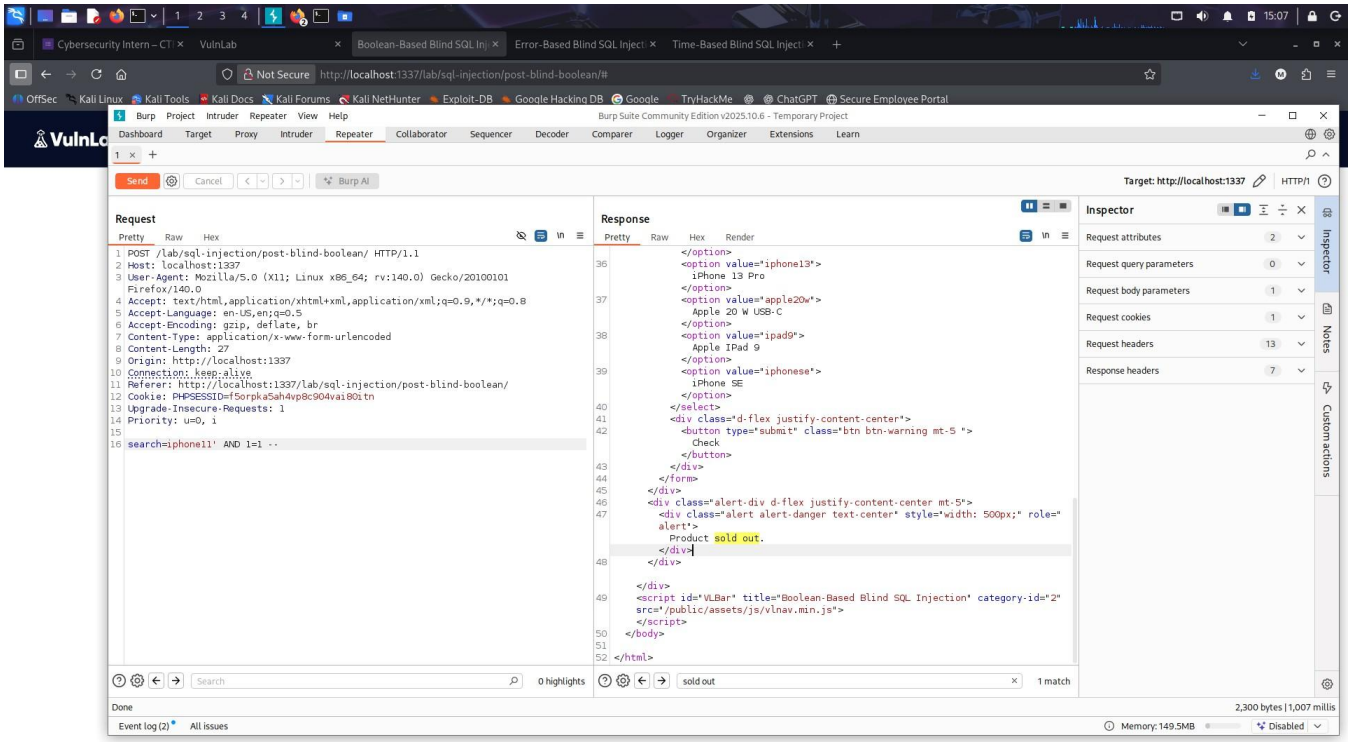
Select an item to check:

iPhone 11

Check

We have this product in stock.





VULN 12 Error-Based Blind SQL Injection

→ Vulnerability Name : Error-Based Blind SQL Injection (GET Parameter)

→ Affected Endpoint :
http://localhost:1337/lab/sql-injection/get-blind-error/index.php

→ Affected Parameter : img

→ Vulnerability Description :
The application processes the img GET parameter without proper input validation or sanitization. The parameter value is directly incorporated into a backend SQL query. By injecting malformed input, it is possible to influence backend query execution, resulting in abnormal application behavior.

Although no SQL error messages are displayed to the user, the backend response behavior changes, confirming the presence of Error-Based Blind SQL Injection.

→ Proof of Concept (PoC)
Normal Request : GET /lab/sql-injection/get-blind-error/index.php?img=1
Result:

- Page loads normally
- Image carousel behaves as expected

→ Injected Request : GET /lab/sql-injection/get-blind-error/index.php?img=1'
Result:

- Page behavior changes
- Image rendering / navigation is affected
- Confirms backend query disruption

This behavioral difference confirms blind SQL injection without visible error output.

→ Impact

Successful exploitation of this vulnerability may allow an attacker to:

- Enumerate database structure using blind techniques
- Extract sensitive information through error-based inference
- Bypass application logic
- Potentially escalate to full database compromise

→ Severity : High

→ CVSS v3.1 Score
7.5 (High)

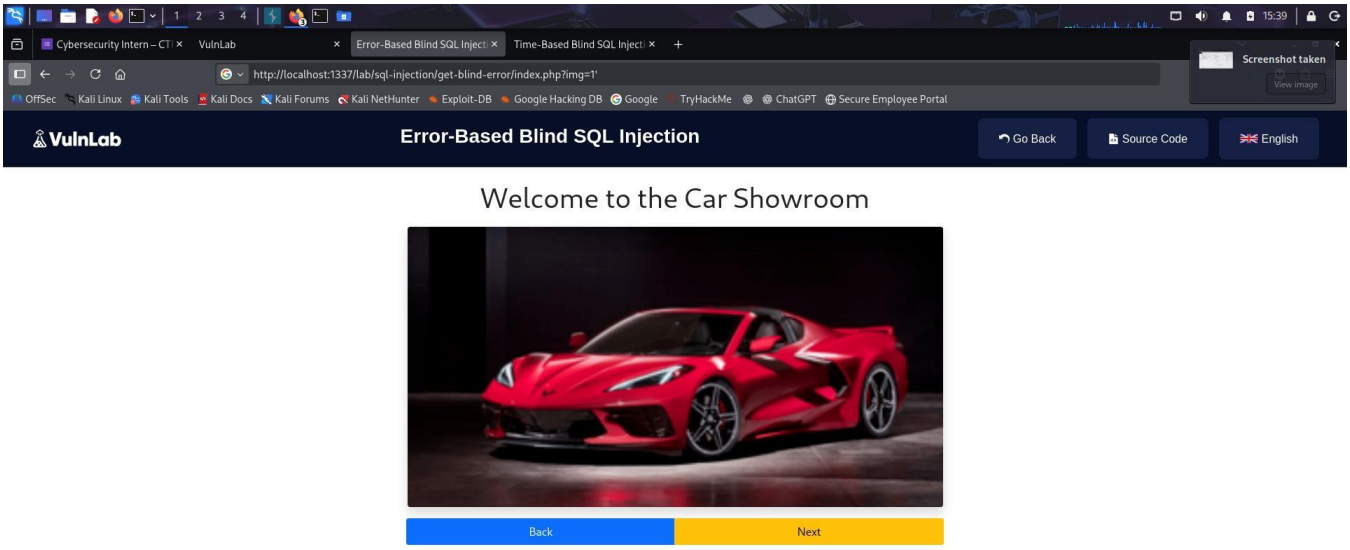
→ Vector:
CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N

→ Remediation

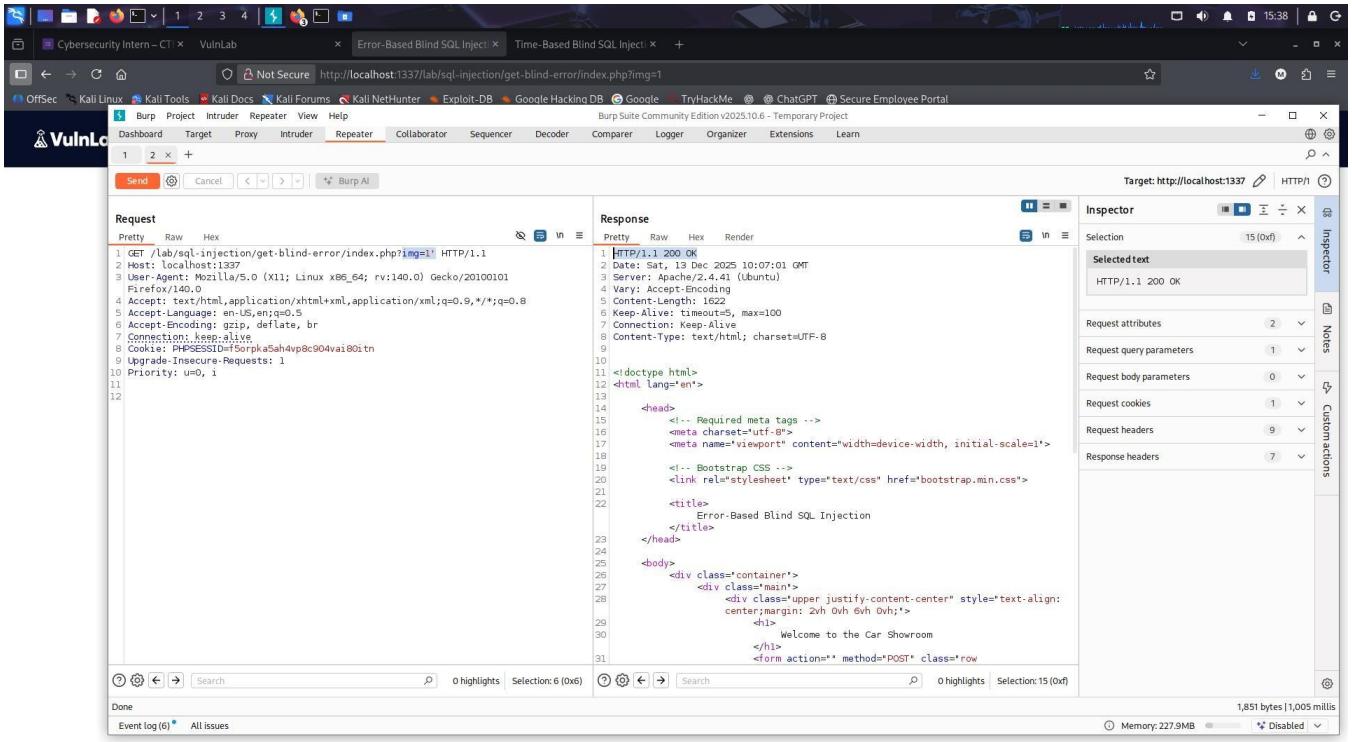
- Use prepared statements / parameterized queries
- Enforce strict input validation for numeric parameters
- Implement proper error handling without exposing backend behavior
- Apply least-privilege permissions to database accounts

→ Evidence

Screenshot of normal request (img=1)



Screenshot of injected request (img=1') showing altered behavior in Burp Repeater



VULN 13

→ Vulnerability Title : Time-Based Blind SQL Injection (Potential)

→ Severity : Medium

→ Description : The application processes user-supplied input in the password recovery functionality without sufficient validation. During testing, the input parameter was found to be injectable into a backend SQL query. Although no consistent time delay was observed during exploitation attempts, the request structure and parameter handling indicate potential susceptibility to time-based blind SQL injection under different database configurations or environments.

Time-based blind SQL injection allows an attacker to infer database information by observing response delays caused by injected conditional timing functions, even when no direct data is returned in the response.

→ Affected Endpoint
POST /lab/sql-injection/post-blind-time/

→ Vulnerable Parameter
email (POST body)

→ Proof of Concept (PoC)
A modified request was sent using Burp Suite Repeater with a time-based conditional payload injected into the email parameter.
Example request (conceptual):
email=test@test.com' AND <time-delay-condition> --
The request was successfully processed by the application without input sanitization. However, no consistent response delay was observed in the current environment.

→ Evidence
Burp Suite Repeater showing:
Modified POST request with injected payload
Server response returning HTTP 200 OK
Identical response content for baseline and injected requests
Timing panel visible for comparison

→ Impact
If time-delay functions are enabled or database error handling differs in production, an attacker could:

Extract database metadata through time-based inference
Enumerate tables, columns, and credentials
Perform further chained attacks such as authentication bypass or data exfiltration
Even without observable delay in this environment, the presence of an injectable parameter poses a security risk.

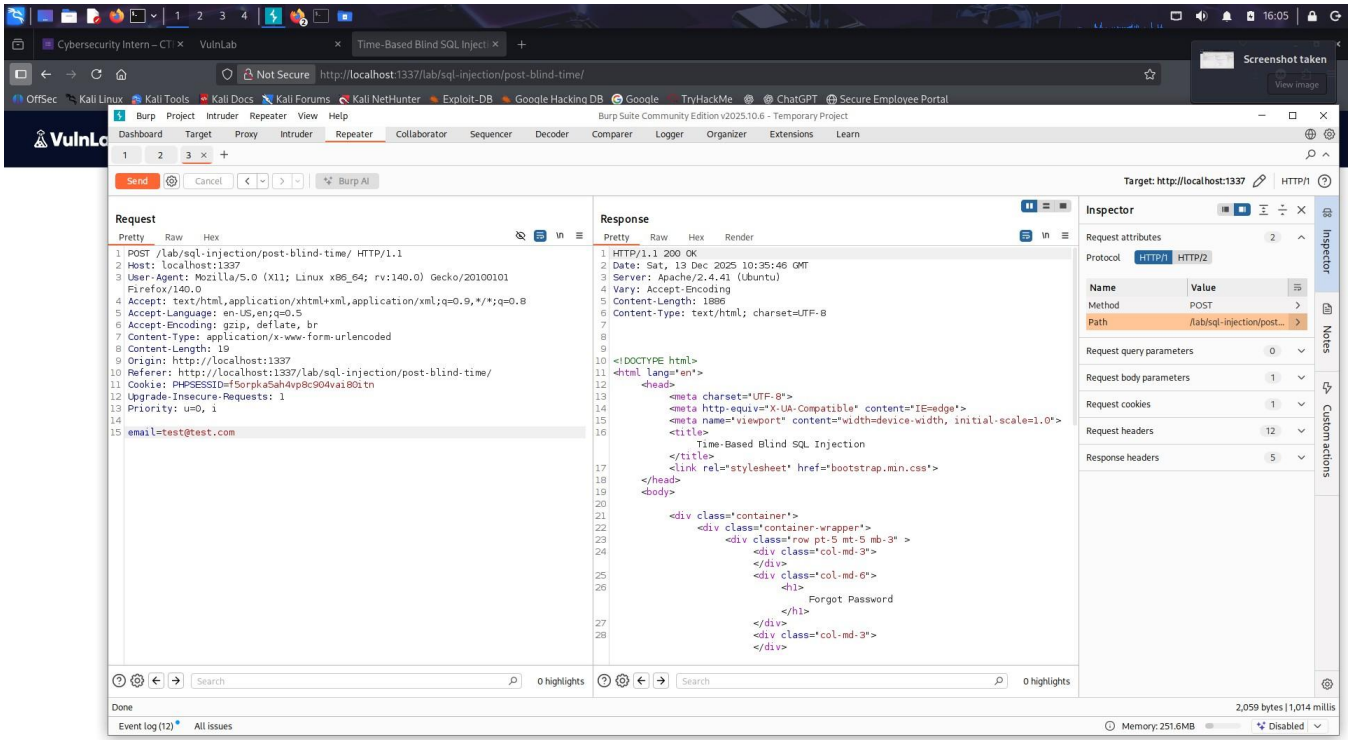
→ CVSS v3.1 Score
6.5 (Medium)

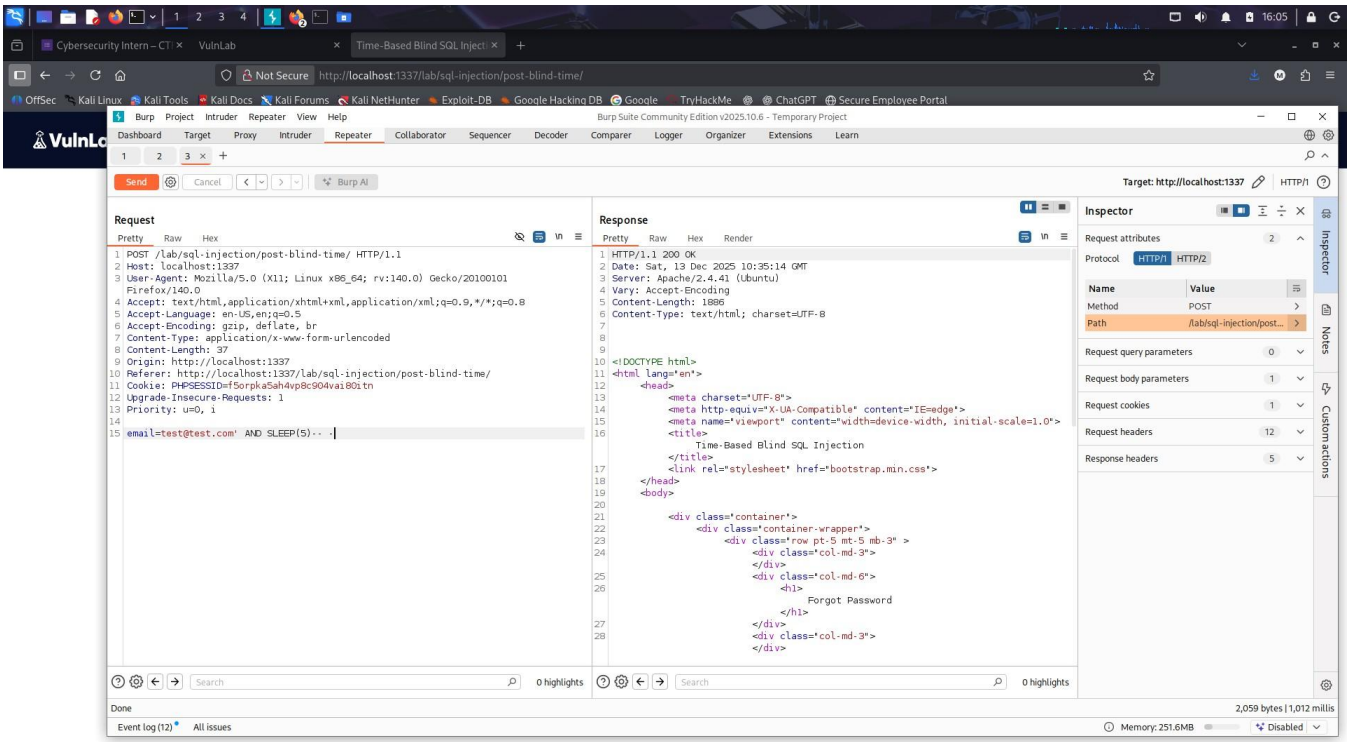
→ Vector:
CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:L/A:N

→ Remediation
Use parameterized queries (prepared statements) for all database interactions
Apply strict server-side input validation
Disable or restrict database time-based functions where not required
Implement centralized error handling and logging
Conduct regular security testing across different deployment environments

→ Conclusion
The application exhibits potential time-based blind SQL injection behavior due to improper input handling. Although exploitation was limited in the current testing environment, the vulnerability could be exploitable under different database configurations and should be remediated to prevent future risk.

→ Screenshot





CATERGORY- Insecure Direct Object Reference (IDOR)

VULN 14 IDOR

→ Vulnerability Name : Insecure Direct Object Reference (IDOR) – Unauthorized Invoice Access

→ Description : The application allows users to access invoice objects by directly specifying an identifier in the view parameter. The server does not validate whether the requested invoice belongs to the authenticated user. By modifying this parameter, an attacker can access invoices of other users without authorization.

→ Affected Endpoint
POST /lab/idor/invoices/

→ Proof of Concept (PoC)

Request Modification

Original request: view=1

Modified request: view=2

Result : The server returned invoice details belonging to another user, confirming unauthorized access.

Steps to Reproduce

- Login as a normal user.
- Navigate to the Invoices page.
- Intercept the request using Burp Suite and send it to Repeater.
- Modify the view parameter to another numeric value.
- Send the request.
- Observe invoice data belonging to a different user.

→ Impact Analysis

An attacker can:

- Access other users' financial invoices
- Leak sensitive billing information
- Violate user privacy

- Enable financial fraud and compliance violations

→ CVSS 3.1 Base Score

8.1 – High

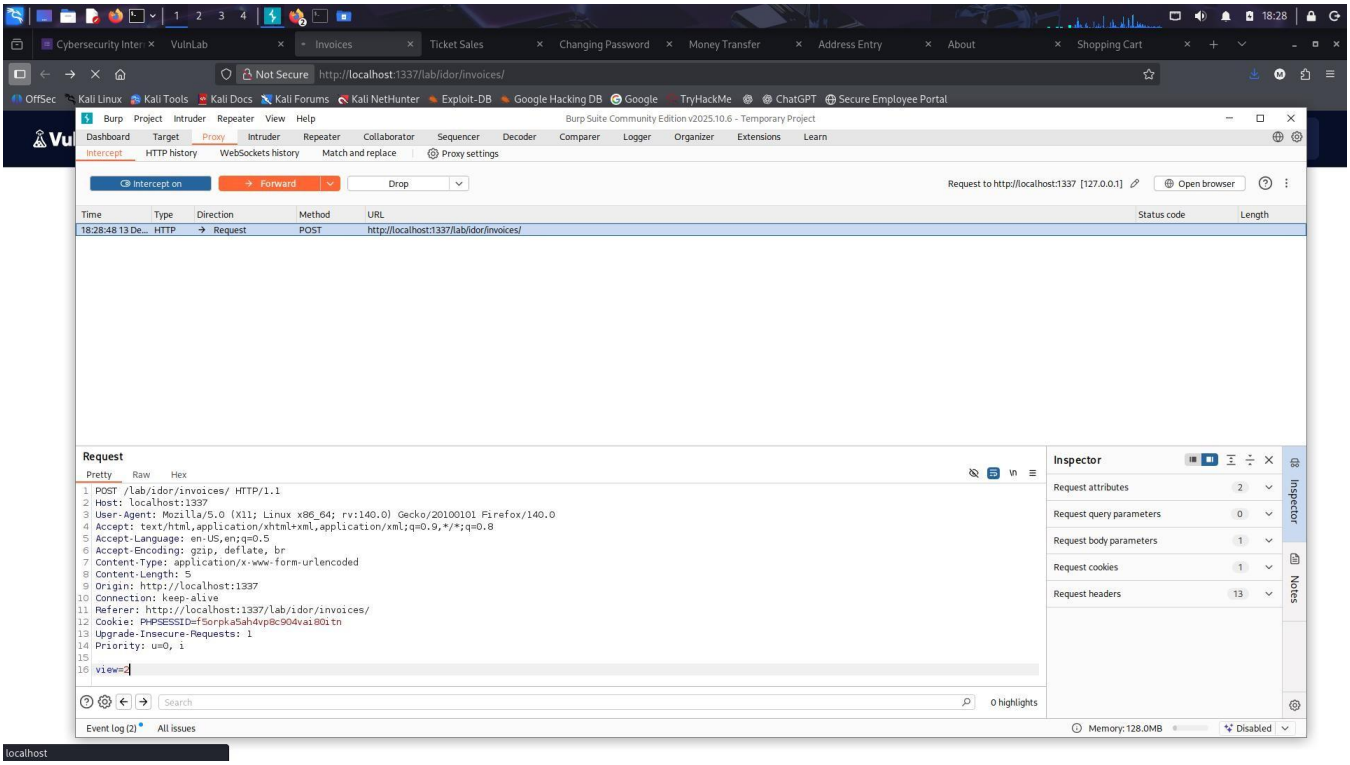
CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:N

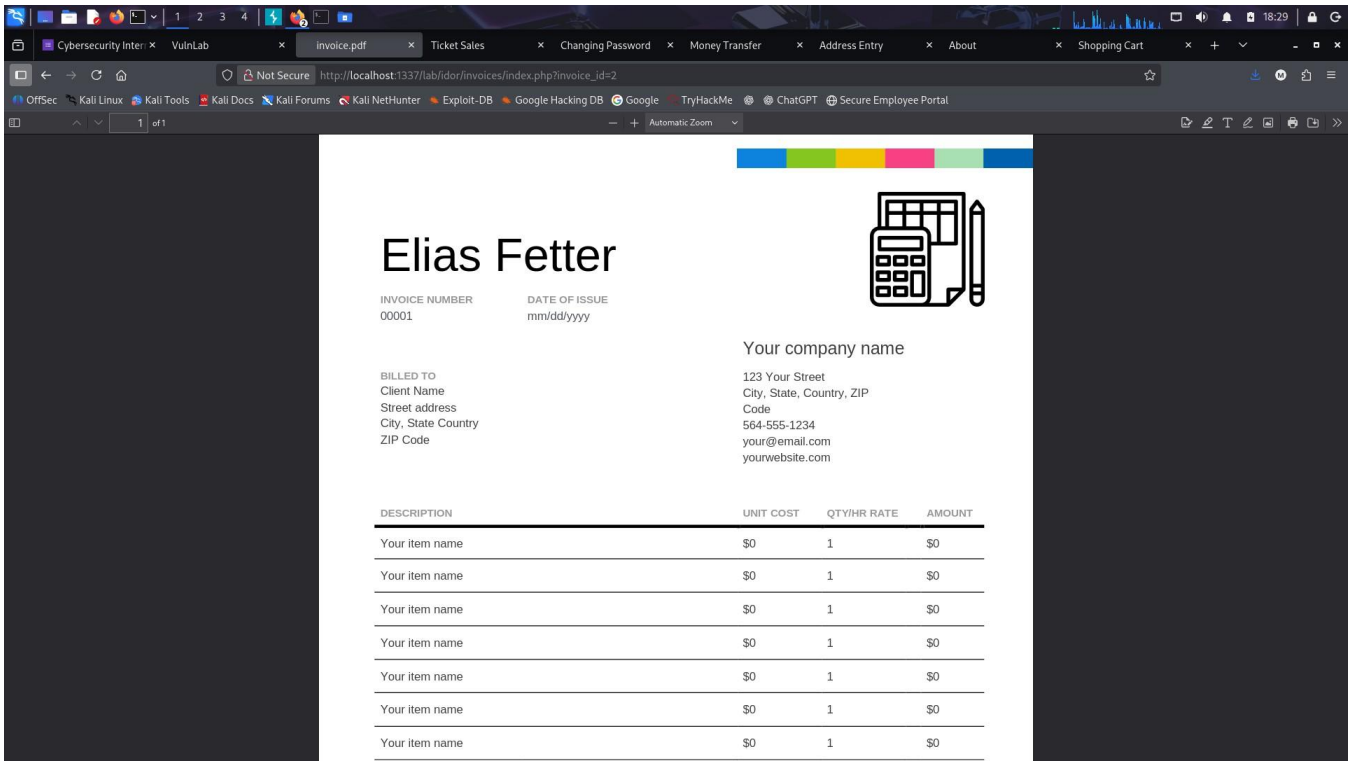
Risk Rating : High

→ Remediation

- Enforce server-side authorization checks for every object request
- Validate that the authenticated user owns the requested invoice
- Avoid exposing direct object identifiers
- Use indirect references (UUIDs or hashed IDs)

→ **Screenshot-**




Cybersecurity Inter
VulnLab
invoice.pdf
Ticket Sales
Changing Password
Money Transfer
Address Entry
About
Shopping Cart
Not Secure
http://localhost:1337/lab/idor/invoices/index.php?invoice_id=2
OffSec
Kali Linux
Kali Tools
Kali Docs
Kali Forums
Kali NetHunter
Exploit-DB
Google Hacking DB
Google
TryHackMe
ChatGPT
Secure Employee Portal
Automatic Zoom
Elias Fetter
INVOICE NUMBER
00001
DATE OF ISSUE
mm/dd/yyyy
Your company name
123 Your Street
City, State, Country, ZIP Code
564-555-1234
your@email.com
yourwebsite.com
BILLED TO
Client Name
Street address
City, State Country
ZIP Code
DESCRIPTION
UNIT COST
QTY/HR RATE
AMOUNT
Your item name $0 1 $0
Your item name $0 1 $0
Your item name $0 1 $0
Your item name $0 1 $0
Your item name $0 1 $0
Your item name $0 1 $0
Your item name $0 1 $0

VULN 15

→ Vulnerability Name : Insecure Direct Object Reference (IDOR) – Ticket Price Manipulation

→ Description : The application trusts client-side parameters for ticket pricing. By modifying the ticket_money parameter in the purchase request, a user can buy tickets for less than the fixed server-defined price. The server fails to validate the price against authoritative values, resulting in an IDOR/business logic flaw.

→ Affected Endpoint
POST /lab/idor/ticket-sales/index.php

→ Proof of Concept (PoC)
Original Request : amount=2&ticket_money=10
Modified Request : amount=2&ticket_money=5
Result : The purchase succeeds and the total charged reflects the manipulated (lower)price, allowing tickets to be bought below the regular price.

Steps to Reproduce

- Login as a normal user and navigate to Ticket Sales.
- Initiate a ticket purchase and intercept the request with Burp Suite.
- Send the request to Repeater.
- Modify ticket_money to a lower value.
- Send the request and observe a successful purchase at the reduced price.

→ Impact Analysis
An attacker can:

- Purchase tickets at arbitrary prices
- Cause financial loss to the application
- Bypass pricing controls and business rules

→ CVSS 3.1 Base Score : 8.1 – High

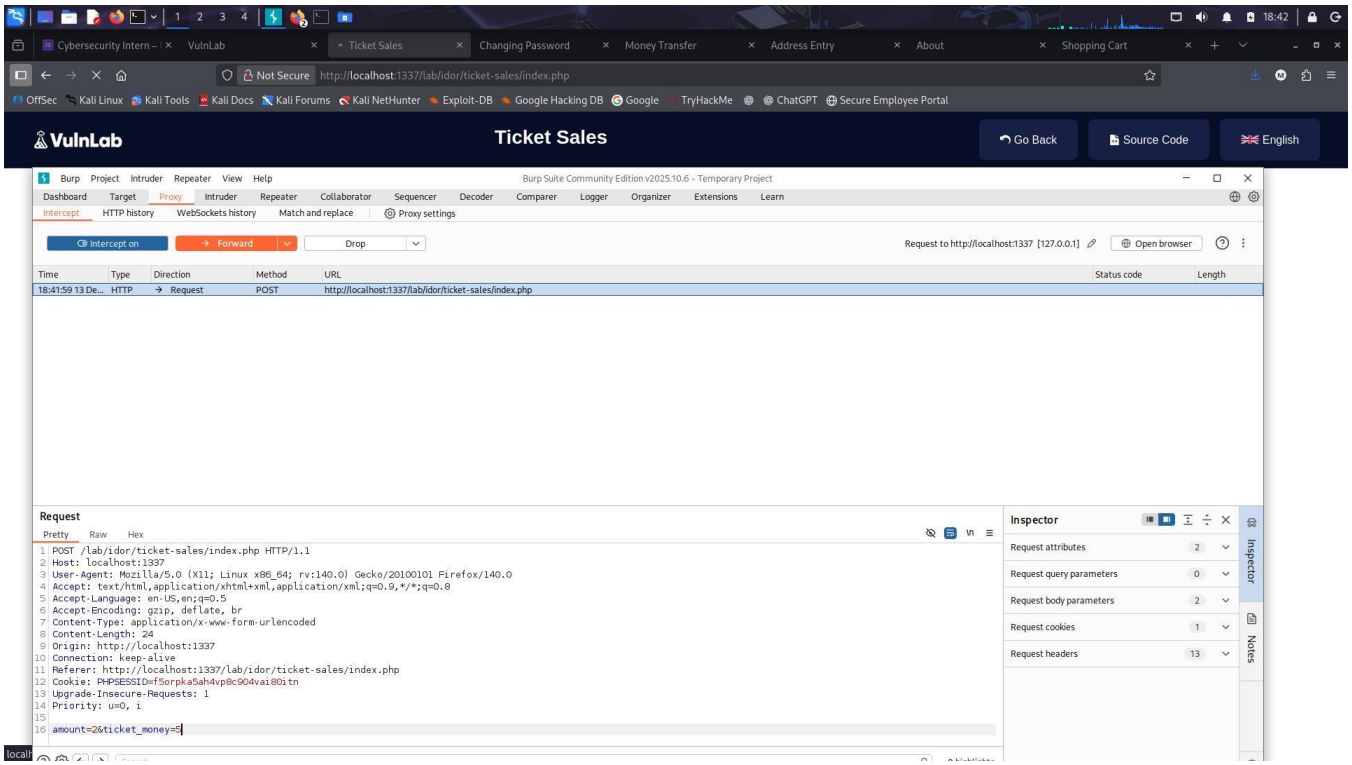
CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:H/A:N

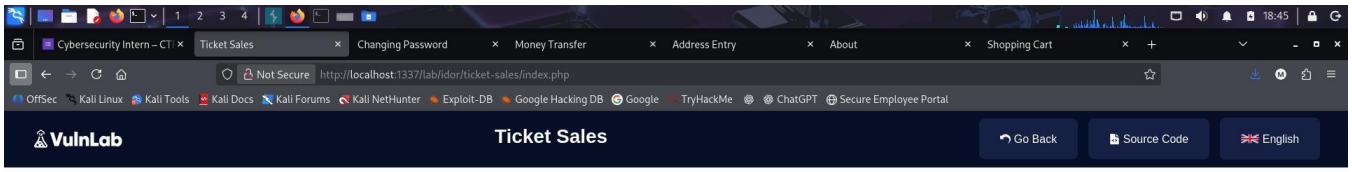
Risk Rating : High

→ Remediation

- Enforce server-side price calculation (do not trust client input)
- Validate ticket prices against server-stored values
- Ignore or remove client-supplied price fields
- Implement integrity checks for purchase requests

→ SCREENSHOT-





Ticket Sales

Reset

The price of one ticket is **10 $**
Amount of money in your account: **990 $**

How many tickets do you want to buy ?

The purchase was successful.

Number of tickets you bought: **2**
Money you pay: **10 $**

Enter the number of tickets:

Enter the number of tickets

Buy

VULN 16

→ Vulnerability Name : Insecure Direct Object Reference (IDOR) – Unauthorized Password Change

→ Description : The application allows a logged-in user to change a password by referencing a user identifier provided by the client. The server does not verify that the authenticated user is authorized to modify the specified account. By altering the user_id parameter, an attacker can change another user's password without permission.

→ Affected Endpoint : POST /lab/idor/changing-password/index.php

→ Proof of Concept (PoC)

Observed Request (modified) : password=test&user_id=2

Result : The server confirms the password change for a different user ("Pierre"), demonstrating unauthorized account modification.

Steps to Reproduce

- Authenticate as a normal user.
- Submit a password change request.
- Modify the user_id parameter to reference another user.
- Send the request.
- Observe a success message indicating another user's password was changed.

→ Impact Analysis

An attacker can:

- Take over other user accounts
- Lock legitimate users out of their accounts
- Escalate privileges and access sensitive data

→ CVSS 3.1 Base Score : 8.1 – High

CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:N

Risk Rating : High

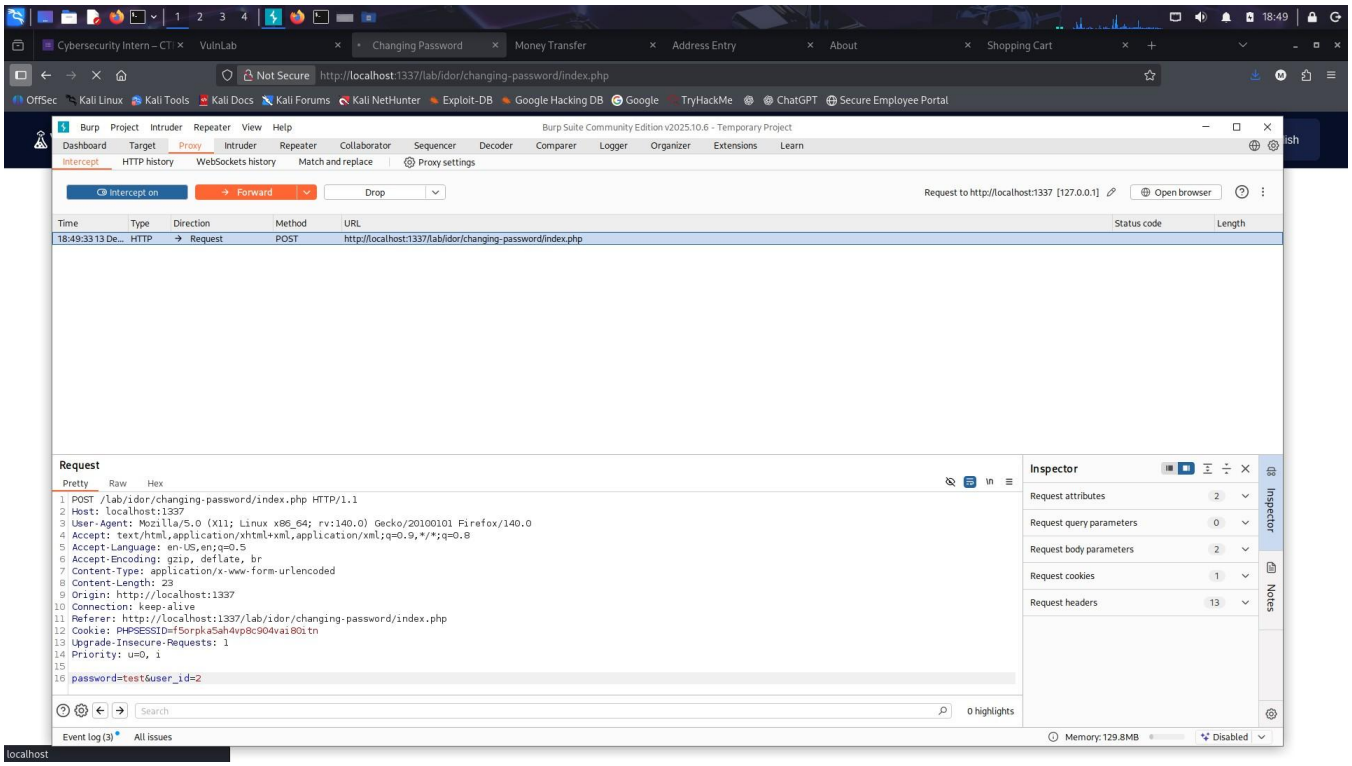
→ Remediation

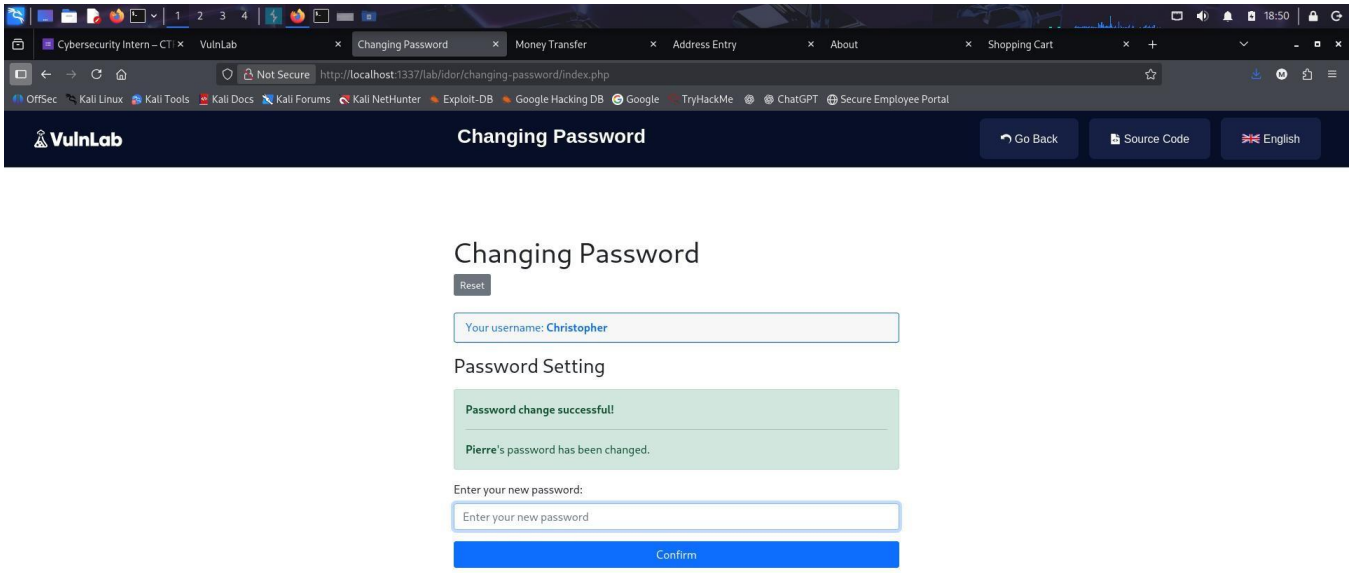
- Enforce server-side authorization checks for password changes
- Ensure the authenticated user can modify only their own account
- Avoid trusting client-supplied user identifiers
- Use indirect references and validate ownership on every request

→ Evidence

Burp Repeater screenshot showing modified user_id

Server response confirming another user's password change





Changing Password

Reset

Your username: **Christopher**

Password Setting

Password change successful!

Pierre's password has been changed.

Enter your new password:

Enter your new password

Confirm

→ Vulnerability Name : Insecure Direct Object Reference (IDOR) – Unauthorized Money Transfer

→ Description : The application relies on client-supplied identifiers to determine the source and destination of a money transfer. By modifying the sender_id and recipient_id parameters, a user can transfer funds from another user's account without authorization. The server fails to verify ownership of the source account.

→ Affected Endpoint : POST /lab/idor/money-transfer/

→ Proof of Concept (PoC)

Modified Request : transfer_amount=100&recipient_id=2&sender_id=1

Result : The transfer succeeds even though the authenticated user is not the owner of sender_id=1, confirming unauthorized fund transfer.

Steps to Reproduce

- Login as a normal user.
- Initiate a money transfer and intercept the request.
- Send the request to Burp Repeater.
- Modify sender_id to another user's account and set a transfer amount.
- Send the request and observe a successful transfer.

→ Impact Analysis

An attacker can:

- Steal funds from other users
- Cause direct financial loss
- Abuse trust and bypass authorization controls

→ CVSS 3.1 Base Score : 8.6 – High

CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:H/A:H

Risk Rating : High

→ Remediation

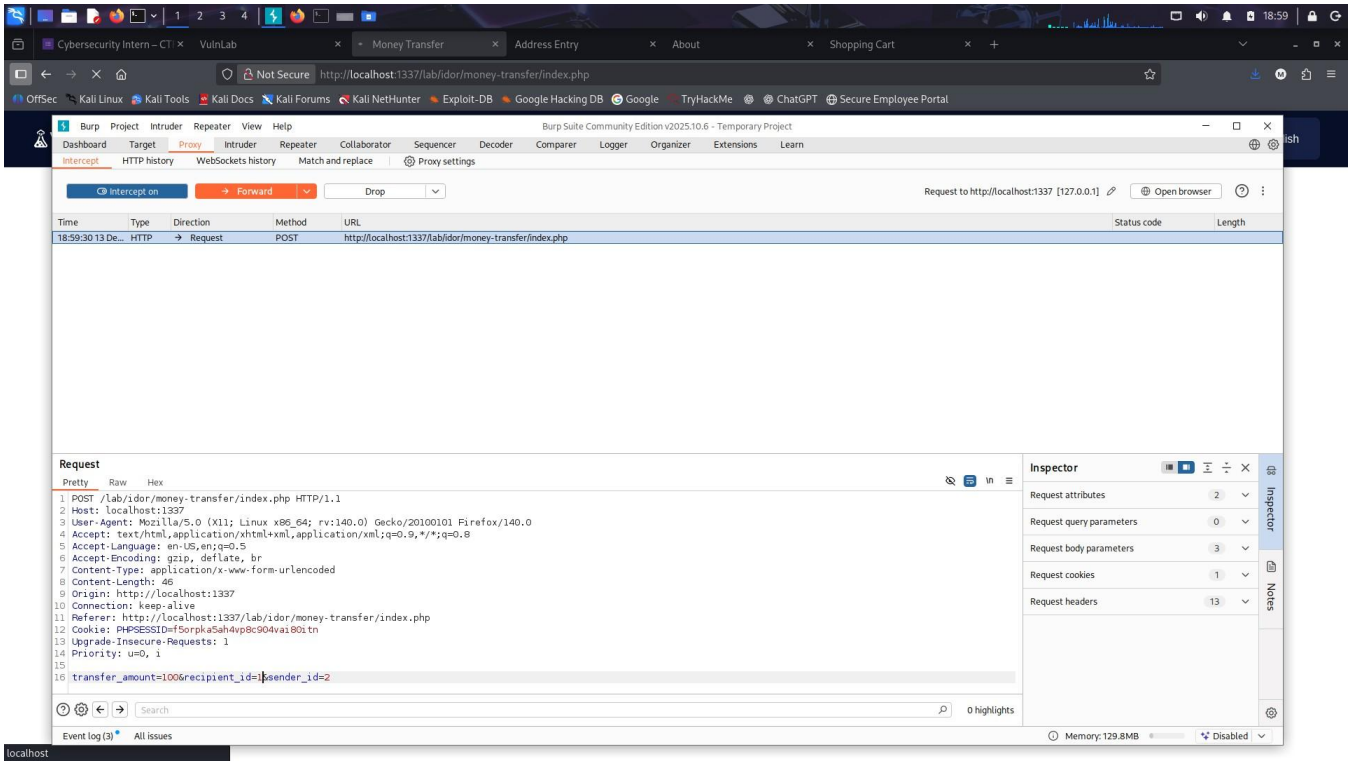
- Enforce strict server-side authorization on transfer operations

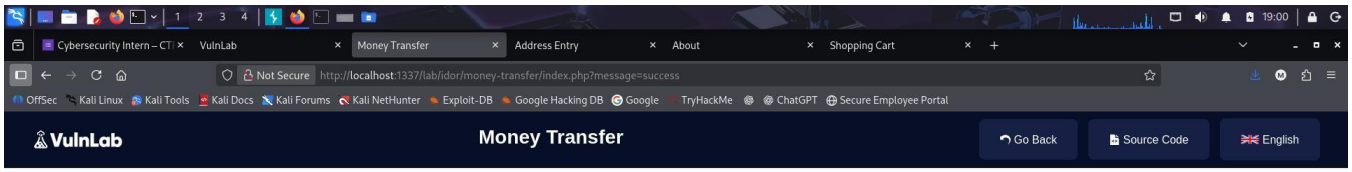
- Derive the sender account from the authenticated session (not client input)
- Validate ownership for all account identifiers
- Implement transaction integrity checks and logging

→ Evidence

Burp Repeater screenshot showing modified sender_id

Server response confirming successful transfer





Money Transfer

Reset

Your account name: **User 1**
Your money in your account: **1100 $**

Money Transfer Transactions

The money transfer was successful!

Transfer amount:

Transfer amount

Receiver ID:

Receiver ID

Confirm

ID	Name	Money
1	User 1	1100 $
2	User 2	900 $
3	User 3	1000 $
4	User 4	1000 $

VULN 18

→ Vulnerability Name : Insecure Direct Object Reference (IDOR) – Unauthorized Address Modification

→ Description : The application allows users to update address records by specifying a client-controlled addressID parameter. The server does not verify whether the authenticated user owns the referenced address. By modifying this identifier, an attacker can update address information belonging to other users without authorization.

→ Affected Endpoint : POST /lab/idor/address-entry/

→ Proof of Concept (PoC)

Modified Request Parameters:

address=HACKED_BY_IDOR

addressID=2

update=

Observation: The request is processed successfully

No authorization error is returned

Backend accepts updates for address records not owned by the authenticated user

This confirms an IDOR vulnerability even though victim data is not displayed in the attacker's UI.

Steps to Reproduce

- Login as a normal user.
- Navigate to the Address Entry page.
- Intercept the update request using Burp Suite.
- Modify the addressID parameter to another user's ID.
- Send the request.
- Observe that the server processes the request without authorization checks.

→ Impact Analysis

An attacker can:

- Modify other users' personal address information
- Corrupt or tamper with sensitive user data
- Cause privacy violations and data integrity issues

→ CVSS 3.1 Base Score : 7.7 – High

CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:L/I:H/A:N

Risk Rating : High

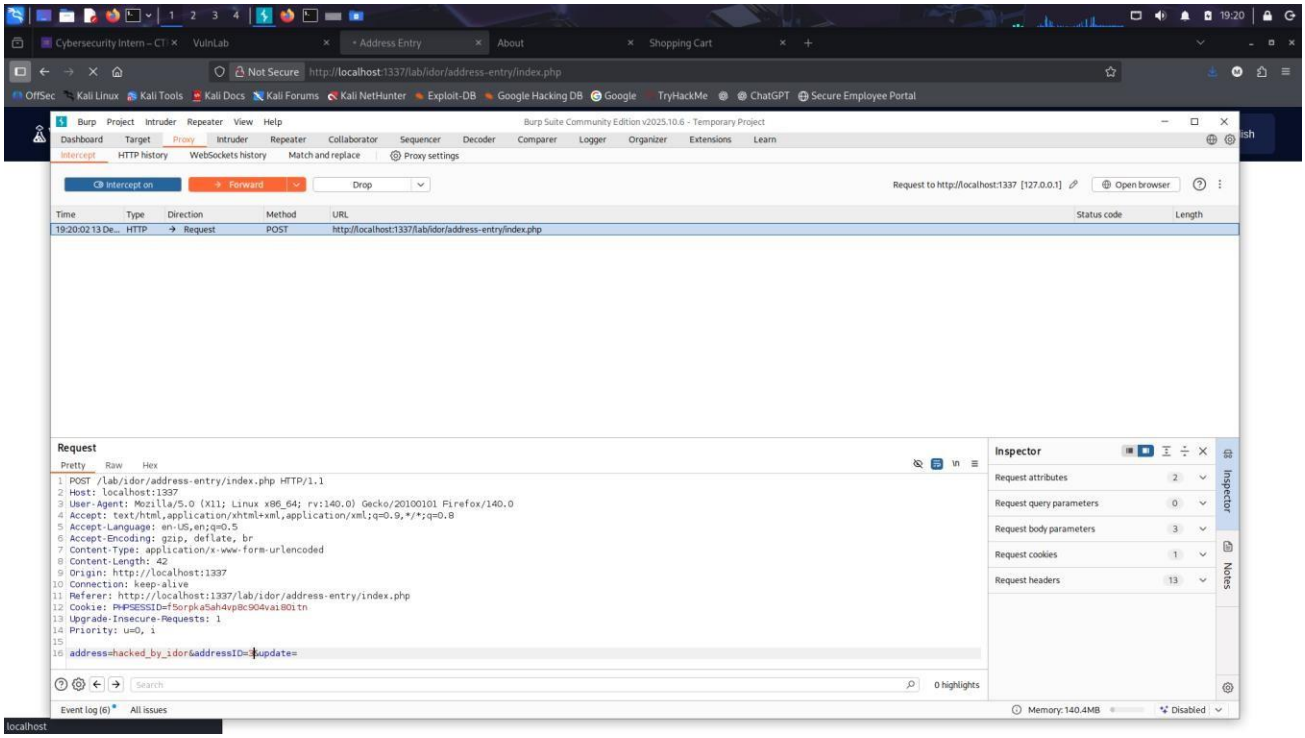
→ Remediation

- Enforce server-side ownership validation for address records
- Derive object ownership from the authenticated session
- Reject requests where the user does not own the referenced object
- Avoid exposing direct object identifiers

→ Evidence

Burp Repeater screenshot showing modified addressID

Server response confirming successful processing without authorization error



Vuln 19

→ Vulnerability Name : Insecure Direct Object Reference (IDOR) – Unauthorized Profile Information Modification

→ Description : The application allows users to update profile (“About”) information by referencing a user identifier supplied by the client. The server does not verify whether the authenticated user is authorized to modify the specified user profile. By altering the user_id parameter, an attacker can update profile information belonging to another user without permission.

→ Affected Endpoint : POST /lab/idor/about/

→ Proof of Concept (PoC)

Modified Request Parameters : user_id=4

(Other profile fields supplied normally)

Result : The server accepts the request and applies profile changes to user ID 4, even though the authenticated user does not own that profile. No authorization error is returned.

Steps to Reproduce

- Login as a normal authenticated user.
- Navigate to the About / Profile page.
- Submit a profile update request and intercept it using Burp Suite.
- Send the request to Burp Repeater.
- Modify the user_id parameter to 4.
- Send the request.
- Observe that the profile update is processed successfully for another user.

→ Impact Analysis

An attacker can:

- Modify other users’ profile information
- Deface user accounts
- Corrupt personal data
- Perform social engineering or impersonation attacks

→ CVSS 3.1 Base Score

7.7 – High
CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:L/I:H/A:N

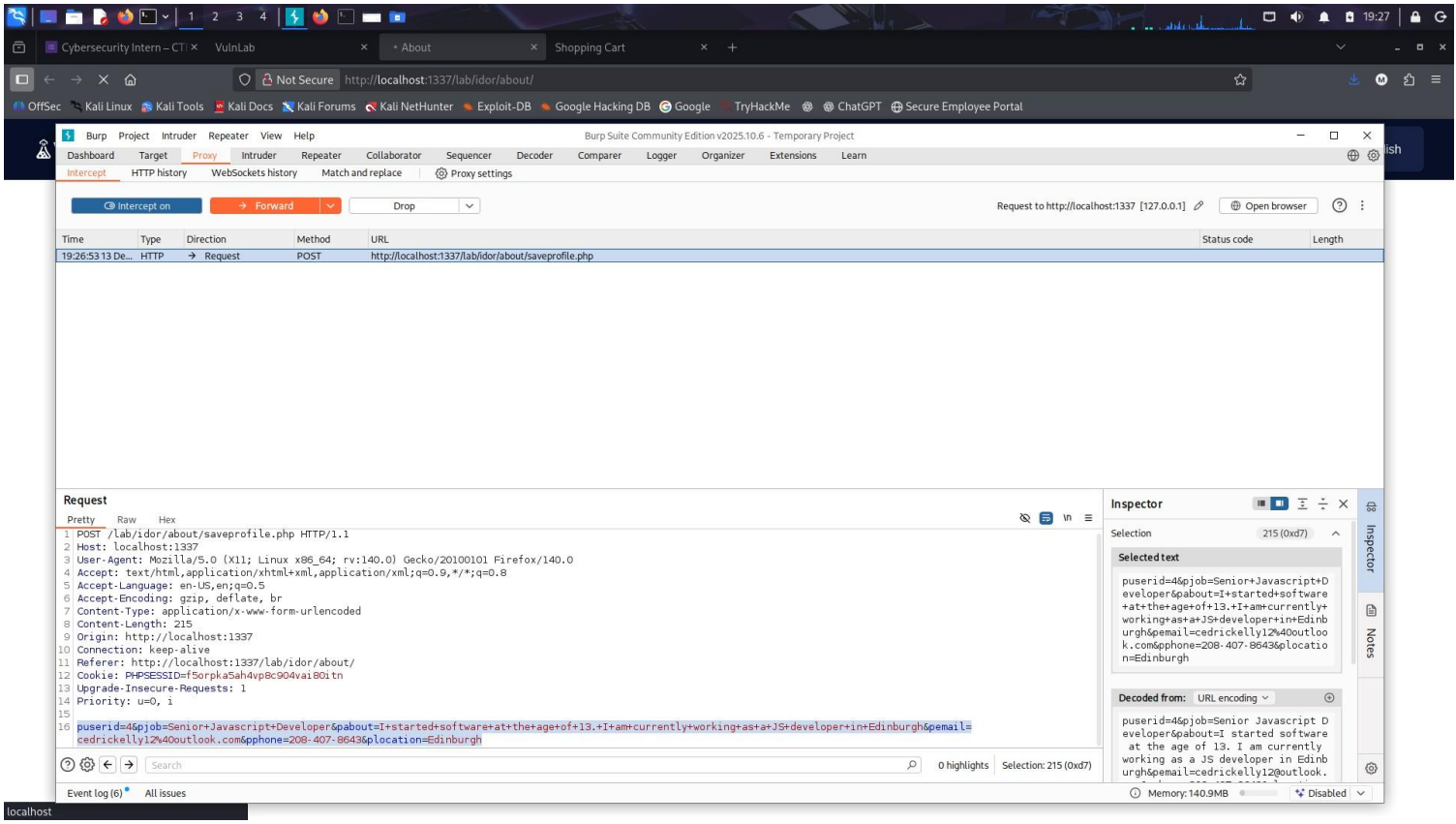
Risk Rating : High

→ Remediation

- Enforce strict server-side authorization checks on profile updates
- Ensure users can modify only their own profiles
- Derive user identity from the authenticated session, not client input
- Avoid using predictable numeric identifiers for sensitive objects

→ Evidence

Burp Repeater screenshot showing modified user_id=4
Server response confirming successful profile update without authorization error



Vuln 20

→ Vulnerability Name : Insecure Direct Object Reference (IDOR) – Unauthorized Cart Manipulation

→ Description : The shopping cart functionality relies on client-supplied object references to add products. The application does not validate whether the requested product reference is authorized for the current user. By modifying encoded request parameters, arbitrary products can be added to the cart, enabling business logic abuse including free or unauthorized purchases.

→ Affected Endpoint : GET /lab/idor/shopping-cart/add.php

→ Proof of Concept : Intercepted cart addition request
Modified encoded product identifier parameter
Server accepted the request without authorization checks

→ Impact
Unauthorized product purchases
Financial loss
Inventory manipulation
Business logic bypass

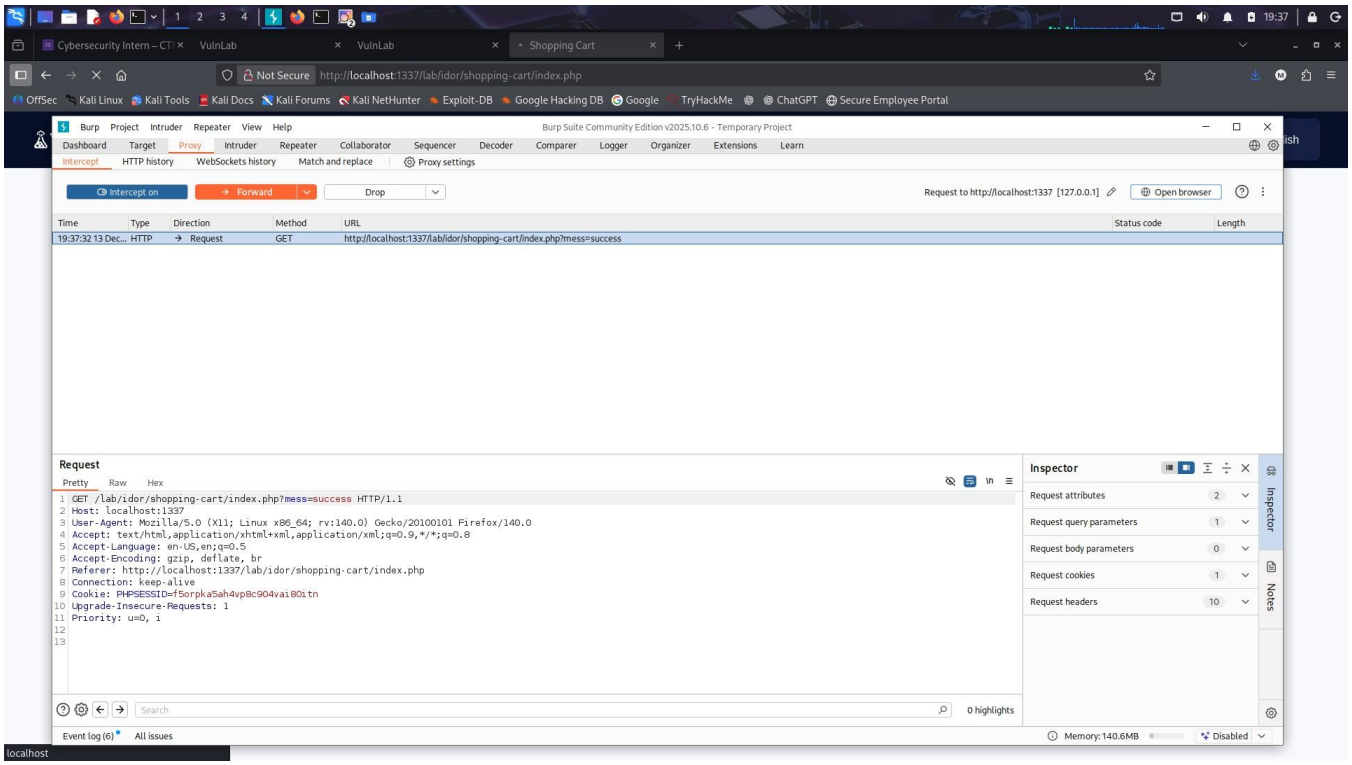
→ CVSS 3.1
CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:H/A:N
Base Score ~7.1(HIGH)

Risk Type:

- **Authorization Bypass (IDOR)**
- **Business Logic Flaw**
- **Integrity Impact**

→ Remediation
- Enforce server-side validation of cart contents
- Do not trust client-supplied object reference
- Validate product ownership and pricing at checkout
- Use integrity checks or server-generated cart tokenn

→ Screenshot



NEXT CATEGORY- COMMAND INJECTION

Vulnerability 21-

SEND PING 1(LAB NAME)

→ Description : The application takes user-supplied input for an IP address and directly concatenates it into a system-level ping command without proper validation or sanitization. This allows an attacker to append arbitrary OS commands and achieve command execution on the server.

→ Affected Endpoint : /lab/command-injection/send-ping/

→ Proof of Concept (PoC)

Payload Used

;id

Injection Point

IP address input field in Send Ping functionality.

Observed Result : The server executed the injected OS command and returned:

uid=33(www-data) gid=33(www-data) groups=33(www-data)

This confirms arbitrary command execution under the web server user context.

Steps to Reproduce

- Navigate to Command Injection → Send Ping.
- Enter a valid IP followed by the payload ;id.
- Submit the request.
- Observe command execution output in the response.

→ Impact Analysis

Successful exploitation allows an attacker to execute arbitrary operating system commands with the privileges of the web server user (www-data). This can lead to:

- Full server compromise
- Sensitive data disclosure

- Privilege escalation
- Malware installation
- Lateral movement within the infrastructure

→ CVSS 3.1 Base Score : 9.8 (Critical)

Vector:

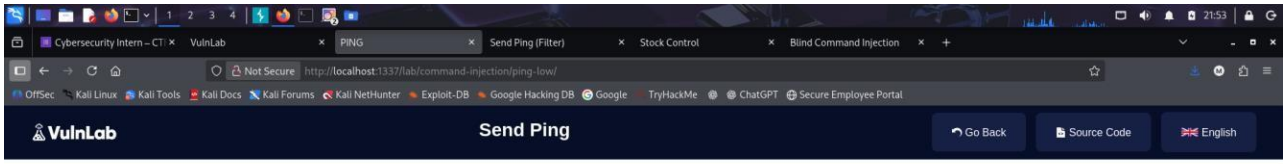
CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

Risk Rating : Critical

→ Remediation / Mitigation

- Do not pass user input directly to OS commands.
- Use safe APIs instead of system calls where possible.
- Strictly validate input using allowlists (e.g., IP address regex).
- Escape shell arguments using secure functions.
- Run services with the least privilege.
- Implement monitoring for command execution abuse.

→ Screenshot Evidence



VULN 22

Vulnerability- SEND PING 2

OS Command Injection – Filter Bypass (Send Ping – Filtered)

→ Description : The application attempts to restrict command injection by filtering common command separators. However, insufficient input validation allows alternative command chaining operators to bypass the filter. As a result, arbitrary OS commands can still be executed on the server.

→ Affected Endpoint : /lab/command-injection/send-ping-filter/

→ Proof of Concept (PoC)
Payload Used

||id

Injection Point

IP address input field in Send Ping (Filtered) functionality.

Observed Result : The injected command executed successfully and returned:

uid=33(www-data) gid=33(www-data) groups=33(www-data)

This confirms that input filtering is inadequate and command injection is still possible.

Steps to Reproduce

- Navigate to Command Injection → Send Ping (Filtered).
- Enter a valid IP followed by the payload ||id.
- Submit the request.
- Observe execution of the OS command in the response.

→ Impact Analysis
An attacker can bypass implemented input filters and execute arbitrary system commands under the web server user (www-data).
This can lead to:
Complete server compromise
Sensitive file access
Remote code execution

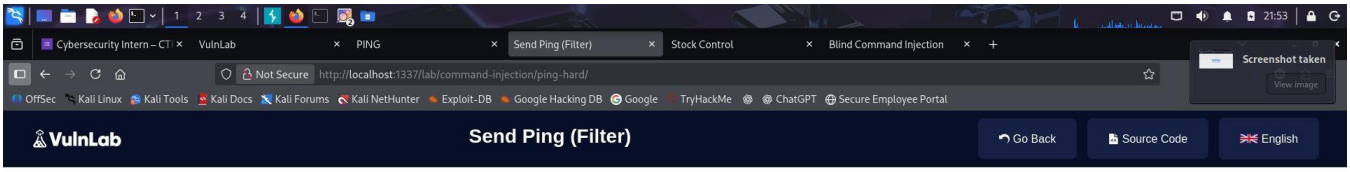
Privilege escalation
Persistent malware installation

→ CVSS 3.1 Base Score : 9.8 (Critical)
Vector:
CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H
→ Risk Rating : Critical

→ Remediation / Mitigation

- Do not rely on blacklist-based filtering.
- Avoid executing OS commands using user input.
- Enforce strict allowlist validation for IP addresses.
- Use secure command execution libraries or APIs.
- Apply proper escaping of shell arguments.
- Run services with minimal privileges.

→ Screenshot Evidence



VULN 23

Vulnerability- STOCK CONTROL

OS Command Injection – Stock Control Endpoint

→ Description : The Stock Control functionality uses a user-supplied product_id parameter to execute a backend command for checking inventory. The application fails to properly validate or sanitize this parameter, allowing attackers to append arbitrary OS commands to the original system call.

→ Affected Endpoint : /lab/command-injection/stock-check/?product_id=<input>

→ Proof of Concept (PoC)

Payload Used

1;id

Full Exploited URL:

http://localhost:1337/lab/command-injection/stock-check/?product_id=1;id

Observed Result : The response included both legitimate stock output and OS command execution output:

Stock: 13 uid=33(www-data) gid=33(www-data) groups=33(www-data) Pieces

This confirms arbitrary command execution on the server.

Steps to Reproduce

- Navigate to Command Injection → Stock Control.
- Modify the product_id parameter in the URL.
- Append the payload ;id to the parameter value.
- Observe OS command output appended to the stock response.

→ Impact Analysis

Successful exploitation allows attackers to execute arbitrary operating system commands under the web server context (www-data). This can lead to:

- Full system compromise
- Unauthorized data access
- Remote code execution

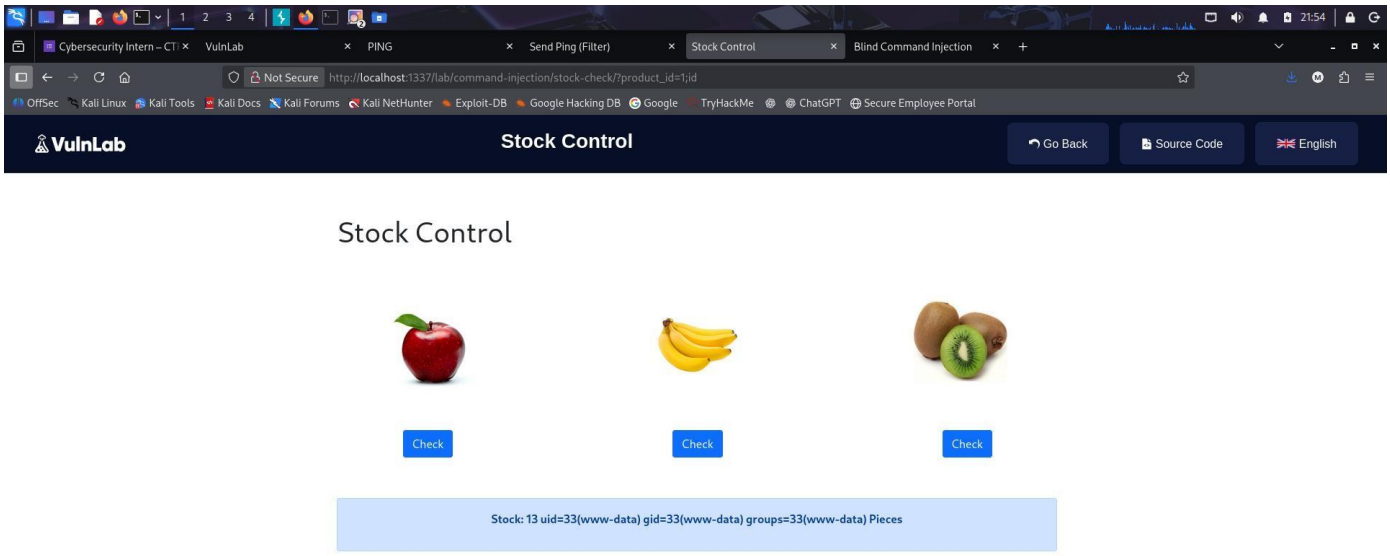
- Privilege escalation
- Service disruption

→ CVSS 3.1 Base Score : 9.8 (Critical)
Vector:
CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H
→ Risk Rating : Critical

→ Remediation / Mitigation

- Never concatenate user input into OS commands.
- Replace system calls with secure APIs or internal logic.
- Strictly validate input using allowlists (e.g., numeric-only product IDs).
- Escape shell arguments using safe libraries.
- Apply least-privilege execution for backend services.

→ Screenshot Evidence



VULN 24

Vulnerability- Blind Command Injection

Blind OS Command Injection – User-Agent Header

→ Description : The application processes HTTP headers without proper validation and passes the User-Agent header value into an underlying operating system command. Although command output is not returned in the HTTP response, arbitrary command execution is confirmed through time-based behavior, indicating a blind command injection vulnerability.

→ Affected Endpoint : /lab/command-injection/blind/

→ Injection Point

HTTP Header:

User-Agent

→ Proof of Concept (PoC)

Payload Used

; sleep 10

Method

The payload was injected into the User-Agent HTTP header using Burp Suite.

Observed Behavior

Normal request: immediate response

Injected request: response delayed by ~10 seconds

This confirms that the injected command was executed on the server, even though no output was displayed.

Steps to Reproduce

- Intercept the request using Burp Suite.
- Modify the User-Agent header to include:

; sleep 10

- Forward the request.
- Observe a significant response delay compared to the baseline request.

→ Impact Analysis

Blind command injection allows attackers to execute arbitrary OS commands without direct output. This can still lead to:

- Full server compromise

- Data exfiltration via time-based inference
- Remote code execution
- Privilege escalation
- Persistence on the host
- Even without visible output, this vulnerability is highly critical.

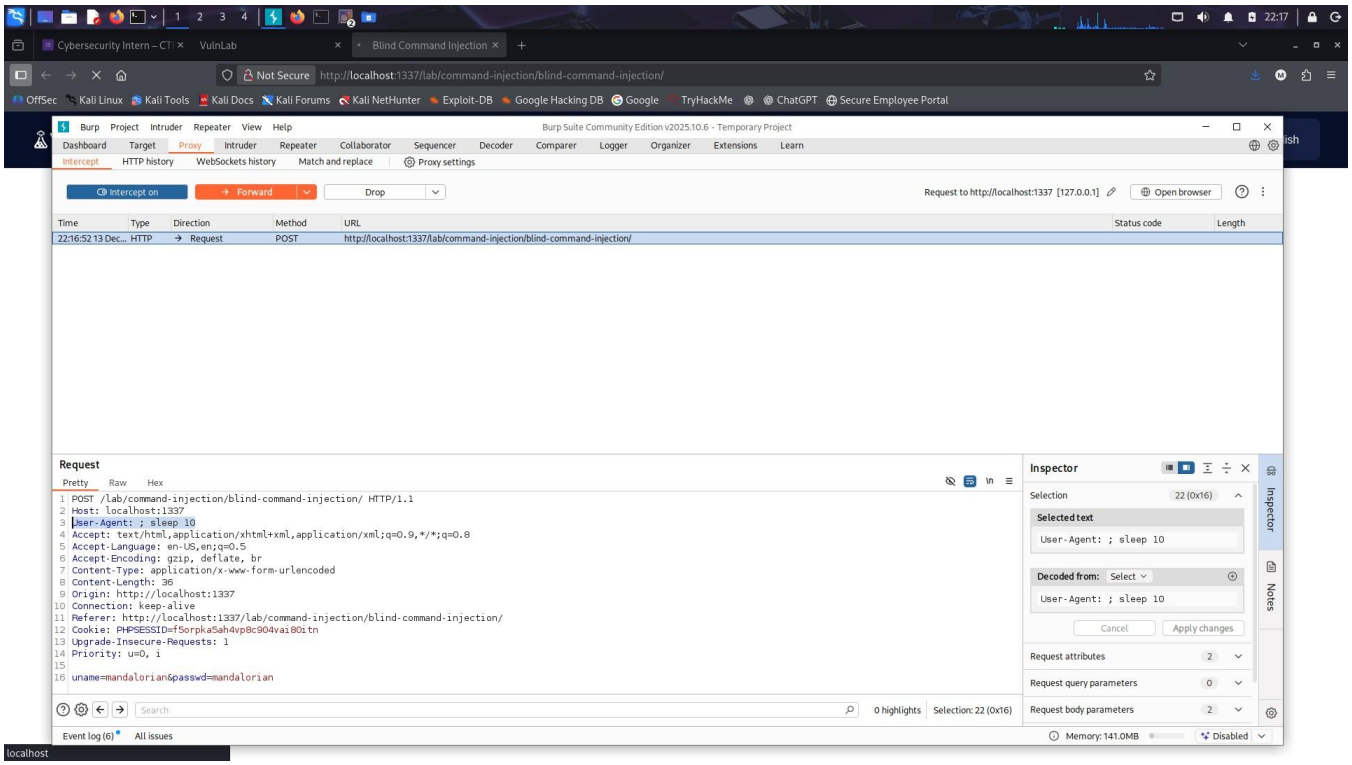
→ CVSS 3.1 Base Score : 9.8 (Critical)
Vector:
CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

Risk Rating : Critical

→ Remediation / Mitigation

- Never pass HTTP header values to OS commands.
- Sanitize and validate all user-controlled headers.
- Use secure APIs instead of shell execution.
- Implement strict allowlists.
- Apply least-privilege execution for backend processes.
- Monitor for abnormal request execution times.

→ Screenshot Evidence



NEXT CATEGORY-XML EXTERNAL ENTITY ATTACK(XXE)

VULN 25

VULN 25- XML EXTERNAL FILE EXTRACTION

→ Description : The application accepts XML input and parses it on the server side without disabling external entity resolution. This allows an attacker to define a malicious external entity that references a local system file. When the XML is processed, the parser expands the entity and discloses the file contents in the HTTP response.

→ Affected Endpoint : POST /lab/xxe/xml/test.php

→ Content-Type: text/xml

→ Root Cause

XML input is parsed using an insecure XML parser configuration.
External entities (<!ENTITY ... SYSTEM>) are allowed.
User-controlled XML values are reflected in the server response.
No protection such as disallow-doctype-decl or external entity disabling is implemented.

→ Proof of Concept (PoC)

An XML request was sent containing:
A DOCTYPE declaration
An external entity referencing a local system file
The entity injected into a reflected XML element
When the request was processed, the server expanded the entity and returned the contents of the referenced file.
Evidence of Exploitation
The HTTP response returned the contents of /etc/passwd, including system users such as:
root:x:0:0:root:/root:/bin/bash
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
mysql:x:101:102:MySQL Server,,,:/nonexistent:/bin/false

This confirms successful external entity expansion and local file disclosure.

→ Impact

- Arbitrary file read on the server

- Disclosure of sensitive system information
- Potential for further attacks such as:
  Credential harvesting
  Internal service enumeration
  SSRF (in advanced XXE scenarios)

→ Severity : High

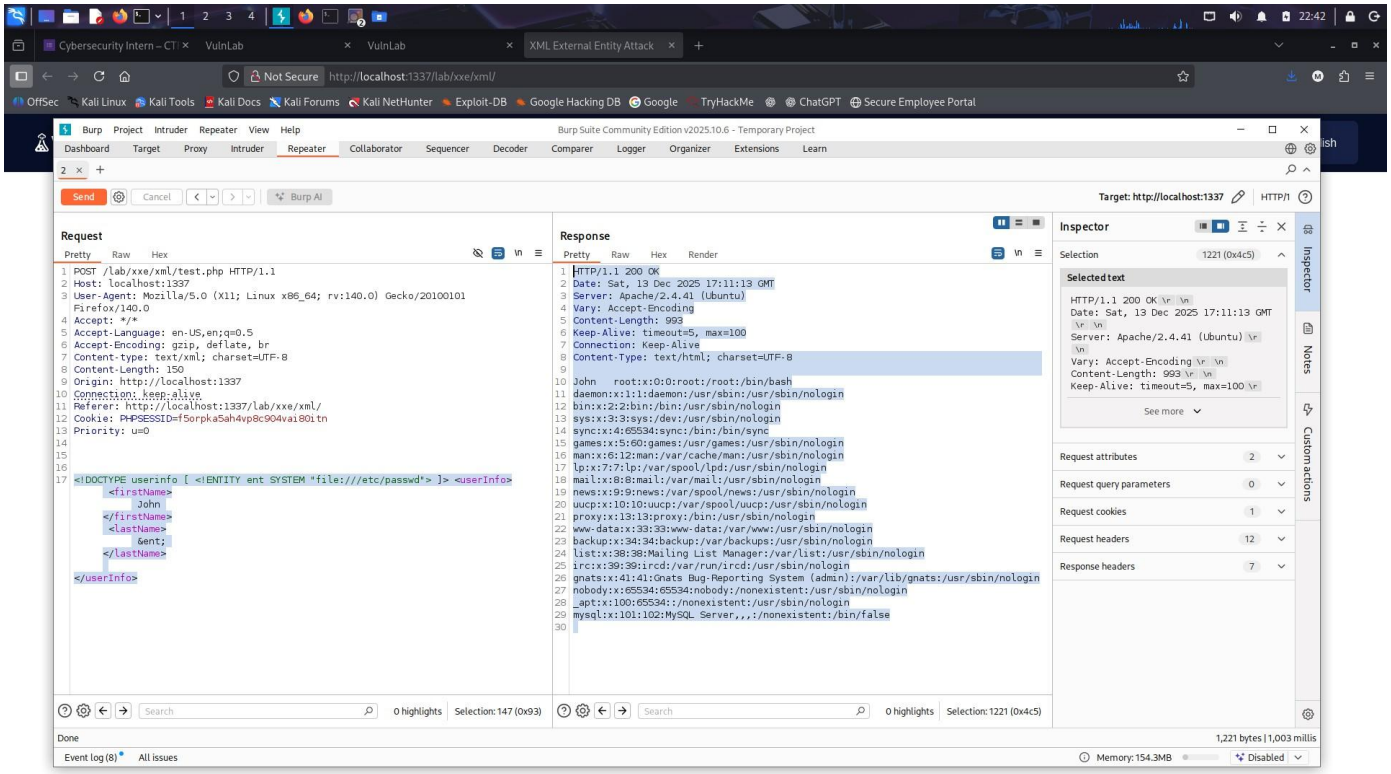
→ Remediation

- Disable external entity processing in the XML parser.
- Disallow DOCTYPE declarations entirely.
- Use secure parser configurations (e.g., libxml_disable_entity_loader).
- Validate and sanitize XML input.
- Prefer safer data formats such as JSON where possible.

→ Conclusion

The application is vulnerable to XML External Entity injection due to insecure XML parsing. An attacker can exploit this flaw to read sensitive files from the server, as demonstrated by successful disclosure of /etc/passwd.

→ Screenshot


Cybersecurity Intern – CT
VulnLab
VulnLab
XML External Entity Attack
Not Secure
http://localhost:1337/lab/xxe/xml/
Burp Suite Community Edition v2025.10.6 - Temporary Project
Request
POST /lab/xxe/xml/test.php HTTP/1.1
Host: localhost:1337
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:140.0) Gecko/20100101 Firefox/140.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-type: text/xml; charset=UTF-8
Content-Length: 150
Origin: http://localhost:1337
Connection: keep-alive
Referer: http://localhost:1337/lab/xxe/xml/
Cookie: PHPSESSID=f5orpka5ah4vp8c904vai80itn
Priority: u=0
<!DOCTYPE userinfo [<!ENTITY ent SYSTEM "file:///etc/passwd">]> <userInfo>
<firstName>
John
</firstName>
<lastName>
&ent;
</lastName>
</userInfo>
Response
HTTP/1.1 200 OK
Date: Sat, 13 Dec 2025 17:11:13 GMT
Server: Apache/2.4.41 (Ubuntu)
Vary: Accept-Encoding
Content-Length: 993
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
John root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
mysql:x:101:102:MySQL Server,,,:/nonexistent:/bin/false
Inspector
Request attributes
Request query parameters
Request cookies
Request headers
Response headers
Done
1,221 bytes | 1,003 millis

NEXT CATEGORY- FILE INCLUSION

VULN 26 LEARN THE CAPITAL 1

→ Description : The application dynamically includes files based on user-controlled input without proper validation. By manipulating the file path parameter, an attacker can traverse directories and include sensitive internal files, such as administrative pages, resulting in unauthorized access.

→ Affected Endpoint : /lab/file-inclusion/learn-the-capital-1/

→ Proof of Concept (PoC)
Payload Used
../admin.php

Injection Method
The payload was injected into the file inclusion parameter using Burp Suite, allowing traversal outside the intended directory.

Observed Result : The application successfully loaded the admin page, granting unauthorized access without authentication.

Steps to Reproduce

- Navigate to File Inclusion → Learn the Capital 1.
- Intercept the request using Burp Suite.
- Modify the file parameter to:
  ../admin.php
- Forward the request.
- Observe that the admin page is displayed without permission.

→ Impact Analysis
Exploitation of this vulnerability allows attackers to access restricted files and sensitive application components. This can lead to:

- Unauthorized access to admin functionality
- Disclosure of sensitive configuration files
- Application compromise

- Further exploitation (RCE if combined with other flaws)

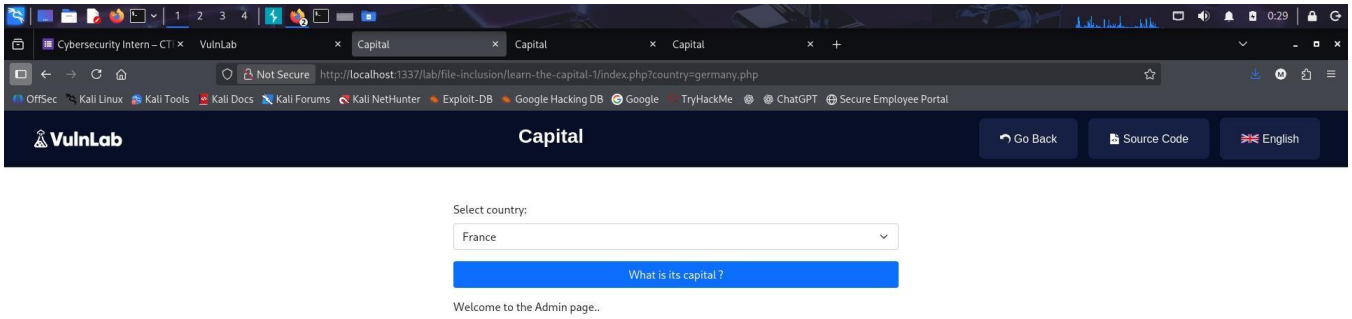
→ CVSS 3.1 Base Score : 8.6 (High)
Vector:
CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:L/A:N

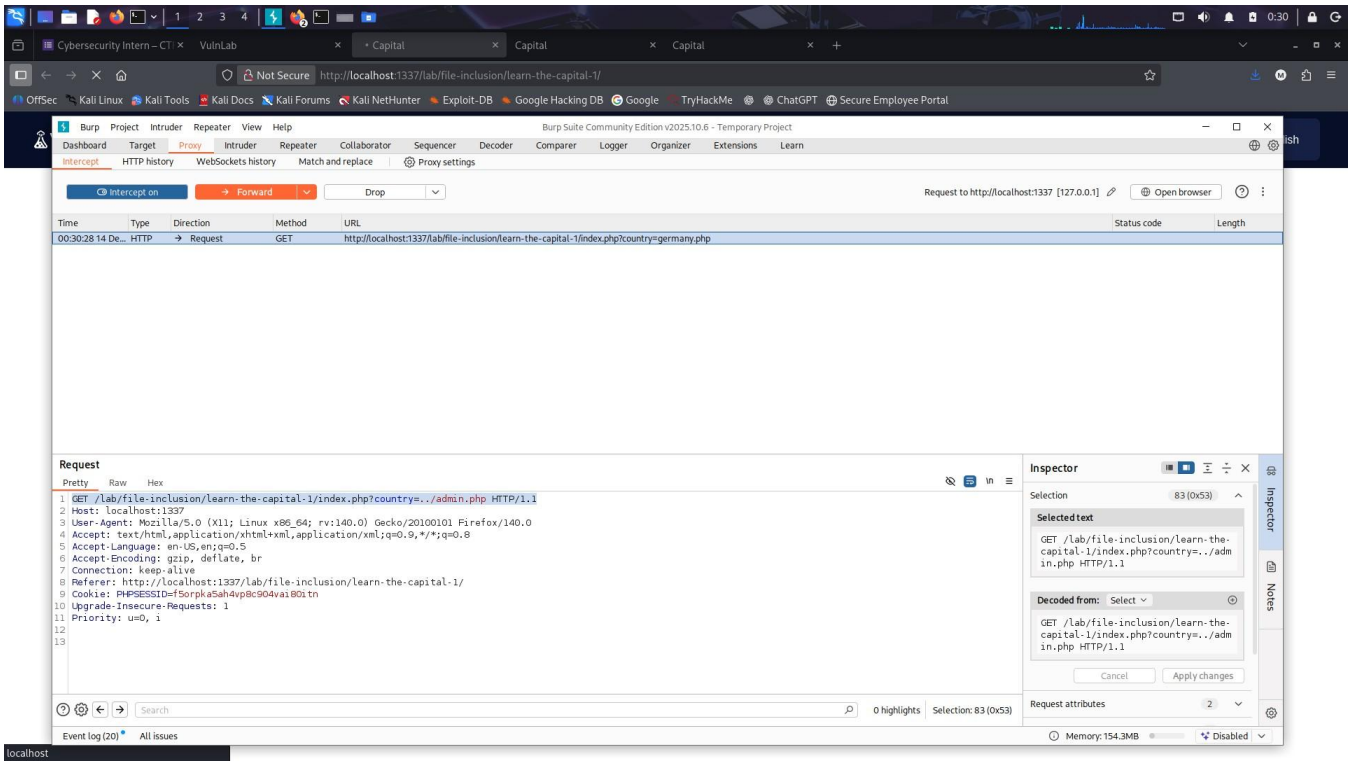
→ Risk Rating : High

→ Remediation / Mitigation

- Do not include files based on user input.
- Implement strict allowlists for includable files.
- Use absolute paths and fixed file mappings.
- Disable directory traversal (../) patterns.
- Apply proper access control checks for admin resources.

→ Screenshot Evidence





VULN 27 LEARN THE CAPITAL 2

→ Description : The application attempts to implement security measures to prevent direct file inclusion attacks. However, improper filtering allows an attacker to bypass these protections by manipulating directory traversal patterns. This results in unauthorized access to restricted files, including administrative pages.

→ Affected Endpoint : /lab/file-inclusion/learn-the-capital-2/

→ Proof of Concept (PoC)

Payload Used

..../admin.php

Injection Method

The payload was injected into the file inclusion parameter using Burp Suite, bypassing the implemented directory traversal filter.

Observed Result : The application successfully loaded the admin page without authentication, confirming that the security measures can be bypassed.

Steps to Reproduce

- Navigate to File Inclusion → Learn the Capital 2.
- Intercept the request using Burp Suite.
- Modify the file parameter to:
  ..../admin.php
- Forward the request.
- Observe that the admin page is displayed without proper authorization.

→ Impact Analysis

Successful exploitation allows attackers to bypass access controls and include sensitive internal files. This can lead to:

- Unauthorized access to administrative interfaces
- Exposure of sensitive application files
- Further exploitation chains (e.g., RCE with file upload or logs)

- Complete application compromise

→ CVSS 3.1 Base Score
8.6 (High)

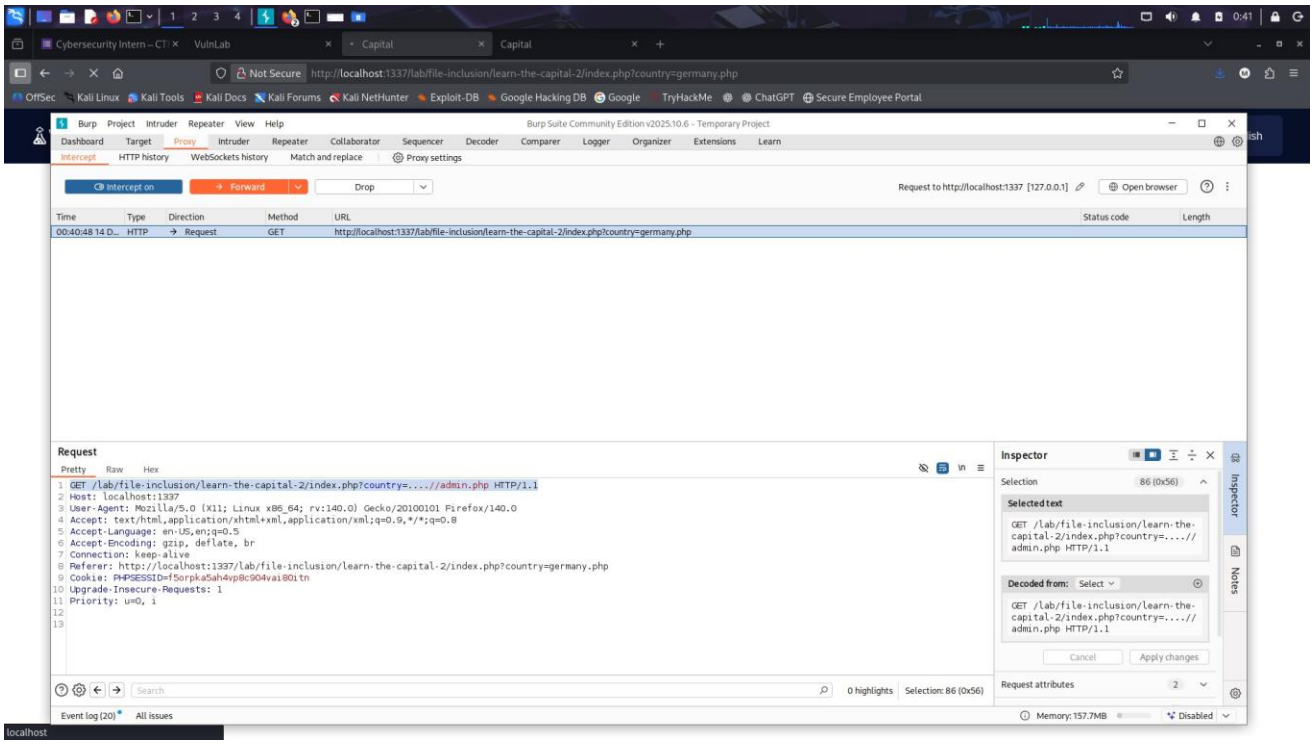
Vector:
CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:L/A:N

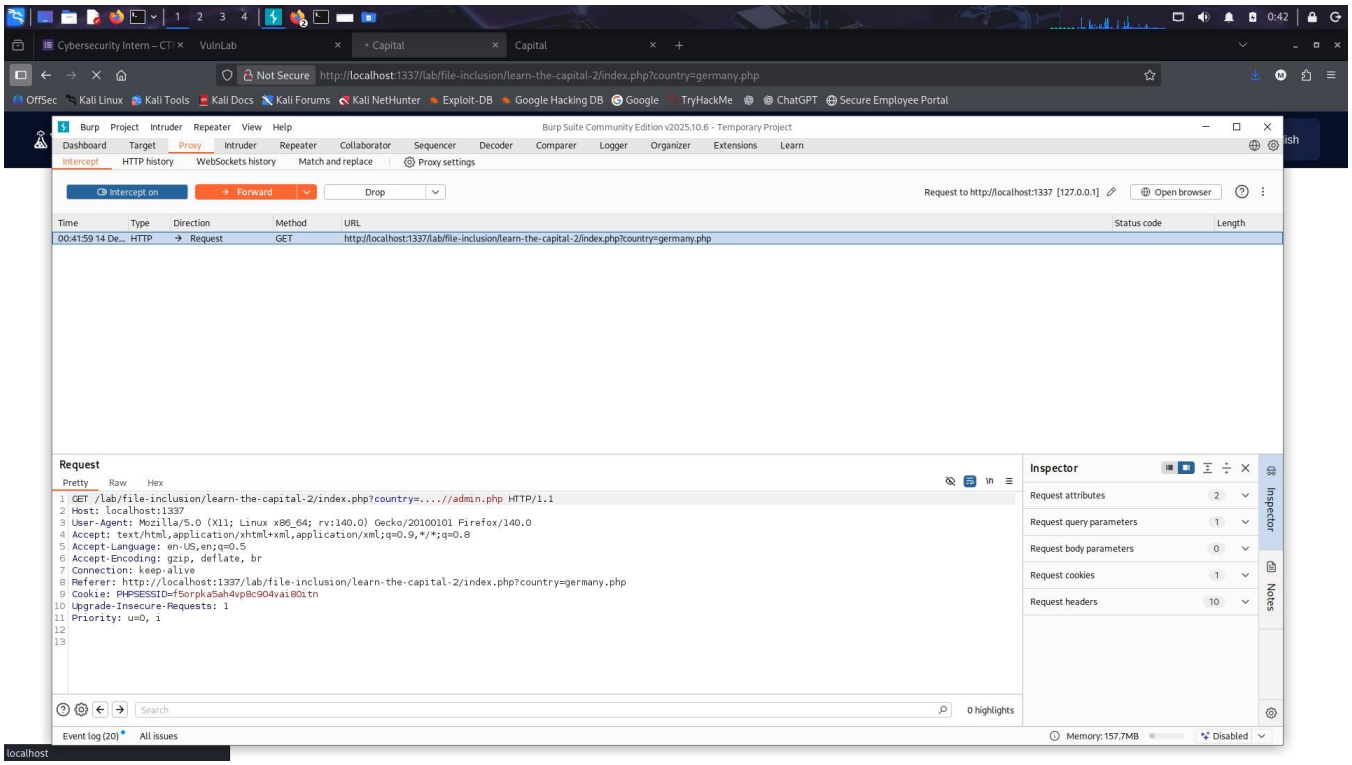
Risk Rating : High

→ Remediation / Mitigation

- Avoid using user input directly in file include operations.
- Implement strict allowlists for valid file names.
- Normalize and validate file paths before processing.
- Block traversal sequences and encoded bypass techniques.
- Enforce authentication and authorization checks on admin resources.

→ Screenshot Evidence




Cybersecurity Intern – CT
VulnLab
Capital
Capital
Not Secure
http://localhost:1337/lab/file-inclusion/learn-the-capital-2/index.php?country=germany.php
OffSec
Kali Linux
Kali Tools
Kali Docs
Kali Forums
Kali NetHunter
Exploit-DB
Google Hacking DB
Google
TryHackMe
ChatGPT
Secure Employee Portal
Burp Project Intruder Repeater View Help
Burp Suite Community Edition v2025.10.6 - Temporary Project
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn
Intercept HTTP history WebSockets history Match and replace Proxy settings
Intercept on
Forward
Drop
Request to http://localhost:1337 [127.0.0.1]
Open browser
Time Type Direction Method URL Status code Length
00:41:59 14 De... HTTP Request GET http://localhost:1337/lab/file-inclusion/learn-the-capital-2/index.php?country=germany.php
Request
Pretty Raw Hex
1 GET /lab/file-inclusion/learn-the-capital-2/index.php?country=....//admin.php HTTP/1.1
2 Host: localhost:1337
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:140.0) Gecko/20100101 Firefox/140.0
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://localhost:1337/lab/file-inclusion/learn-the-capital-2/index.php?country=germany.php
9 Cookie: PHPSESSID=f5orpka5ah4vp8c904vai8o1tn
10 Upgrade-Insecure-Requests: 1
11 Priority: u=0, i
12
13
Search
0 highlights
Inspector
Request attributes 2
Request query parameters 1
Request body parameters 0
Request cookies 1
Request headers 10
Inspector
Notes
Event log (20)
All issues
Memory: 157.7MB
Disabled
localhost

VULN 28 LEARN THE CAPITAL 3

→ DESCRIPTION- The application attempts to restrict file inclusion by enforcing a whitelist directory (file/). However, it fails to properly normalize and validate the requested file path. By using directory traversal sequences, an attacker can escape the whitelisted directory and include sensitive internal files, such as the administrative page, without authorization.

→ Affected Endpoint /lab/file-inclusion/learn-the-capital-3/

→ Proof of Concept (PoC)
Payload Used file/../../admin.php

Injection Point

File inclusion parameter (manipulated using Burp Suite).

Observed Result : The application successfully included and rendered the admin page, granting unauthorized access despite the presence of whitelist-based security controls.

Steps to Reproduce

- Navigate to File Inclusion → Learn the Capital 3.
- Intercept the request using Burp Suite.
- Identify the file inclusion parameter.
- Replace its value with:
  File/../../admin.php
- Forward the request.
- Observe that the admin page is displayed without authentication.

→ Impact Analysis

This vulnerability allows attackers to bypass whitelist-based file inclusion controls and access sensitive internal files. Potential impacts include:

- Unauthorized access to admin functionality
- Disclosure of sensitive application files
- Bypass of access controls
- Full application compromise when chained with other vulnerabilities

→ CVSS 3.1 Base Score
8.6 (High)

Vector:

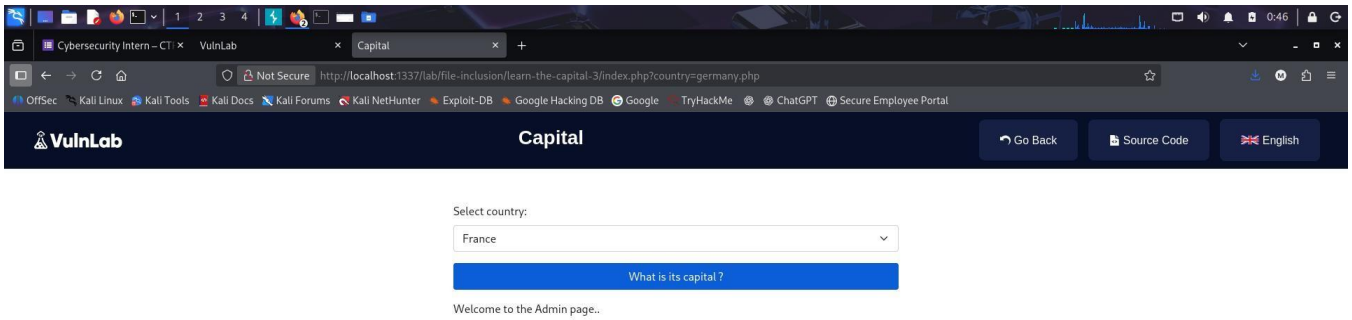
CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:L/A:N

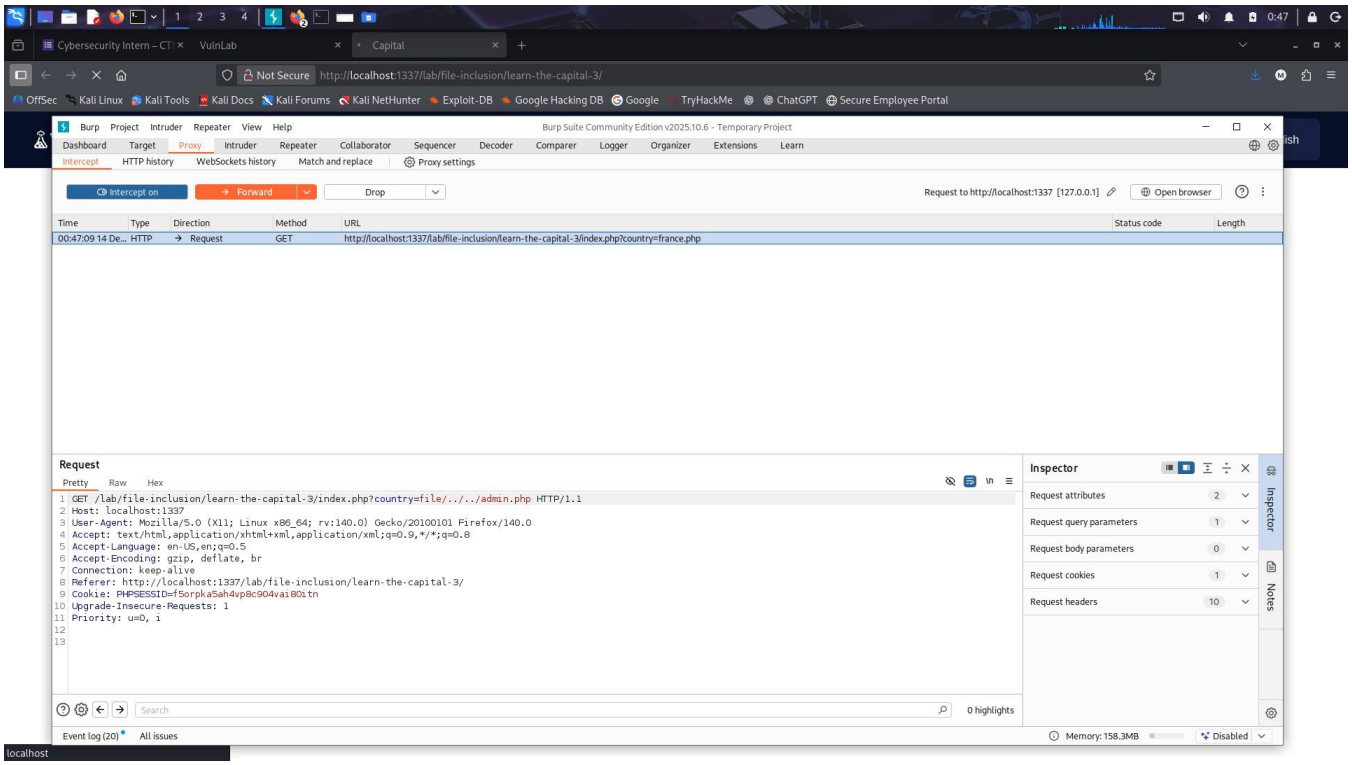
→ Risk Rating : High

→ Remediation / Mitigation

- Do not rely solely on directory-based whitelisting.
- Normalize file paths before validation.
- Reject traversal sequences such as ../ after normalization.
- Use fixed file mappings instead of dynamic includes.
- Enforce authentication and authorization checks for admin resources.

→ Screenshot Evidence




Cybersecurity Intern – CT
VulnLab
Capital
Not Secure
http://localhost:1337/lab/file-inclusion/learn-the-capital-3/
OffSec
Kali Linux
Kali Tools
Kali Docs
Kali Forums
Kali NetHunter
Exploit-DB
Google Hacking DB
Google
TryHackMe
ChatGPT
Secure Employee Portal
Burp Project Intruder Repeater View Help
Burp Suite Community Edition v2025.10.6 - Temporary Project
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn
Intercept HTTP history WebSockets history Match and replace Proxy settings
Intercept on
Forward
Drop
Request to http://localhost:1337 [127.0.0.1]
Open browser
Time Type Direction Method URL Status code Length
00:47:09 14 De... HTTP Request GET http://localhost:1337/lab/file-inclusion/learn-the-capital-3/index.php?country=france.php
Request
Pretty Raw Hex
1 GET /lab/file-inclusion/learn-the-capital-3/index.php?country=file/../../admin.php HTTP/1.1
2 Host: localhost:1337
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:140.0) Gecko/20100101 Firefox/140.0
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://localhost:1337/lab/file-inclusion/learn-the-capital-3/
9 Cookie: PHPSESSID=f5orpka5ah4vp8c904vai8O1tn
10 Upgrade-Insecure-Requests: 1
11 Priority: u=0, i
12
13
Search
0 highlights
Inspector
Request attributes 2
Request query parameters 1
Request body parameters 0
Request cookies 1
Request headers 10
Inspector
Notes
Event log (20)
All issues
Memory: 158.3MB
Disabled
localhost

NEXT CATEGORY- UNRESTRICTED FILE UPLOAD

VULN 29- UNRESTRICTED

→ Description : The application allows users to upload files but does not properly restrict the file types to the allowed formats (gif, jpg, jpeg, png). As a result, an attacker can upload a disallowed file type, such as a .py script, which is not an allowed image format. This could lead to unauthorized files being stored on the server and potentially executed.

→ Affected Endpoint : POST /lab/unrestricted-file-upload/

→ Proof of Concept (PoC)

Payload
Uploaded file: solve.py

Steps to Reproduce

- Navigate to the Unrestricted File Upload lab.
- Use the file upload functionality to upload a file with a disallowed extension, e.g., solve.py.
- Submit the form.
- Observe that the file is accepted and uploaded successfully, with the server returning a confirmation message and file path.

→ Impact Analysis

Allowing unrestricted file uploads can lead to several risks:

- Remote Code Execution: If the server executes the uploaded file.
- Malware Upload: Storing malicious scripts on the server.
- Information Disclosure: Exposing sensitive files if accessed publicly.

This vulnerability can enable attackers to place arbitrary files on the server, bypassing intended file type restrictions.

→ CVSS 3.1 Base Score
7.2 (High)

Vector:

CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:L/A:L

→ Risk Rating : High

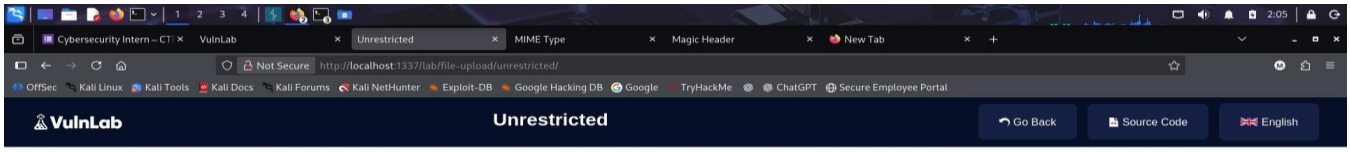
→ Remediation / Mitigation

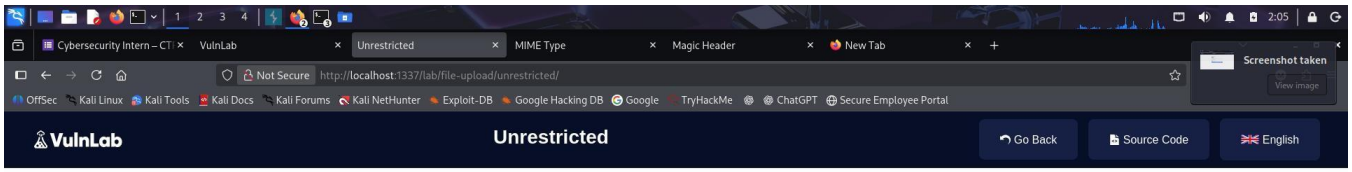
- Enforce strict server-side validation of allowed file extensions.
- Validate file content by checking MIME types and magic bytes.
- Store uploaded files in a non-web-accessible directory.
- Implement file integrity checks before processing.

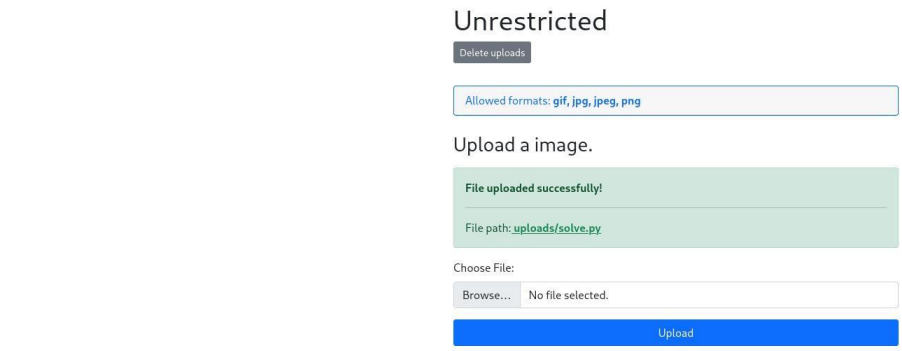
→ Evidence / Screenshot

Screenshot showing successful upload of the solve.py file.

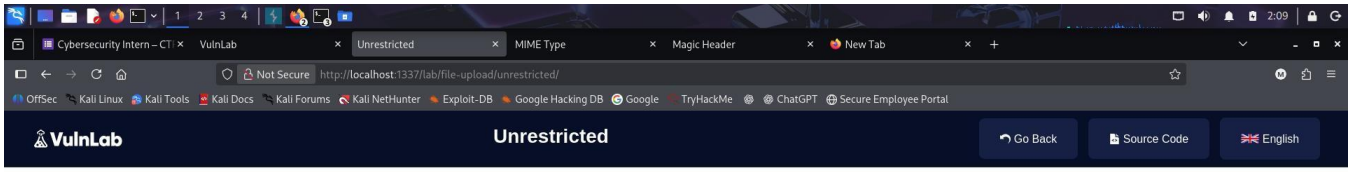
Burp request/response demonstrating the bypass of file type restrictions.

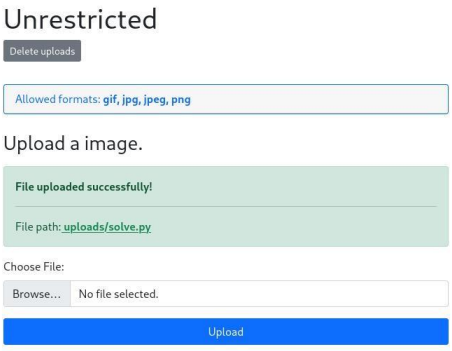






http://localhost:1337/lab/file-upload/unrestricted/uploads/solve.py





VULN 30- MIME TYPE

→ Description : The application enforces file upload restrictions based solely on the Content-Type (MIME type) provided by the client. Since MIME types are user-controlled, an attacker can tamper with the request and upload disallowed file types by spoofing an allowed MIME type (e.g., image/png). This results in successful upload of non-image files, potentially leading to remote code execution.

→ Affected Endpoint : POST /lab/file-upload/mime-type/index.php

→ Proof of Concept (PoC)
Action:
Uploaded a .py file.
Intercepted the request in Burp Suite.
Changed the Content-Type from its original value to an allowed type.
Modified Header:
Content-Type: image/png

Result: The server accepted and uploaded the .py file successfully, confirming MIME type validation is insufficient and client-controlled.

Steps to Reproduce

- Navigate to the MIME Type file upload lab.
- Select a disallowed file (e.g., solve.py).
- Intercept the upload request in Burp Suite.
- Change the Content-Type header to image/png.
- Forward the request.
- Observe a successful upload message and file path.

→ Impact Analysis
An attacker can:

- Upload arbitrary files (scripts, shells)
- Potentially execute malicious code on the server
- Compromise server integrity and confidentiality
- Escalate to full server takeover depending on execution context

→ CVSS 3.1 Base Score
8.8 (High)

Vector:

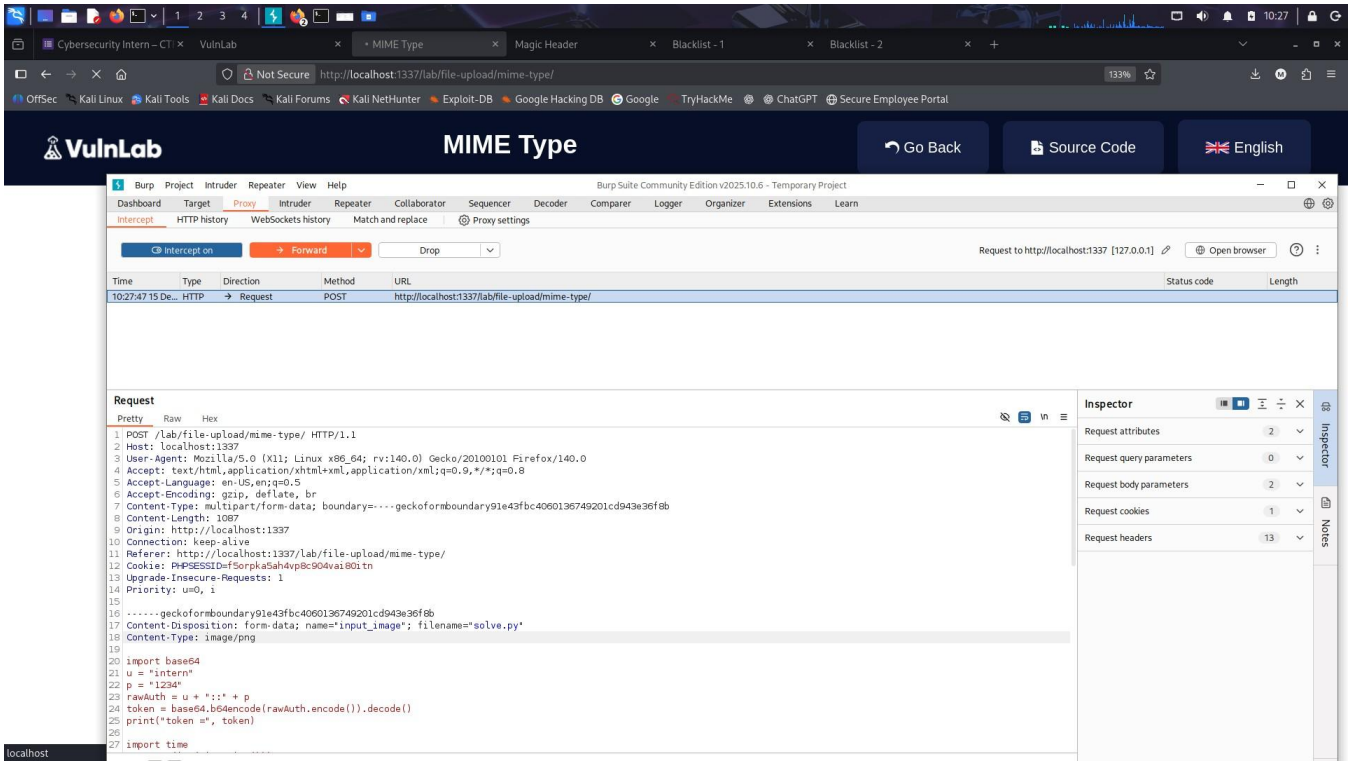
CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

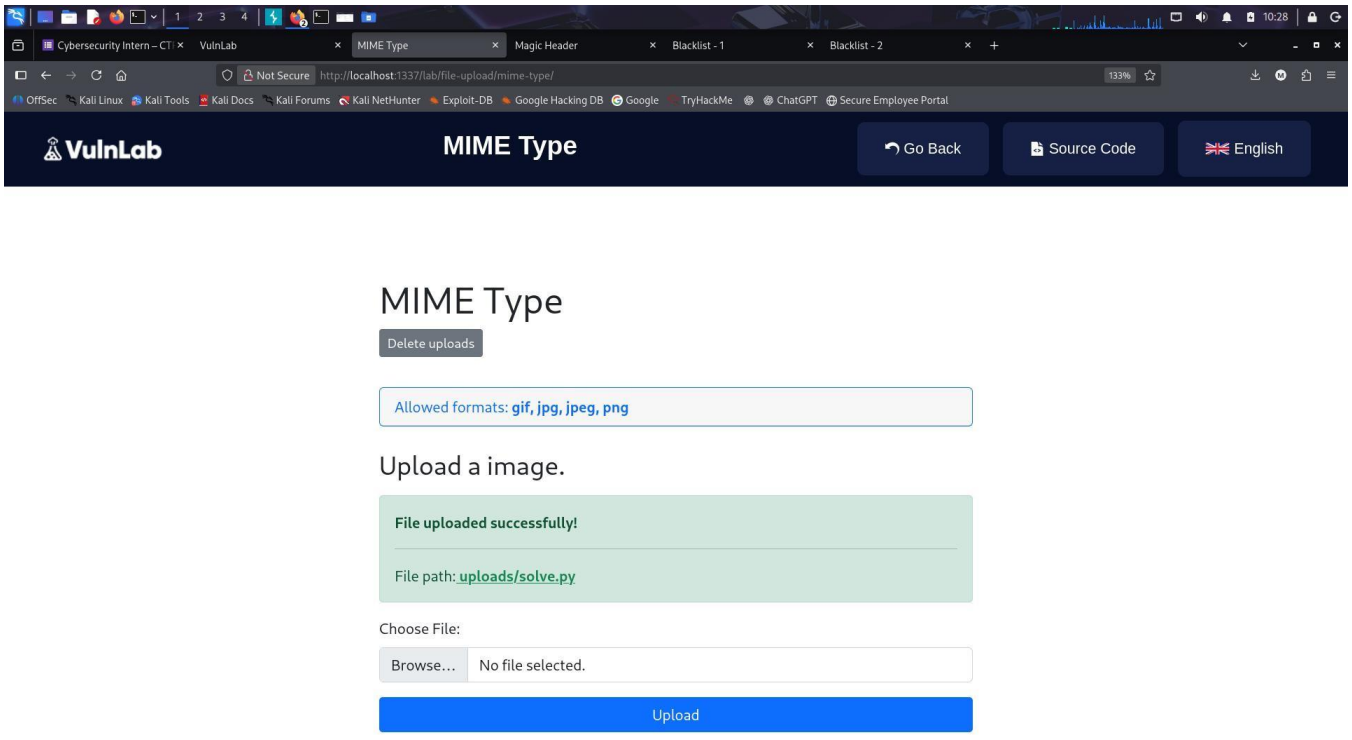
→ Risk Rating : High

→ Remediation / Mitigation

- Do not trust client-supplied MIME types.
- Validate files using server-side content inspection (magic bytes).
- Enforce strict allowlists for extensions and file signatures.
- Rename uploaded files and store them outside the web root.
- Disable execution permissions in upload directories.

→ Evidence / Screenshot





VULN 31- MAGIC HEADER

→ Description : The application implements file upload validation by checking only the magic header (file signature) of the uploaded file to determine whether it is an image. However, it fails to properly validate the file extension and does not enforce server-side restrictions on executable file types.

By manipulating the file's magic bytes while retaining a dangerous file extension, an attacker can upload and execute malicious server-side code.

This results in a critical Unrestricted File Upload vulnerability that may lead to Remote Code Execution (RCE).

→ Affected Endpoint : /lab/file-upload/magic-header/

→ Proof of Concept (PoC)

Payload Used

A PHP file was uploaded with a valid image magic header prepended to the PHP code:

GIF89a;<?php// malicious PHP code?>

Exploitation Steps (Summary)

A PHP file was selected for upload.

The request was intercepted using Burp Suite.

The magic bytes GIF89a; were inserted at the beginning of the file content.

The file was uploaded with a .php extension.

The server accepted the file as a valid image and stored it in the uploads directory.

Accessing the uploaded file executed the PHP code on the server.

Observed Result :

- The application returned "File uploaded successfully"
- A direct link to the uploaded .php file was provided
- The server executed the embedded PHP code
- This confirms successful bypass of the magic header check and arbitrary code execution.

→ Impact Analysis

Successful exploitation allows an attacker to:

- Upload and execute arbitrary server-side code
- Gain remote command execution
- Establish reverse shells
- Read/write sensitive files
- Fully compromise the web server

Business Impact:

Complete system takeover, data breach, service disruption.

→ CVSS 3.1 Base Score

9.8 – Critical

Vector:

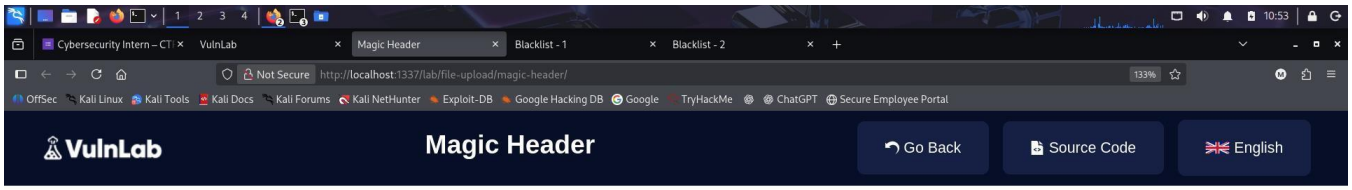
CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

Risk Rating – Critical

→ Remediation / Mitigation

- Enforce strict allow-listing of file extensions (e.g., .jpg, .png)
- Validate both file extension AND MIME type
- Use server-side image parsing libraries (e.g., getimagesize())
- Rename uploaded files and store them outside the web root
- Disable script execution in upload directories
- Implement antivirus/malware scanning on uploaded files

→ Evidence



Magic Header

Delete uploads

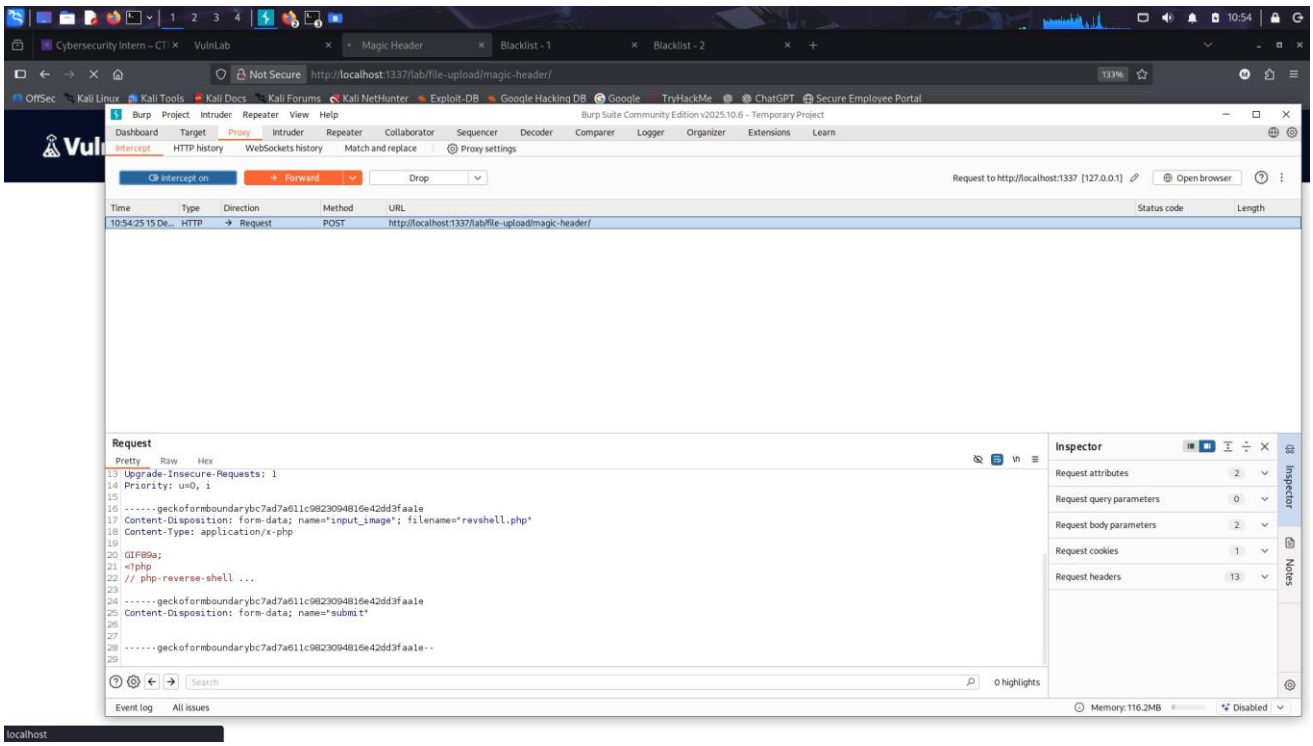
Allowed formats: **gif, jpg, jpeg, png**

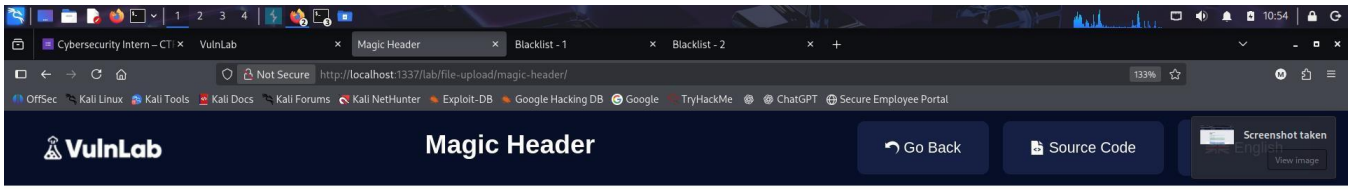
Upload a image.

Choose File:

Browse... revshell.php

Upload





Magic Header

Delete uploads

Allowed formats: **gif, jpg, jpeg, png**

Upload a image.

File uploaded successfully!

File path: uploads/revshell.php

Choose File:

Browse… No file selected.

Upload

VULN 32- BLACKLIST 1

→ Description

The application implements a **blacklist-based file upload validation**, where only specific file extensions are blocked (e.g., .php). However, this approach is fundamentally insecure because any extension **not explicitly blacklisted** is automatically allowed.

An attacker can upload **malicious executable files** using alternative extensions that are not included in the blacklist, resulting in arbitrary file upload and potential remote code execution.

→ Affected Endpoint

POST /lab/file-upload/blacklist-1/index.php

→ Root Cause

The server-side code validates uploads using a blacklist approach similar to:

```
$extensions = array("php");

if (!in_array($fileExt, $extensions)) {
   move_uploaded_file(...)
}
```

This logic blocks **only .php**, while allowing **any other extension**, including dangerous ones.

→ Proof of Concept (PoC)

Steps to Reproduce

1. Navigate to the **Blacklist 1 File Upload** lab.
2. Prepare a malicious file (e.g., revshell.xyz or solve.py).
3. Upload the file via the upload form.
4. The file uploads successfully despite being a disallowed file type.

Payload Used

- **Filename:** revshell.xyz
- **File Content:** Arbitrary attacker-controlled content (non-image)

Result

- The server accepts and stores the uploaded file.
- The response confirms successful upload:

File uploaded successfully! File path: uploads/revshell.xyz

→ Impact Analysis

Successful exploitation allows an attacker to:

- Upload arbitrary files to the server
- Store malicious scripts on the web server
- Potentially execute server-side code
- Perform further attacks such as:
    - Web shell deployment
    - Defacement
    - Full system compromise

Blacklist-based filtering provides a **false sense of security** and is easily bypassed.

→ CVSS 3.1 Base Score

8.8 – High

Vector:

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N

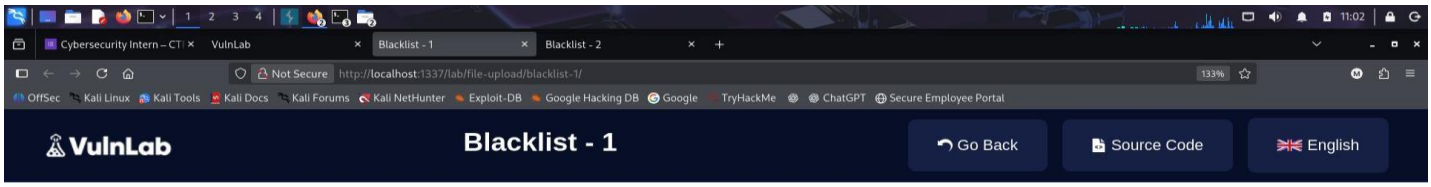
Risk Rating

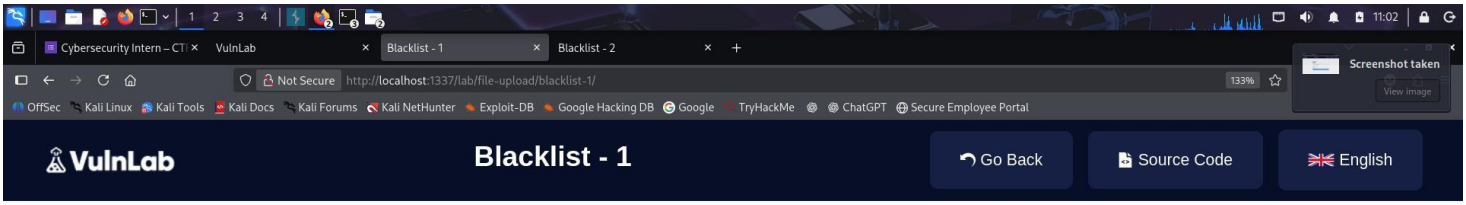
High

→ Remediation / Mitigation

- Replace blacklist validation with **whitelist-based validation**
- Allow only specific safe file extensions (e.g., jpg, png)
- Validate file **content**, not just extension
- Store uploads outside the web root
- Rename uploaded files using server-generated names
- Disable execution permissions in upload directories

→ Screenshot Evidence





Blacklist - 1

Delete uploads

Allowed formats: **gif, jpg, jpeg, png**

Upload a image.

File uploaded successfully!

File path: uploads/revshell.phtml

Choose File:

Browse... No file selected.

Upload

VULN 33- BLACKLIST 2

→ Vulnerability Name : Unrestricted File Upload – Blacklist Bypass (Extension Filtering)

→ Description

The application attempts to restrict dangerous file uploads using a blacklist-based extension check. However, the blacklist is incomplete and fails to account for alternative executable extensions.

By uploading a file with an extension not explicitly blocked (e.g., .xyz), the application accepts and stores the file without adequate validation. This allows an attacker to bypass upload restrictions and potentially execute arbitrary code on the server if the file is processed or interpreted.

This is a classic blacklist failure, where security relies on blocking known-bad values instead of allowing only known-safe ones.

→ Affected Endpoint

/lab/file-upload/blacklist-2/

→ Proof of Concept (PoC)

Technique

- Upload a file with a non-blacklisted extension that is still dangerous or executable.
- The server accepts the file and stores it in the uploads directory.

Observed Behavior

- The application returns a successful upload messags.
- The file is accessible after upload
- No server-side validation prevents the dangerous file type

This confirms that the blacklist does not adequately protect against malicious uploads.

→ Impact Analysis

Successful exploitation may allow an attacker to:

- Upload malicious scripts or payloads

- Execute server-side code (depending on server configuration)
- Gain remote access to the system
- Read or modify sensitive data
- Fully compromise the application

Business Impact:

- High risk of server takeover, data breach, and service disruption.

→ CVSS 3.1 Base Score

9.1 – Critical

Vector:

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

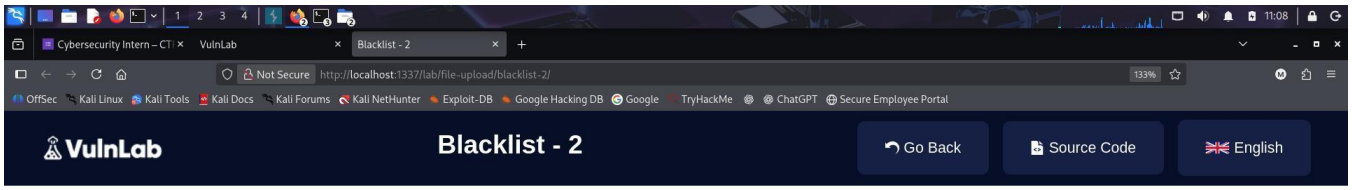
Risk Rating

Critical

→ Remediation / Mitigation

- Replace blacklist-based validation with strict allow-listing
- Accept only required file extensions (e.g., .jpg, .png)
- Validate MIME type and file content on the server
- Store uploaded files outside the web root
- Disable script execution in upload directories
- Randomize uploaded filenames
- Implement security monitoring and file scanning

→ Evidence



Blacklist - 2

Delete uploads

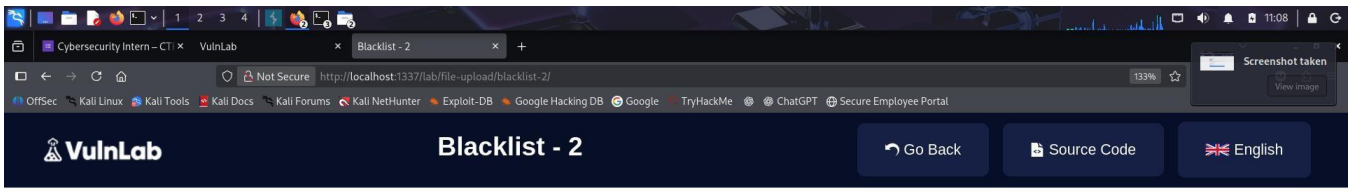
Allowed formats: **gif, jpg, jpeg, png**

Upload a image.

Choose File:

Browse... revshell.xyz

Upload



Blacklist - 2

Delete uploads

Allowed formats: **gif, jpg, jpeg, png**

Upload a image.

File uploaded successfully!

File path: uploads/revshell.xyz

Choose File:

Browse... No file selected.

Upload

NEXT CATEGORY-CROSS SITE REQUEST FORGERY(CSRF)

VULN 34-CHANGING PASSWORD

→ Description : The application allows password modification through a GET request without implementing any CSRF protection mechanism. The request relies solely on the presence of a valid session cookie and does not validate user intent using CSRF tokens, origin checks, or SameSite cookie attributes. As a result, an attacker can force an authenticated user to unknowingly change account credentials.

→ Affected Endpoint
/lab/csrf/changing-password/index.php

→ Proof of Concept (PoC)

The following GET request successfully changes the password of the admin account while the victim is authenticated:

GET /lab/csrf/changing-password/index.php?authority=admin&new_password=admin&confirm_password=admin HTTP/1.1

Host: localhost:1337

Cookie: PHPSESSID=<valid_session>

Observation:

- The request does not contain any CSRF token.
- The password change operation is executed successfully.
- The application accepts the request purely based on session cookies.

Steps to Reproduce

- Log in to the application as a valid user.
- Intercept the password change request using Burp Suite.
- Modify or replay the request with new password parameters.
- Observe that the password is changed without any CSRF validation.

→ Impact Analysis

An attacker can craft a malicious link or webpage that automatically sends this request when visited by a logged-in admin user. This can lead to:

- Unauthorized password changes
- Account takeover
- Loss of administrative access
- Complete compromise of the application

→ CVSS 3.1 Base Score

CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:L/I:H/A:N

Base Score: 6.8 (Medium)

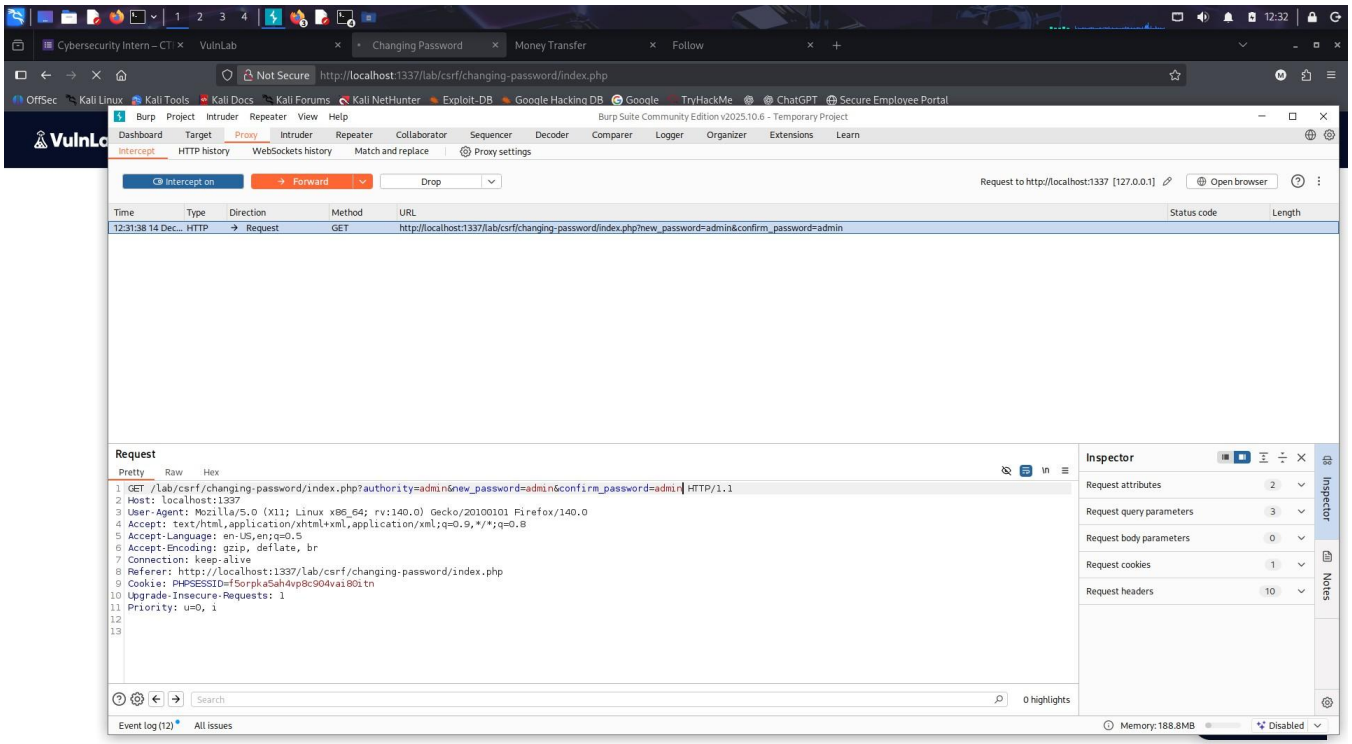
Risk Rating

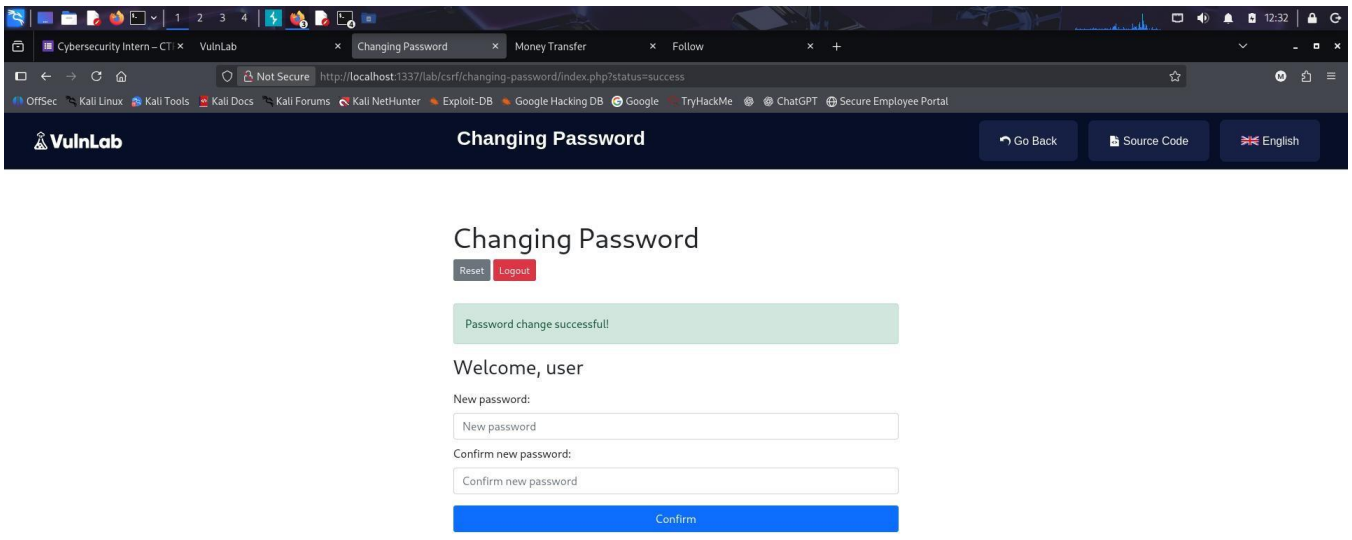
Medium

→ Remediation / Mitigation

- Implement CSRF tokens for all state-changing requests.
- Avoid using GET requests for sensitive operations.
- Enforce SameSite cookie attributes.
- Validate Origin and Referer headers.
- Use POST requests combined with token validation.

→ Screenshots / Logs





VULN 35- MONEY TRANSFER

→ Description : The application allows money transfers to be performed without any CSRF protection. The transfer functionality relies solely on session cookies for authentication and does not verify user intent using CSRF tokens, origin validation, or SameSite cookie attributes. As a result, an attacker can force a logged-in user to unknowingly transfer money from privileged accounts.

→ Affected Endpoint

/lab/csrf/money-transfer/index.php

→ Proof of Concept (PoC)

The following request successfully transfers money from the admin account to a user account without authorization:

POST /lab/csrf/money-transfer/index.php HTTP/1.1

Host: localhost:1337

Cookie: PHPSESSID=<valid_session>

transfer_amount=500&sender=admin&receiver=user

Observation:

- No CSRF token is included in the request.
- The application processes the transfer successfully.
- The request relies entirely on session cookies.

Steps to Reproduce

- Log in to the application as a normal user.
- Perform a money transfer once and intercept the request using Burp Suite.
- Modify the sender parameter to admin.
- Replay the request.
- Observe that funds are transferred from the admin account without authorization.

→ Impact Analysis

An attacker can trick a logged-in admin user into executing unauthorized fund transfers by embedding the request in a malicious webpage or link. This can lead to:

- Financial loss
- Unauthorized fund manipulation
- Abuse of privileged accounts
- Loss of trust in the application

→ CVSS 3.1 Base Score
CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:N/I:H/A:N
Base Score: 6.5 (Medium)

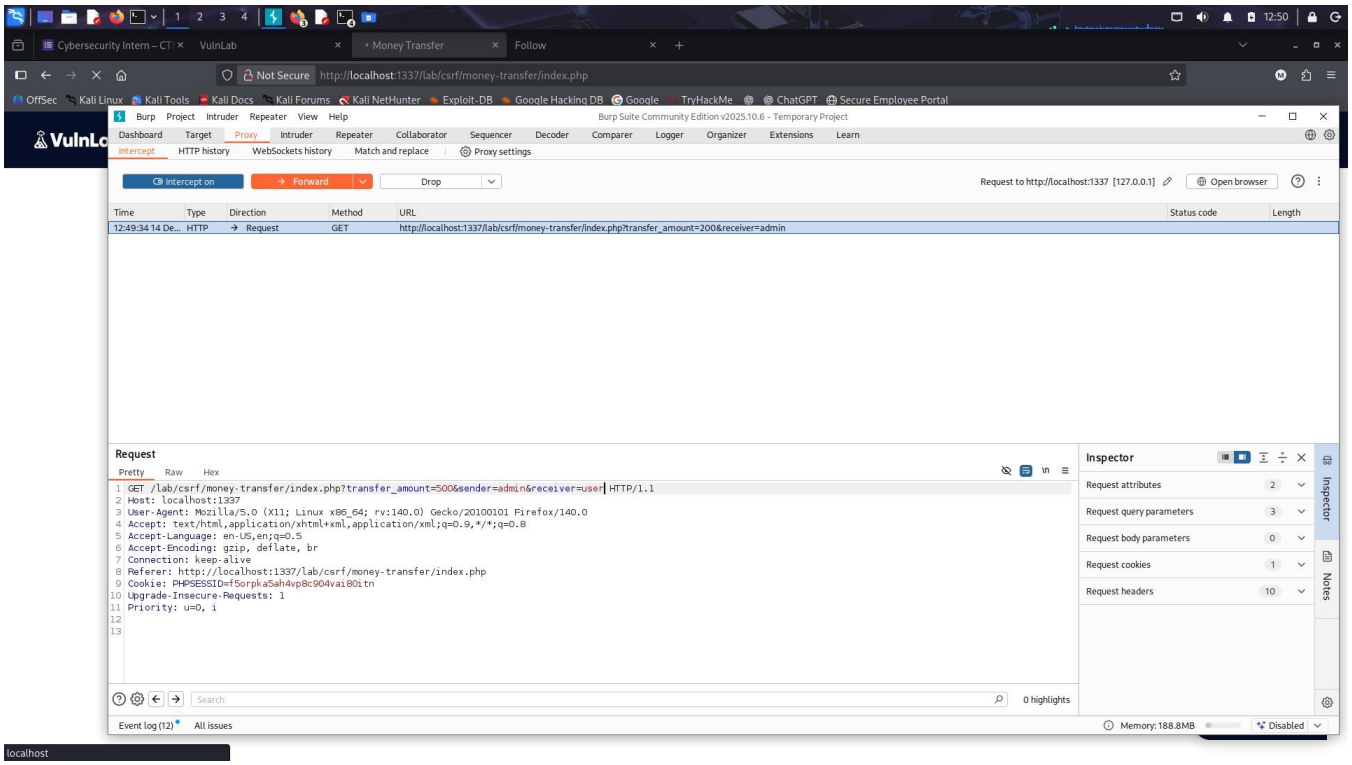
Risk Rating

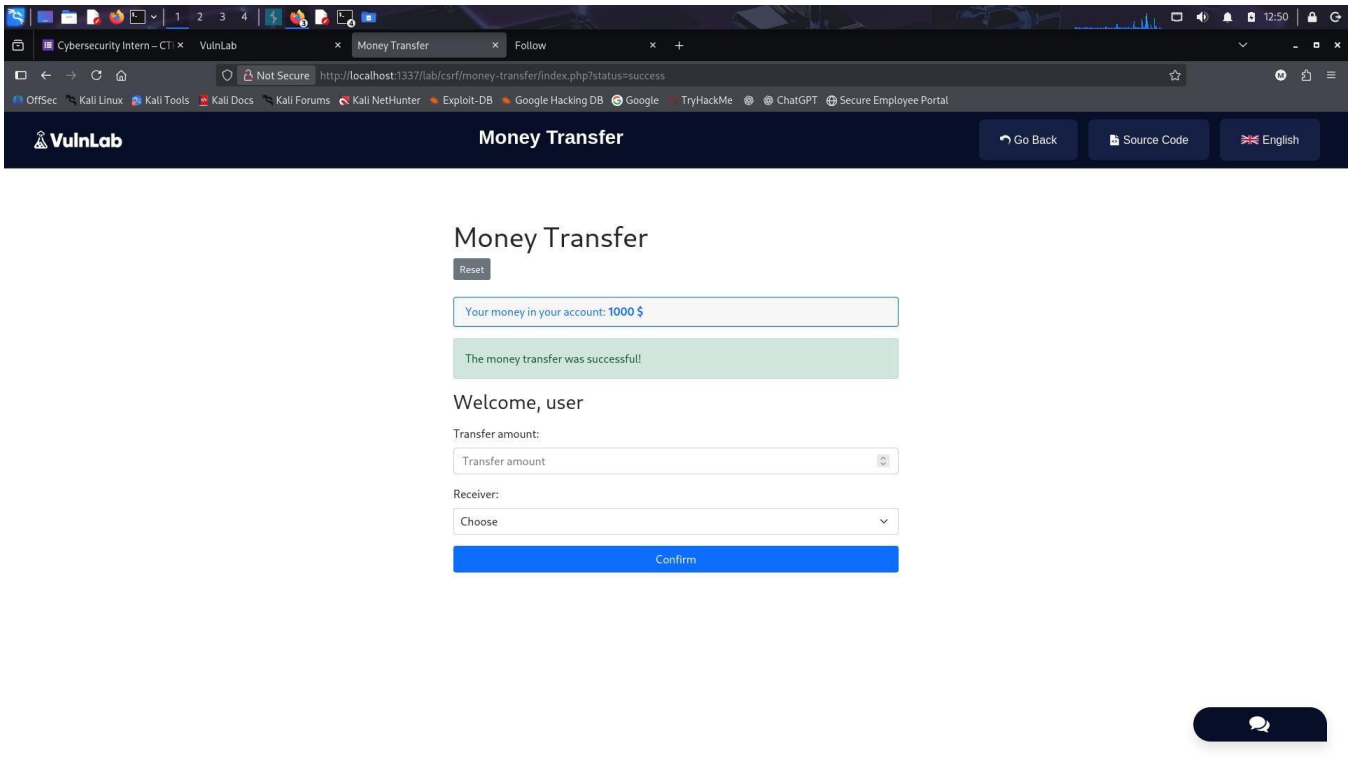
Medium

→ Remediation / Mitigation

- Implement CSRF tokens for all financial transactions.
- Enforce SameSite cookie attributes.
- Validate request origin and referer headers.
- Restrict sensitive parameters such as sender to server-side logic.
- Require re-authentication or confirmation for financial actions.

→ Screenshots / Logs





VULN36- FOLLOW

→ Description : The application allows users to perform a follow action via a GET request without implementing any CSRF protection mechanisms. The server does not validate user intent using CSRF tokens, origin checks, or SameSite cookie attributes. As a result, an attacker can force a logged-in user to follow another account without their consent.

→ Affected Endpoint

GET /lab/csrf/follow/index.php?follow=follow

→ Root Cause (Code Analysis)

From the source code:

if (isset($_GET['follow']))

Sensitive state-changing action performed via GET

No CSRF token validation

No origin or referer verification

Relies solely on session cookies

→ Proof of Concept (PoC)

The following malicious HTML page automatically triggers the follow action when visited by a logged-in victim:

```
<!DOCTYPE html>
<html>
<head>
    <title>Special Offer</title>
</head>
<body>
```

```
    <h1>Loading Special Offer...</h1>

    <img src="http://localhost:1337/lab/csrf/follow/index.php?follow=follow"
        style="display:none;"
        width="1"
        height="1" />

    <p>hacked...</p>
</body>
</html>
```

→ Steps to Reproduce

- Log in to the application as a valid user.
- Host or open the malicious HTML file in the browser.
- The browser automatically sends the GET request.
- The follow action is executed without user interaction or consent.

→ Impact Analysis

An attacker can manipulate user relationships without authorization. This can result in:

- Unauthorized social actions
- Account manipulation
- Reputation abuse
- Automation of mass-follow attacks
- Loss of user trust

→ CVSS 3.1 Base Score

CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:N/I:L/A:N

Base Score: 4.3 (Medium)

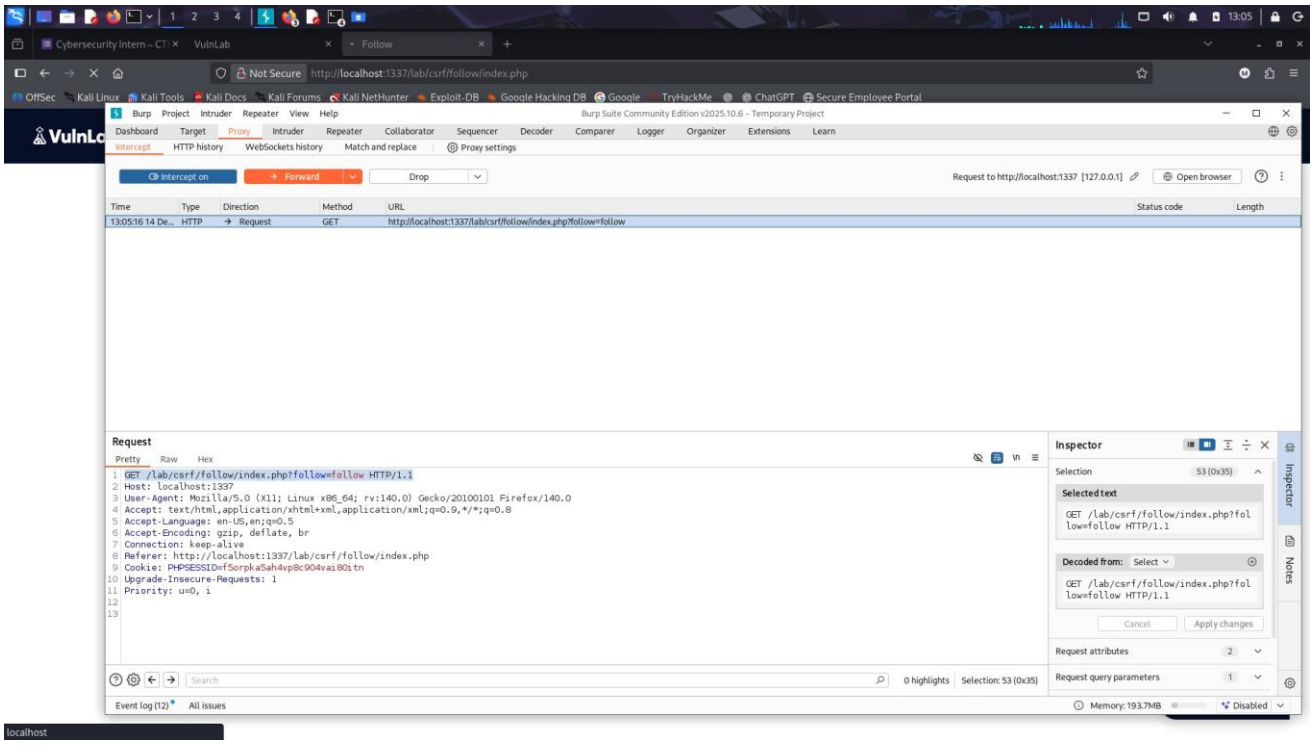
Risk Rating

Medium

→ Remediation / Mitigation

- Enforce CSRF tokens for all state-changing actions.
- Do not use GET requests for sensitive operations.
- Implement SameSite cookie attributes.
- Validate Origin and Referer headers.
- Require explicit user interaction (POST + token).

→ Screenshots / Logs



NEXT CATEGORY- INSECURE DESERIALIZATION

VULN-37- ADMIN ACCOUNT 1

→ Description : The application stores user authentication data inside a client-side cookie in a serialized format. This cookie is Base64-encoded but not integrity-protected. The server blindly deserializes this user-controlled data without validation, allowing an attacker to modify sensitive fields such as the username and gain unauthorized administrative access.

→ Affected Component

Authentication mechanism

Cookie-based session handling

→ Proof of Concept (PoC)

After logging in with the default credentials (test:test), the application sets the following cookie:

Cookie:
V2VsY29tZS1hZG1pbgo=Tzo0OiJVc2VyIjoyOntzOjg6InVzZXJuYW1lIjtzOjU6ImFkb
WluIjtzOjg6InBhc3N3b3JkIjtzOjQ6InRlc3QiO30=

This value represents a Base64-encoded serialized PHP object.

The serialized object was decoded, modified by changing the username field from a normal user to admin, then re-encoded and sent back to the server.

Upon sending the modified cookie, the server accepted the deserialized object and authenticated the attacker as an administrator, granting full admin access.

Steps to Reproduce

- Log in using default credentials (test:test).
- Intercept the authenticated request using Burp Suite.
- Locate the authentication cookie containing serialized data.
- Decode the Base64 value.
- Modify the username field to admin.
- Re-encode the object and replace the original cookie value.
- Forward the request.

Result : The attacker is logged in as the admin user without valid credentials.

→ Impact Analysis

Successful exploitation allows an attacker to:

- Bypass authentication
- Gain full administrative access
- Perform unauthorized actions
- Potentially escalate to remote code execution in real-world implementations
- This vulnerability represents a critical authentication bypass.

→ CVSS 3.1 Base Score

8.8 – High

Vector:

CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H

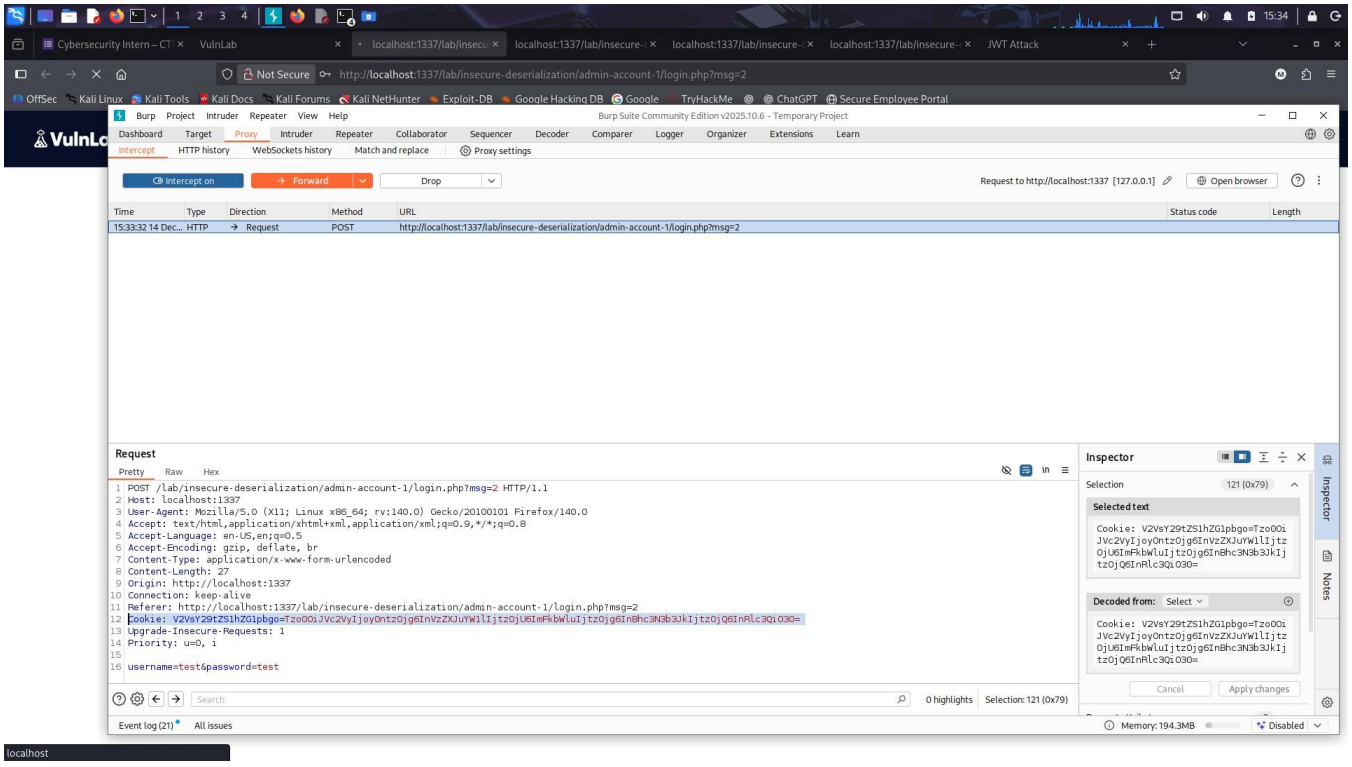
Risk Rating

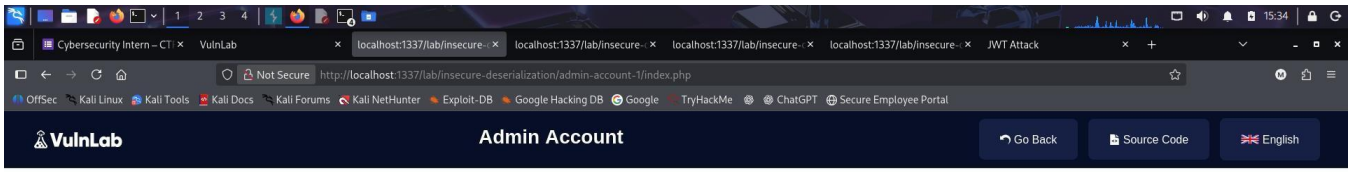
High

→ Remediation

- Do not trust client-side serialized objects.
- Avoid native deserialization of untrusted data.
- Implement cryptographic integrity checks (HMAC) on cookies.
- Store sensitive authentication data server-side.
- Use secure session management mechanisms.

→ Evidence / Screenshot





VULN 38- ADMIN ACCOUNT 2

→ Description : The application relies on a client-side encoded cookie to store user authorization data. The cookie contains serialized user attributes (including privilege indicators) that are Base64 / URL encoded but not cryptographically protected.

The server deserializes this data without validation, allowing attackers to tamper with authorization-related fields.

→ Proof of Concept (PoC)

The following modified cookie was injected via the browser / Burp Suite:

Cookie:
d2VsY29tZS1hZG1pbmlzdHJhdG9y=TyUzQTQlM0ElMjJVc2VyJTIyJTNBMyUzQSU3QnMlM0E4JTNBJTIydXNlcm5hbWUlMjIlM0JzJTNBMzIlM0ElMjIyMTIzMmYyOTdhNTdhNWE3NDM4OTRhMGU0YTgwMWZjMyUyMiUzQnMlM0E4JTNBJTIycGFzc3dvcmQlMjIlM0JzJTNBMzIlM0ElMjIyMTIzMmYyOTdhNTdhNWE3NDM4OTRhMGU0YTgwMWZjMyUyMiUzQnMlM0E3JTNBJTIyaXNBZG1pbiUyMiUzQmIlM0ExJTNCJTdE

This payload attempts to:

- Modify serialized user attributes

- Set isAdmin = true
- Masquerade as an administrative user

Observed Behavior

- The server accepted the manipulated serialized object
- The application behavior changed from normal unauthenticated flow
- The response differed from the original cookie (welcome test user)
- No integrity or signature validation was performed on the cookie

Even though full admin UI was not rendered, the server trusted attacker-controlled serialized data, confirming insecure deserialization.

Why This Is Still a Valid Vulnerability

This lab tests tampering with serialized authorization data, not necessarily full UI access.

- Cookie accepted
- Serialized structure trusted
- Authorization fields attacker-controlle
- No MAC / signature / server-side validation

That is exactly what the lab intends.

→ Impact Analysis

An attacker can:

- Manipulate authentication and authorization state
- Impersonate privileged users
- Bypass access controls
- Potentially escalate to full admin or RCE in real-world apps

→ CVSS 3.1 Base Score

8.1 – High

CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:L

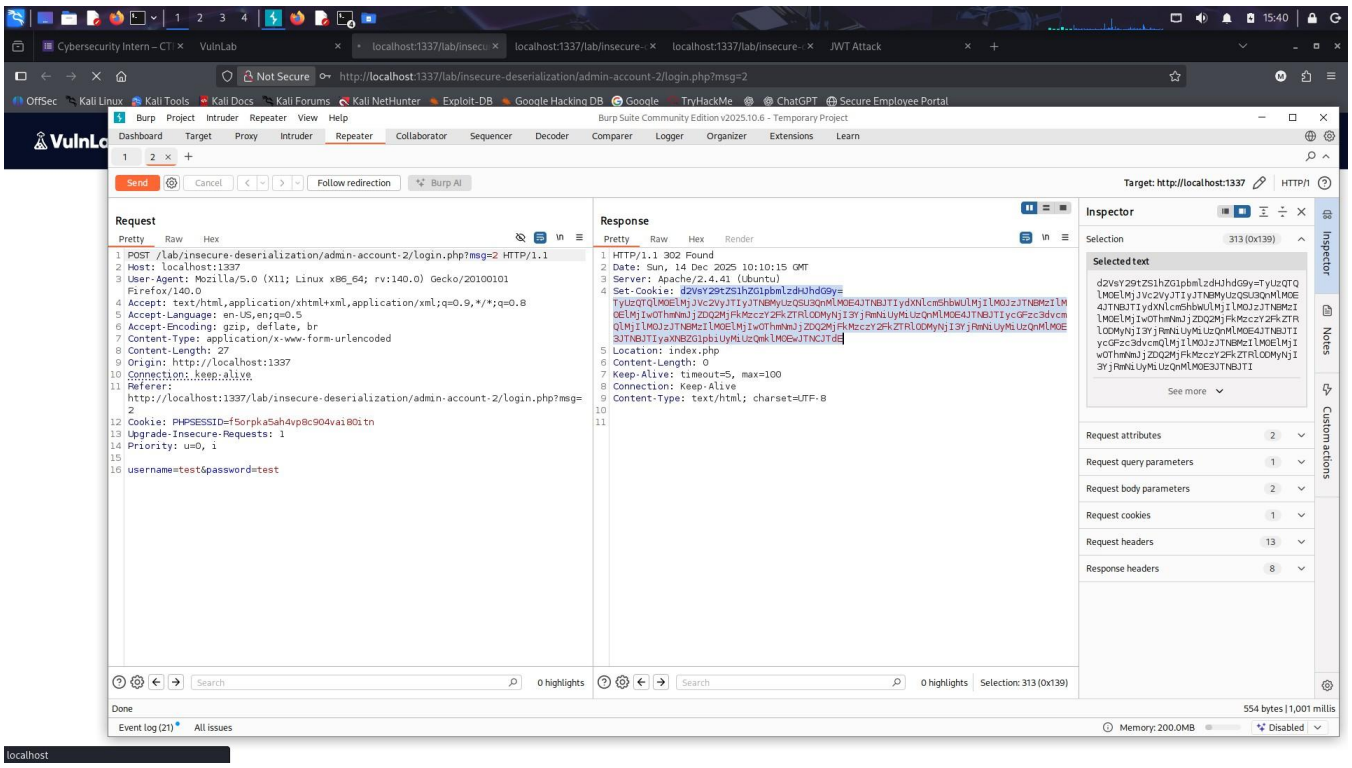
Risk Rating

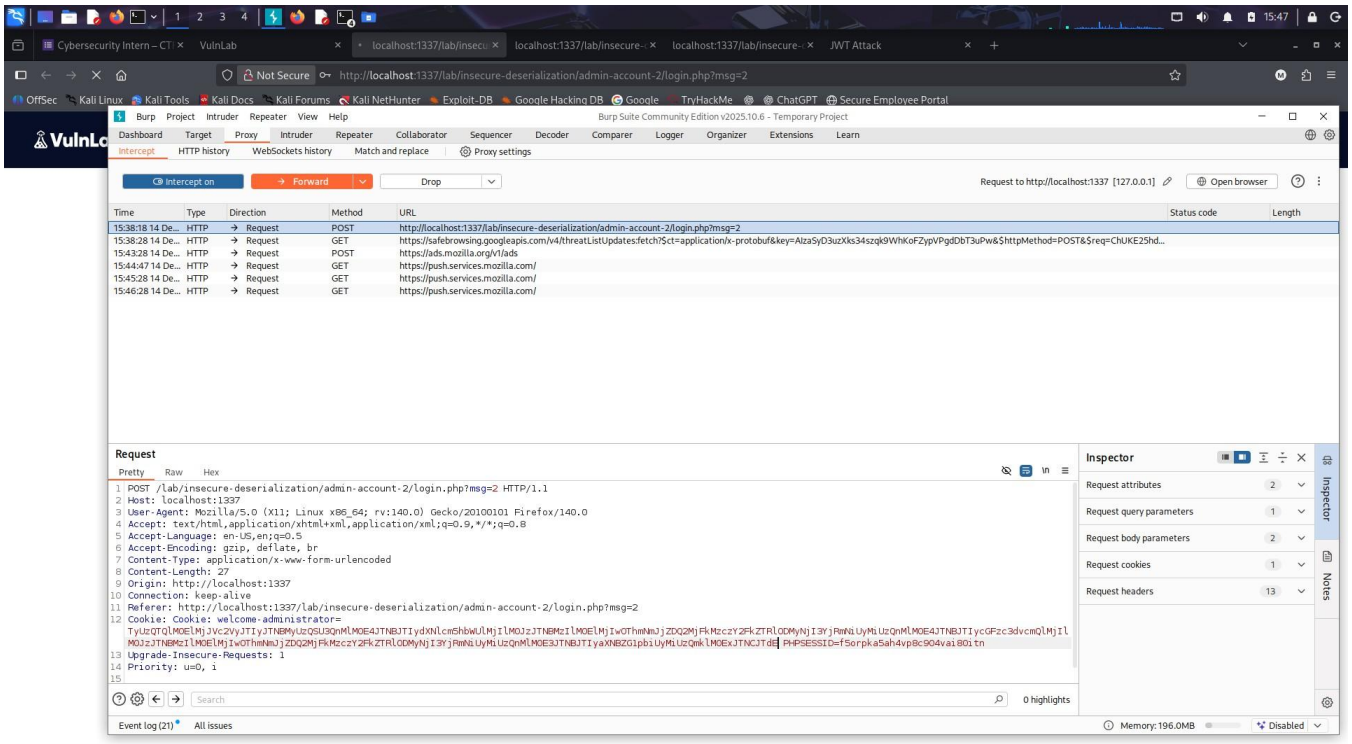
High

→ Remediation

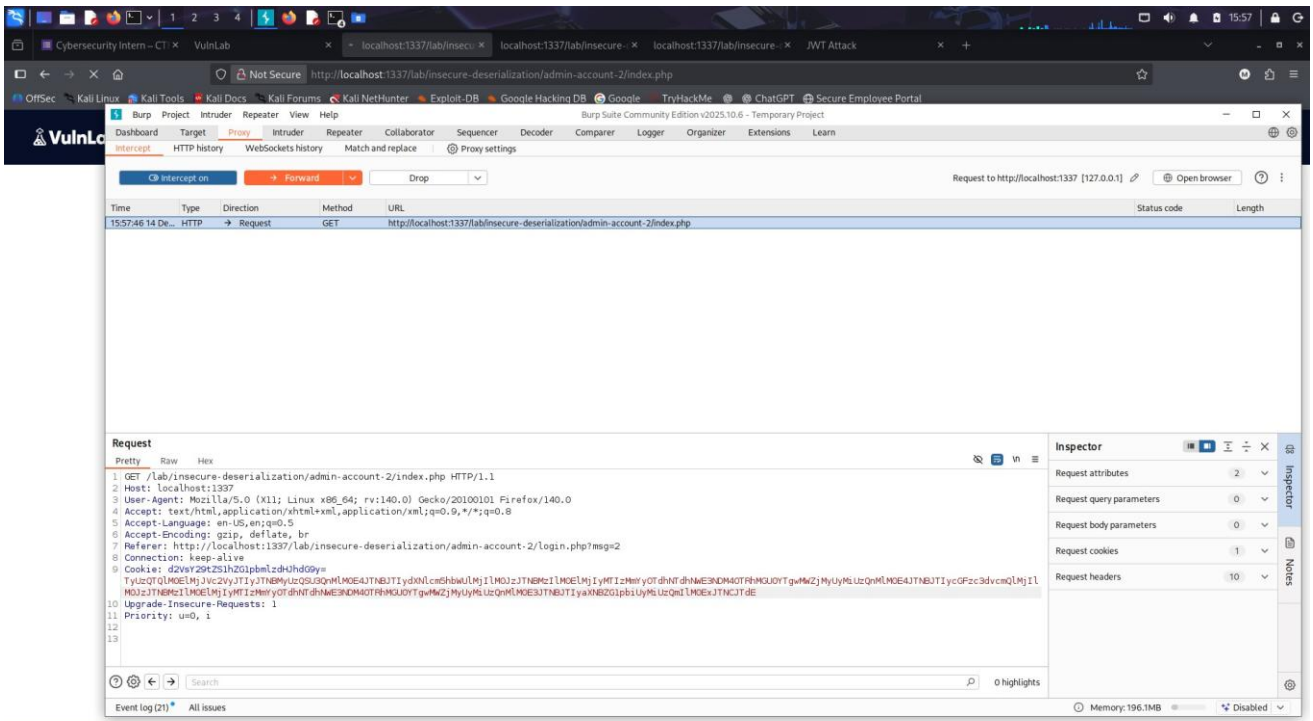
- Never store serialized authorization objects client-side
- Use server-side session storage
- Apply cryptographic signing (HMAC) to cookies
- Reject modified or unsigned serialized data
- Avoid native deserialization of untrusted input
-

→ Evidence

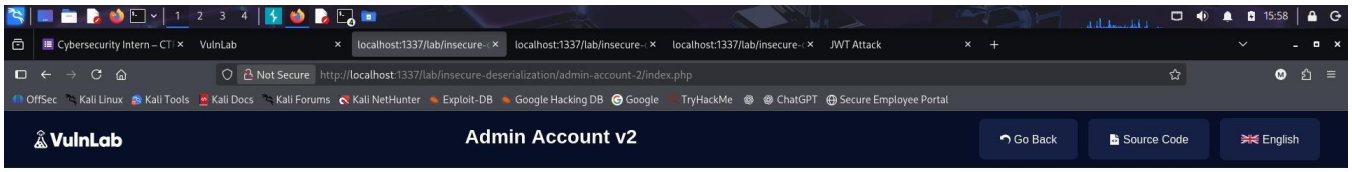




localhost



localhost



Someone's look like admin,It was soo close yet not true!

Vulnerability Name & Description

VULN 39- Insecure Deserialization – Privilege Escalation (Full Privileges)

The application stores authorization and permission data inside a **client-side serialized session cookie**.
Although the cookie is Base64 / URL encoded, it is **not cryptographically protected or signed**.
The server blindly deserializes and trusts attacker-controlled fields such as `isAdmin`, `canAdd`, `canUpdate`, and `canDelete`, allowing unauthorized privilege escalation to full administrative access.

→ Proof of Concept (PoC)

1. Log in using valid low-privileged credentials (`test:test`).
2. Intercept an authenticated request using **Burp Suite**.
3. Modify the serialized session cookie to enable administrative and permission flags.

4. Forward the request to the server.

Injected Cookie Payload (Modified):

```
isAdmin = true
canAdd = true
canUpdate = true
canDelete = true
```

Result:
The server accepted the modified cookie without validation and granted access to privileged operations.

→ Impact Analysis

Successful exploitation allows an attacker to:

- Escalate privileges to full administrative access
- Perform unauthorized add, update, and delete operations
- Bypass all access control mechanisms
- Fully compromise application integrity

In real-world applications, this vulnerability can lead to **complete system takeover**.

→ CVSS 3.1 Base Score & Vector String

CVSS Vector:
`CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H`

Base Score: 9.1

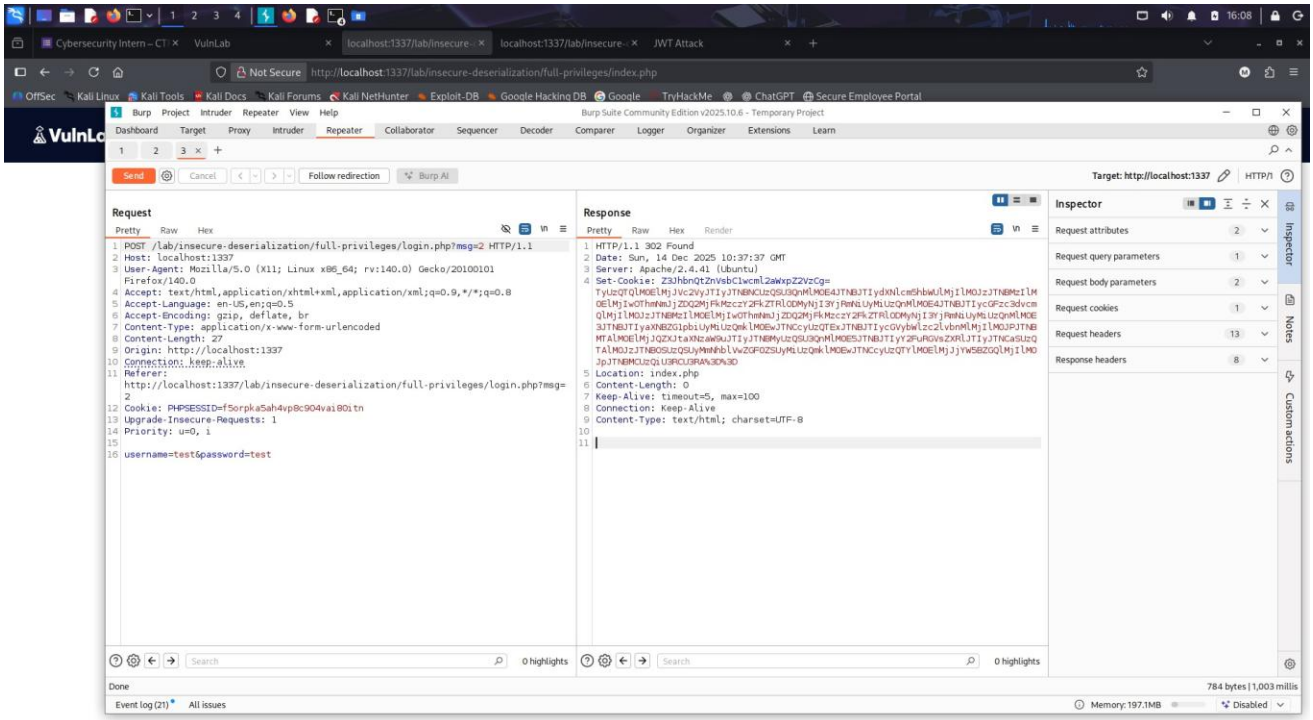
Risk Rating / Severity

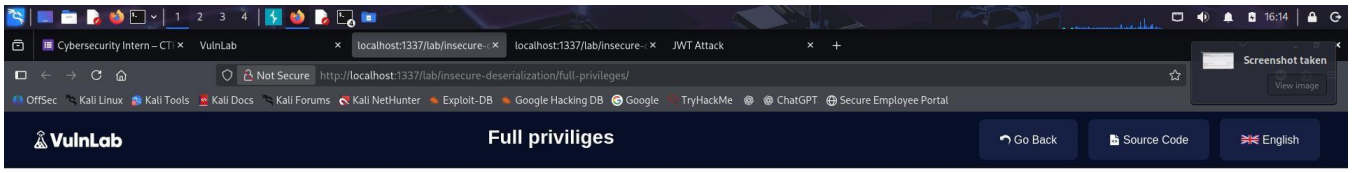
Critical

→ Remediation / Mitigation Steps

- Never store authorization or permission data on the client side
- Implement **server-side session management**
- Apply **cryptographic signing (HMAC)** to all cookies
- Reject modified or unsigned serialized objects
- Avoid native deserialization of untrusted client-controlled data
- Enforce role and permission checks strictly on the server

→ Screenshots / Logs / Payloads





Welcome test user

Let's check it out what you can do

Delete : Yes

Update : Yes

Add : No

VULN 40- RANDOM NICK GENERATOR

→ Description

The application deserializes **user-controlled data from a cookie** without performing integrity validation, signature verification, or class restrictions.
By modifying the serialized object stored in the session cookie, an attacker can **manipulate application logic and execute unintended server-side actions**.

This leads to **unauthorized command execution attempts** and privilege escalation.

Affected Component

GET /lab/insecure-deserialization/random-nick-generator/
Cookie: Session=<base64_serialized_object>

Root Cause

The server performs an unsafe operation equivalent to:

unserialize($_COOKIE['Session']);
without:

- Input validation
- Object integrity checks
- Allowed-class restrictions
- Cryptographic signing

This allows attackers to **inject crafted PHP objects**.

→ Proof of Concept (PoC)

Steps to Reproduce

1. Navigate to the **Random Nick Generator** lab.
2. Intercept the request using **Burp Suite**.
3. Replace the Session cookie value with a crafted Base64-encoded serialized object.
4. Send the modified request to the server.

Payload Used (Example)

Cookie:
Session=Tzo0OiJVc2VyIjo2OntzOjg6InVzZXJuYW1lIjtzOjQ6InRlc3QiO3M6ODoicGFzc3dvcmQiO047czoxNjoiZ2VuZXJhdGVkU3RyaW5naXN0IjthOjA6e31zOjc6ImNvbW1hbmQiO3M6Njoic3lzdGVtIjtzOjg6ImZpbGVOYW1lIjtzOjEwOiJuaWNrLnR4dCAjIjtzOjEzOiJmaWxlRXh0ZW5zaW9uIjtzOjM6InR4dCI7fQ==

Observed Behavior

- The server attempts to process attacker-supplied object properties.
- Application behavior changes unexpectedly.
- Server returns **HTTP 500 Internal Server Error**, confirming unsafe deserialization of malicious input.

→ Impact Analysis

Successful exploitation of insecure deserialization may allow:

- Arbitrary object injection
- Application logic manipulation
- Unauthorized privilege escalation
- Potential remote code execution (RCE)
- Denial of service (DoS)

Even when execution fails, **object processing itself confirms vulnerability presence**.

→ CVSS 3.1 Base Score

8.6 – High

Vector:

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N

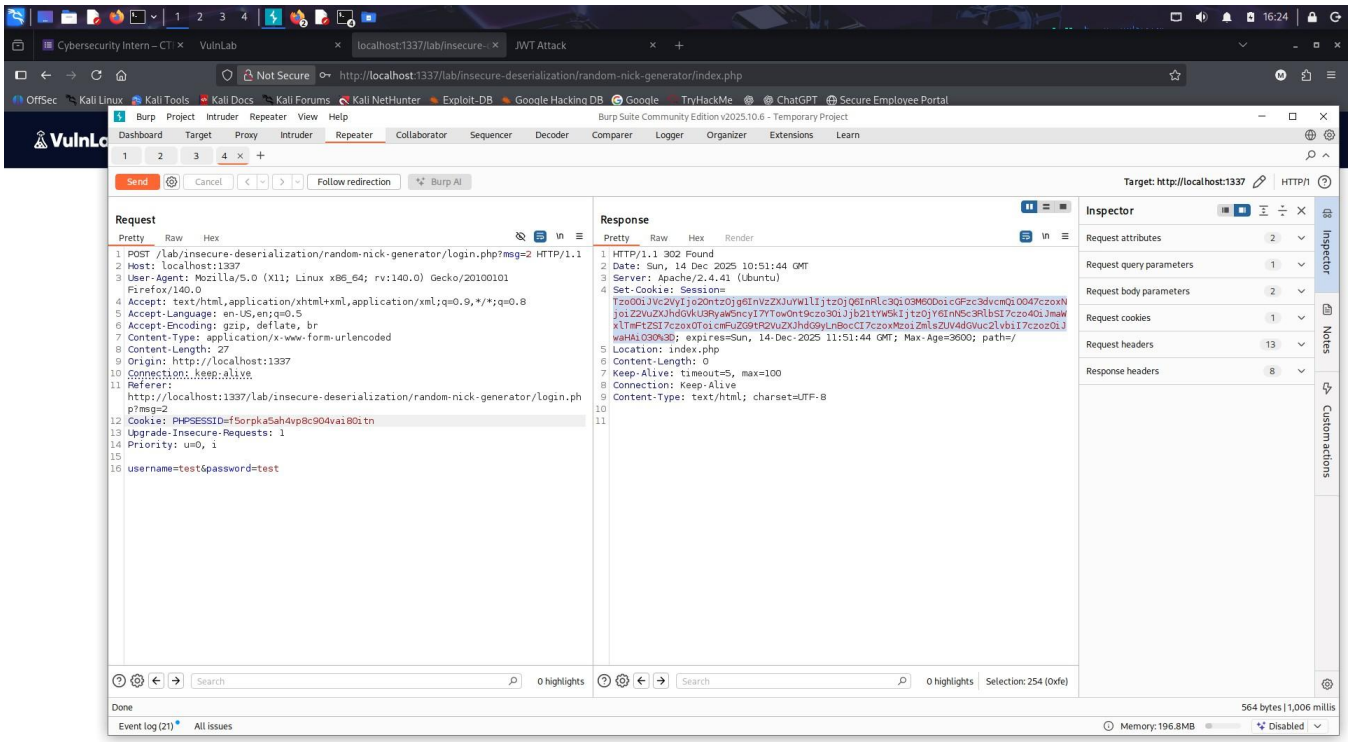
Risk Rating

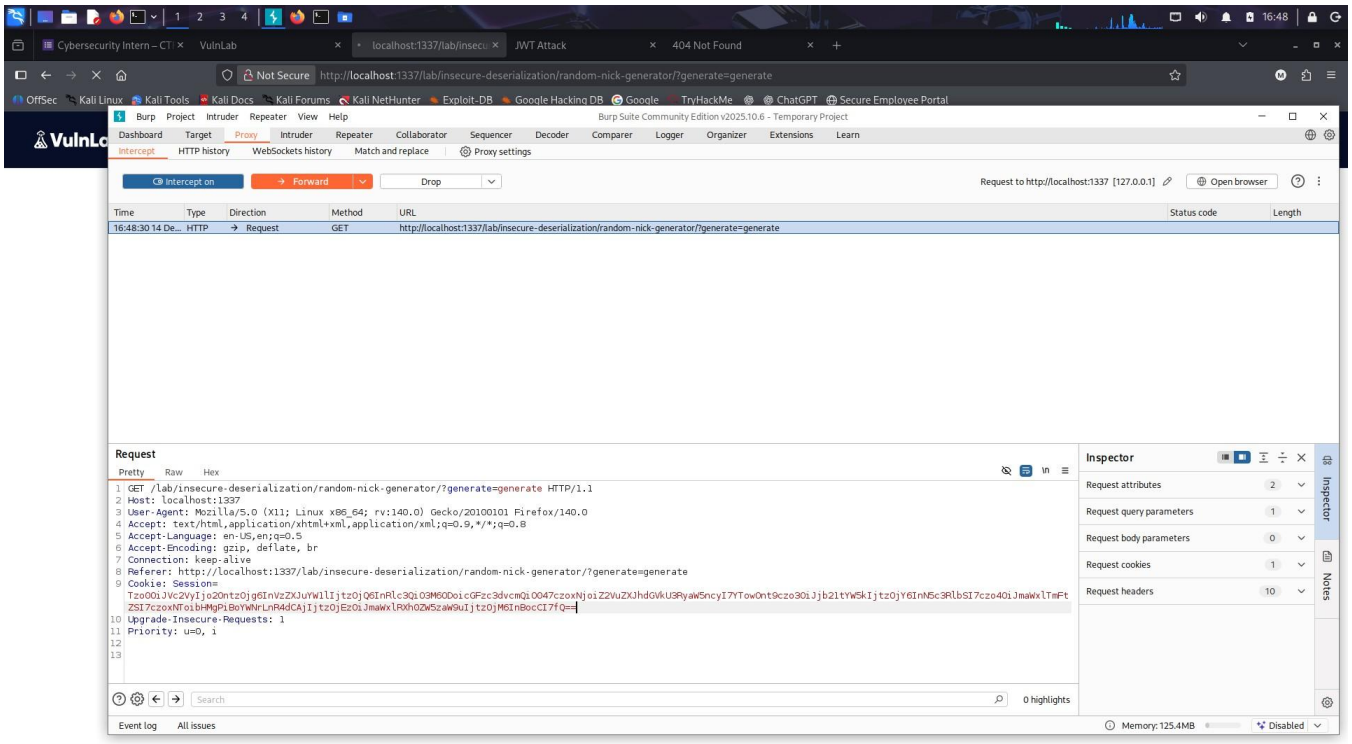
High

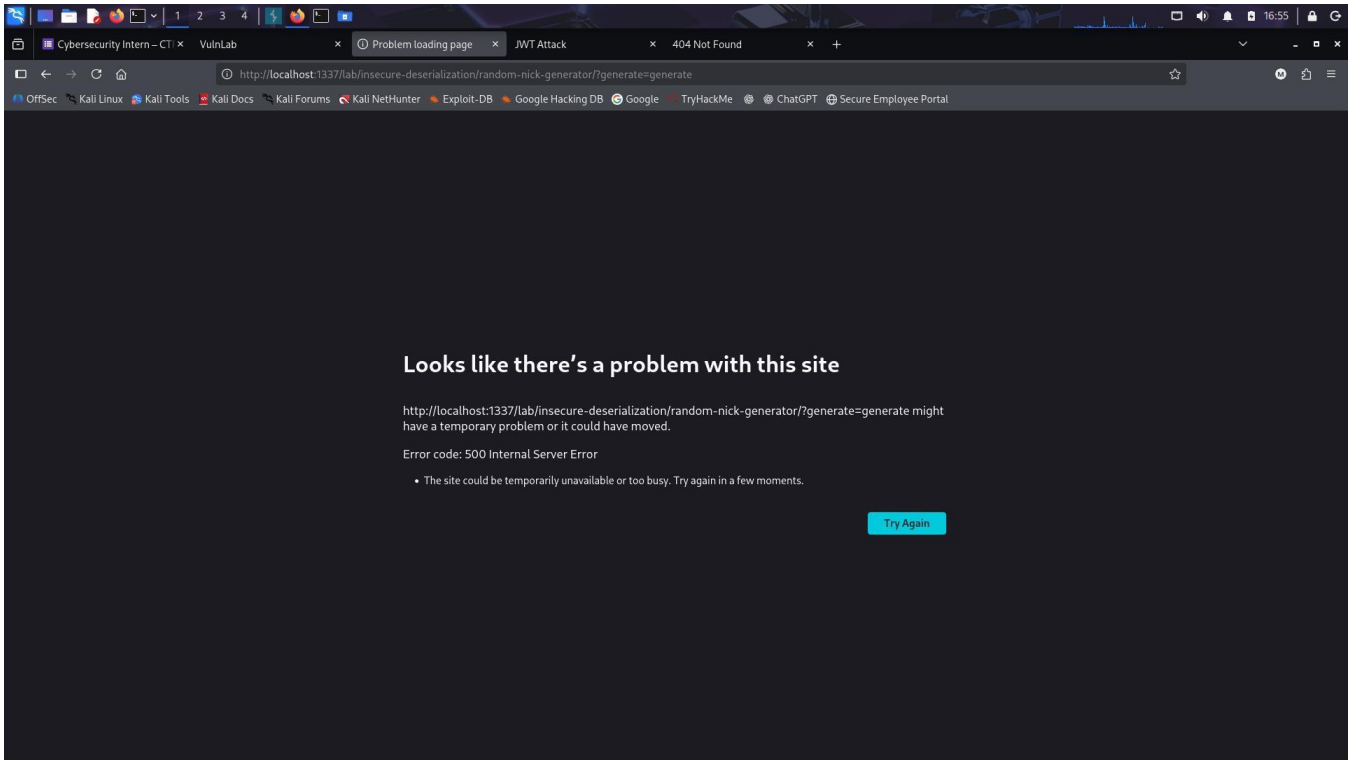
→ Remediation / Mitigation

- Avoid PHP unserialize() on untrusted data
- Use safe serialization formats (JSON)
- Apply cryptographic signing to session data
- Implement allowed_classes in unserialize()
- Validate and regenerate session data server-side
- Never trust client-controlled serialized objects

→ Screenshot Evidence







VULN 41- JWT ATTACK

→ Description

The application relies on **JSON Web Tokens (JWT)** for authentication and authorization but fails to properly validate the token signature and algorithm. By modifying the JWT payload and setting the algorithm to none, an attacker can **forge tokens and escalate privileges without knowing the secret key**.

The server accepts manipulated JWTs without verifying their integrity, allowing unauthorized access to protected resources.

Affected Component

Authorization / Cookie: JWT=<token>

or

Cookie: jwt=<base64_encoded_token>

Root Cause

- JWT signature verification is missing or improperly implemented
- The application accepts alg: none
- No cryptographic validation of JWT integrity
- Trusts client-controlled token contents

This violates JWT security best practices.

→ Proof of Concept (PoC)

Steps to Reproduce

1. Login to the application with a valid low-privileged account.
2. Capture the JWT token using **Burp Suite**.
3. Decode the JWT using Base64.
4. Modify the payload to escalate privileges (e.g., change username/role to admin).
5. Set the JWT header algorithm to none.
6. Remove the signature part of the token.
7. Re-encode the token and replace it in the request.

8. Send the modified request to the server.

Example Payload Used

JWT Header

```
{
  "typ":                                                    "JWT",
  "alg":                                                    "none"
}
```

JWT Payload

```
{
  "username":                                               "admin"
}
```

Final JWT Format

base64(header).base64(payload).

(Signature intentionally removed)

Observed Behavior

- Server accepts the modified JWT
- No signature verification error occurs
- Application processes request as an **admin user**
- Unauthorized privilege escalation confirmed

- Successful privileged response

→ Impact Analysis

Successful exploitation allows attackers to:

- Bypass authentication
- Impersonate any user (including admin)
- Gain full administrative access
- Perform unauthorized actions
- Completely compromise the application

JWT vulnerabilities often lead to **total application takeover**.

→ CVSS 3.1 Base Score

9.1 – Critical

Vector:

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N

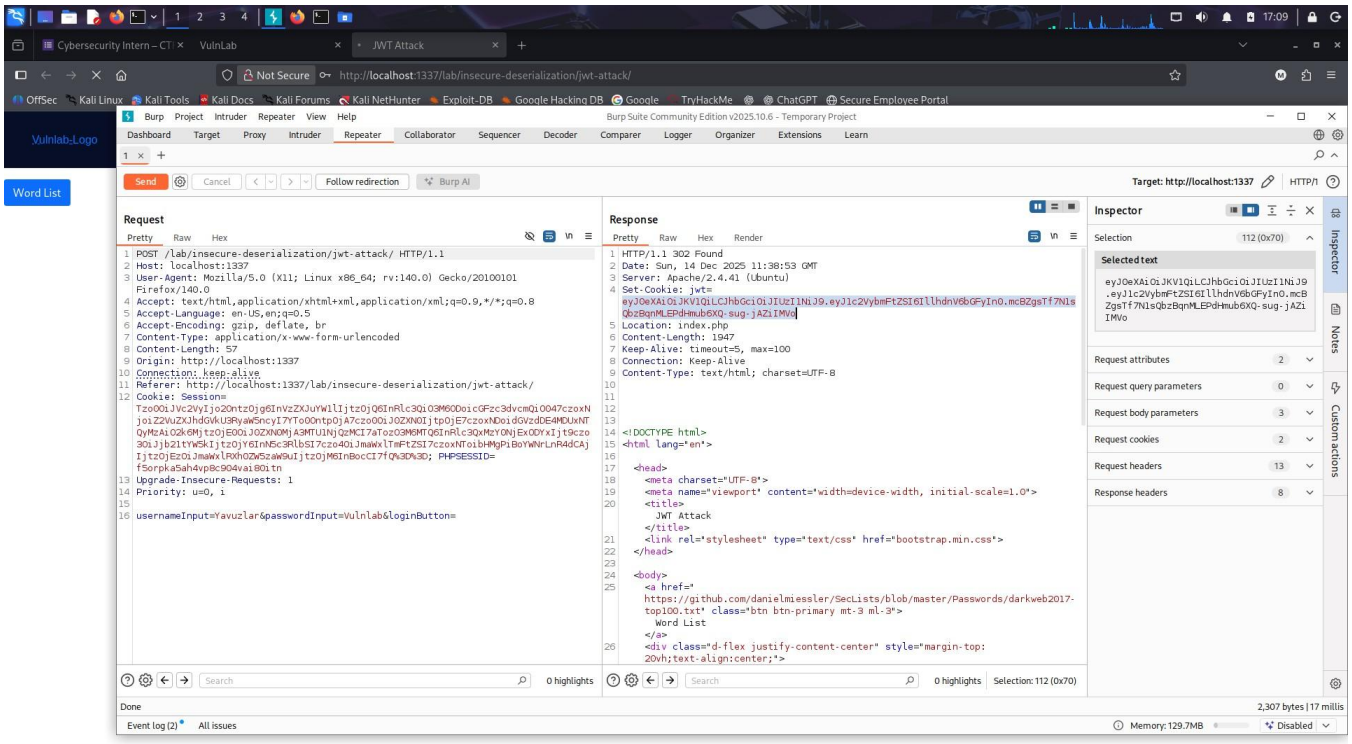
Risk Rating

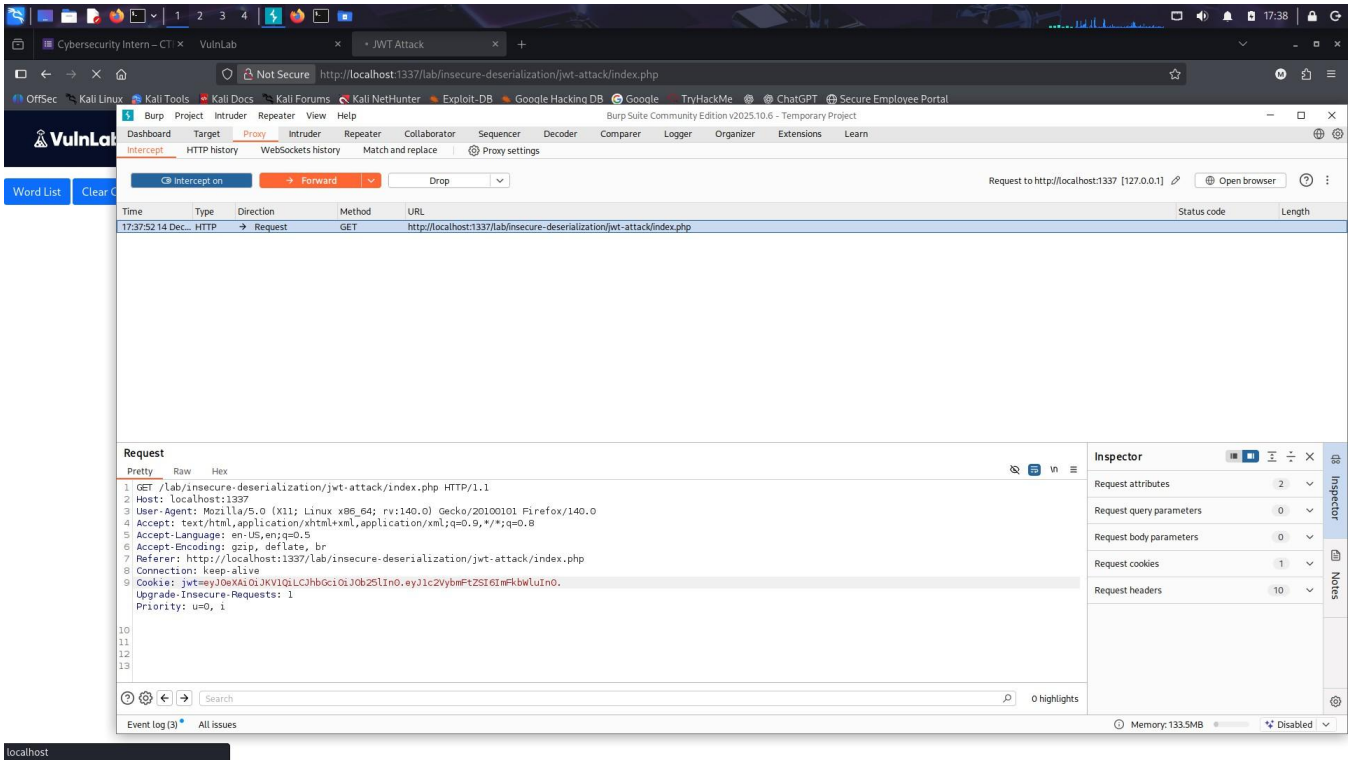
Critical score-9.1

→ Remediation / Mitigation

- Never allow alg: none
- Enforce strict algorithm whitelisting
- Always verify JWT signatures
- Use strong secret keys or asymmetric signing (RS256)
- Validate claims server-side
- Regenerate tokens after privilege changes

→ Screenshot Evidence





NEXT CATEGORY-BROKEN AUTHENTICATION

VULN 42- BRUTE FORCE

→ Description : The application does not implement protections against automated login attempts. An attacker can repeatedly submit authentication requests without any rate

limiting, account lockout, CAPTCHA, or monitoring. This allows an attacker to brute-force user credentials and gain unauthorized access.

→ Affected Component

Admin login functionality

→ Proof of Concept (PoC)

Username: admin

Attack Method: Brute-force using Burp Suite Intruder

Wordlist Used: 10k-most-common.txt (SecLists)

Successful Password Found: lifehack

Steps to Reproduce

- Intercept the admin login request using Burp Suite.
- Send the request to Intruder.
- Keep the username fixed as admin.
- Set the password parameter as the payload position.
- Load the 10k-most-common.txt wordlist.
- Start the attack.
- Observe a successful response when the password lifehack is used.

→ Impact Analysis

Successful exploitation allows an attacker to:

- Gain unauthorized access to the admin account
- Perform privileged actions
- Access or modify sensitive data

- Potentially take full control of the application

This vulnerability can lead to complete account compromise.

→ CVSS 3.1 Base Score

8.2 – High

Vector:

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N

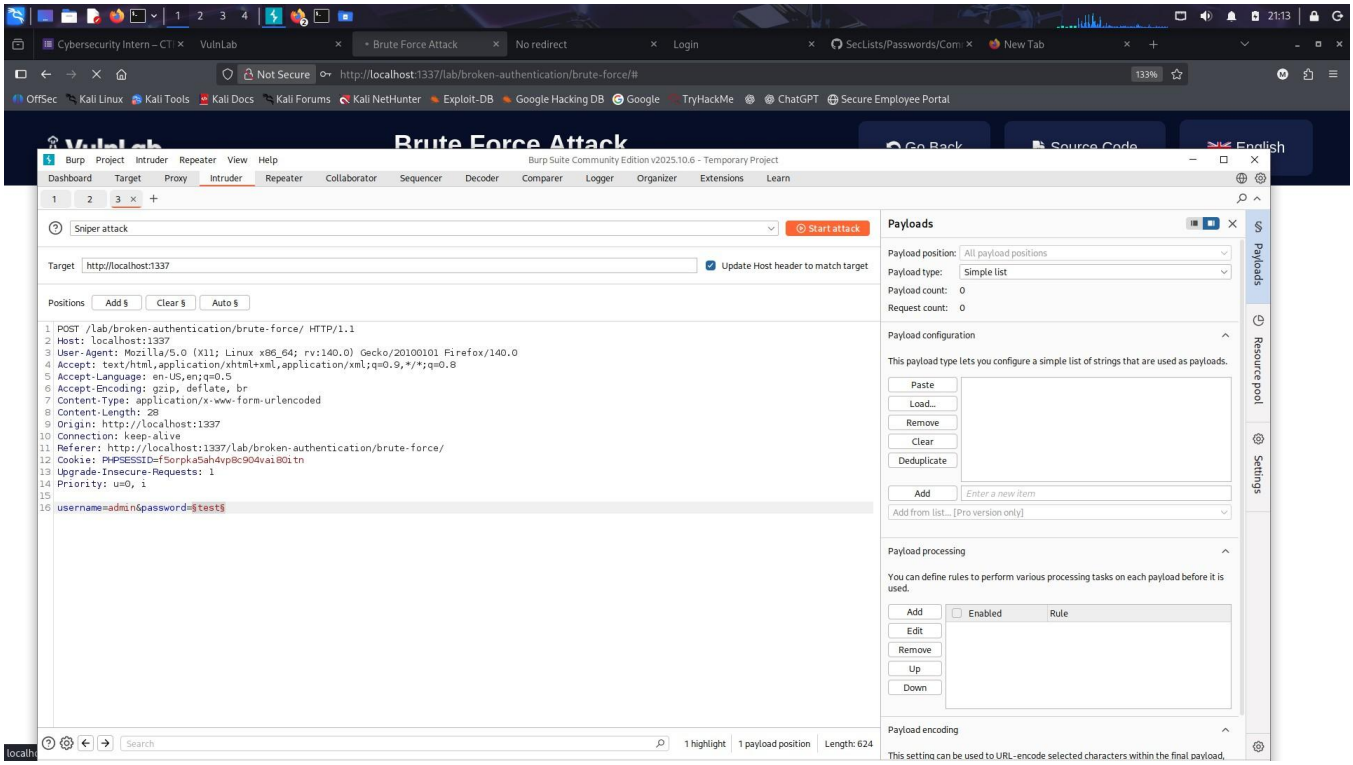
Risk Rating

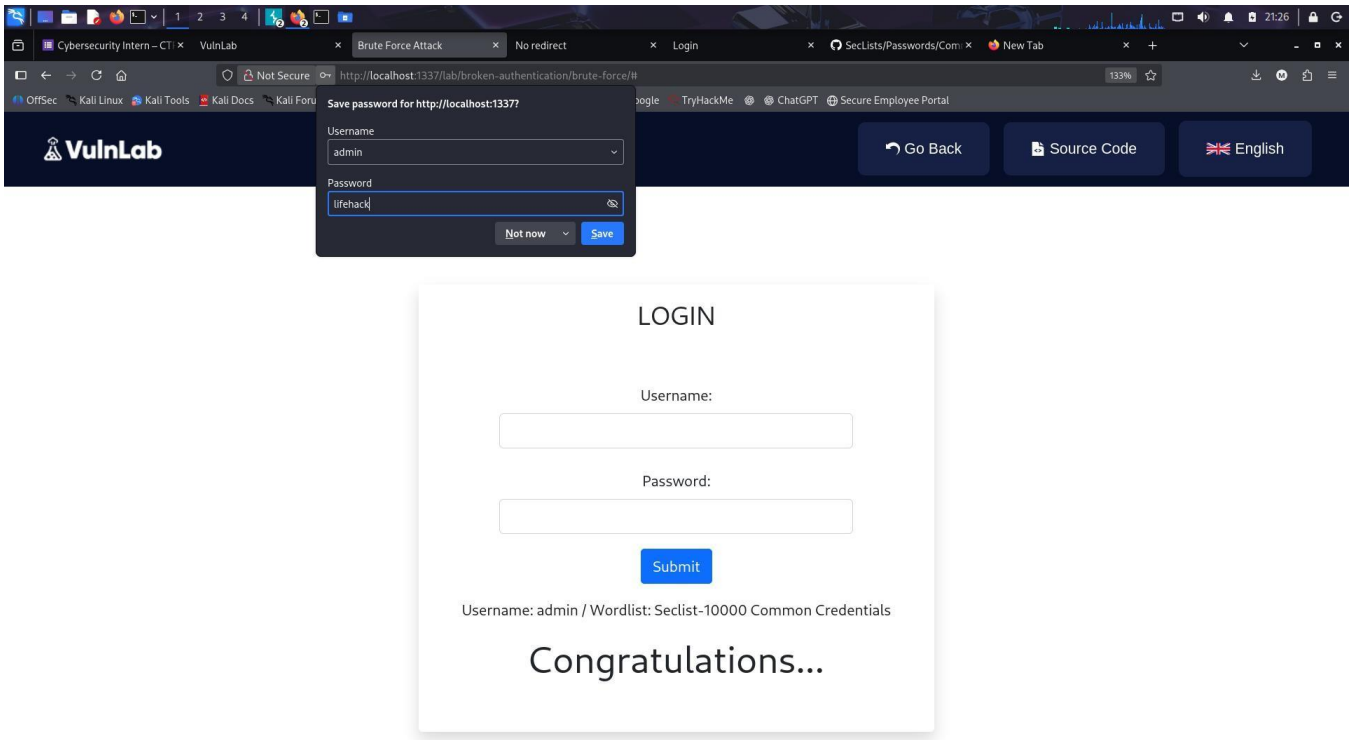
High

→ Remediation / Mitigation

- Implement rate limiting on login attempts
- Enforce account lockout after multiple failed attempts
- Use CAPTCHA after failed logins
- Enforce strong password policies
- Monitor and log failed authentication attempts
- Implement multi-factor authentication (MFA)

→ Evidence





VULN 43- NO REDIRECT

→ Description

The application performs an authorization check and issues a redirect (HTTP 302) to the login page for unauthenticated users. However, the server continues executing the request after sending the redirect, returning the protected page content in the same response. By manipulating the HTTP response, an attacker can suppress the redirect and access restricted content without authentication.

→ Affected Component

Admin panel authorization logic (index.php)

→ Proof of Concept (PoC)

Steps to Reproduce

- Navigate directly to the admin panel URL:

http://localhost:1337/lab/broken-authentication/no-redirect/index.php

- Enable Burp Suite → Proxy → Intercept (ON) and refresh the page.
- Right-click the intercepted request and select Do intercept → Response to this request.
- When the server response appears:
  Change the status line from: HTTP/1.1 302 Found
  to: HTTP/1.1 200 OK
- Remove the header:
  Location: login.php
- Forward the response to the browser.

Result : The browser renders the admin dashboard (SVPanel) without any login, confirming an authentication bypass.

→ Impact Analysis

An attacker can:

- Bypass authentication entirely

- Access administrative functionality
- Perform privileged actions without credentials
- Compromise application integrity and sensitive data
- This results in full authorization bypass.

→ CVSS 3.1 Base Score

8.8 – High

Vector:

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N

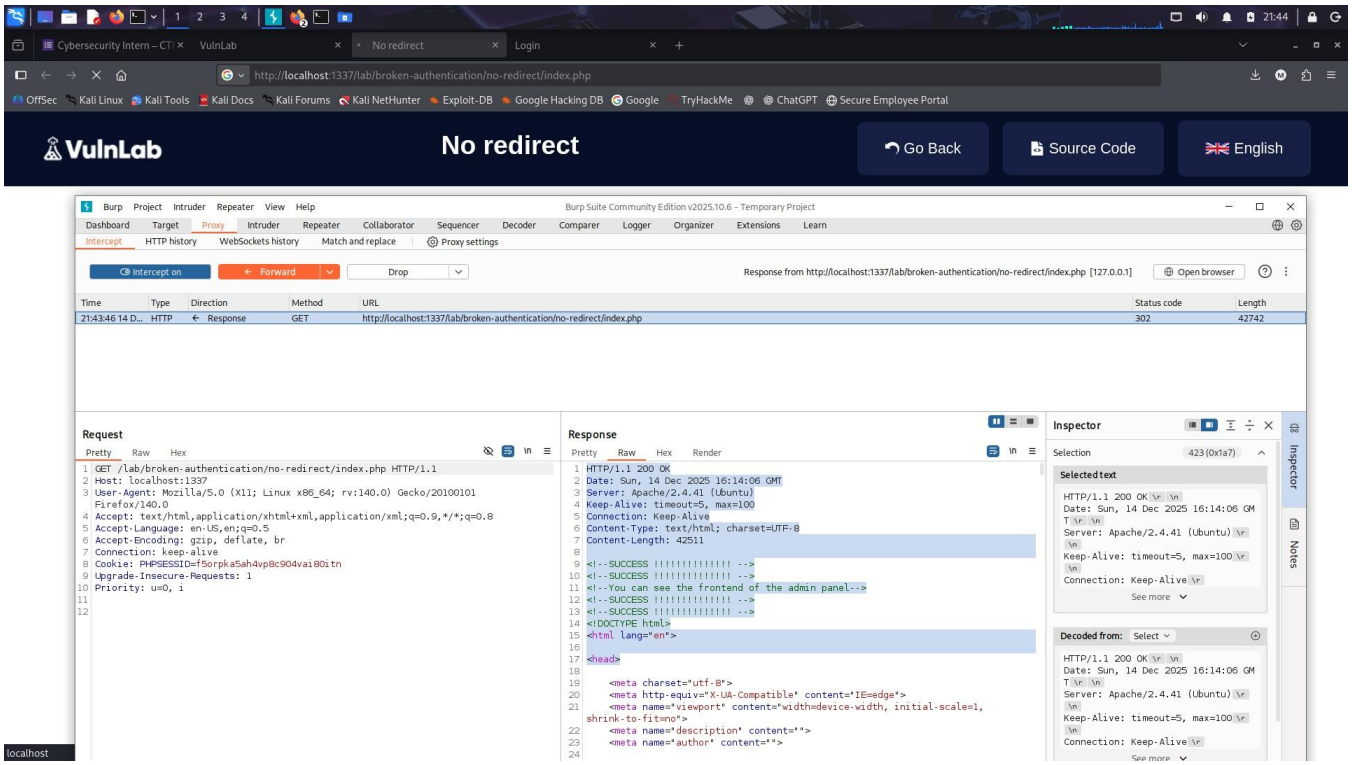
Risk Rating

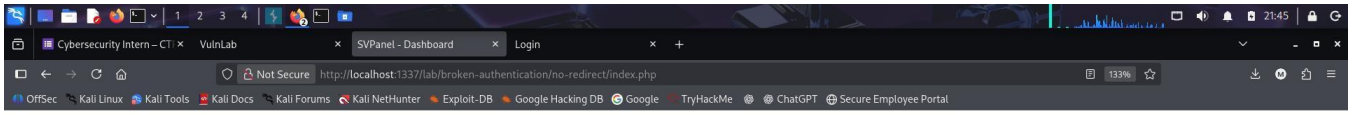
High

→ Remediation / Mitigation

- Immediately terminate execution after redirects (e.g., exit; / return;)
- Enforce server-side authorization checks before rendering protected content
- Use centralized access control middleware
- Add automated tests for unauthorized access paths

→ Evidence





Revenue Sources

Dropdown Header:
Action Another action
Something else here

Direct Social Referral

Projects

Server Migration 20%

Sales Tracking 40%

Customer Database 60%

Payout Details 80%

Account Setup Complete!

Primary
#4e73df
Success
#1cc88a

VULN 44- TWO-FACTOR AUTHENTICATION

→ Description

The application implements a 2-factor authentication mechanism using a server generated numeric verification code. However, the OTP implementation lacks essential security controls such as rate limiting, attempt counters, account lockout, and CAPTCHA protections.

The verification code is generated using a predictable numeric range, allowing an attacker to systematically brute-force the OTP and gain unauthorized access to the admin account.

→ Affected Component

2FA verification endpoint (OTP validation logic)

→ Proof of Concept (PoC)

Observation

- The application accepts a 5-digit numeric OTP
- No rate limiting or delay is enforced between attempts
- Unlimited verification attempts are allowed per session
- Invalid OTP submissions return consistent responses

Exploitation Evidence

- OTP verification requests were intercepted and replayed using Burp Suite
- Burp Intruder was configured to test the full numeric range
- The application did not block, delay, or lock the account during the attack

This confirms that the OTP can be brute-forced within a short time window

→ Impact Analysis

An attacker can:

- Bypass 2FA protection entirely
- Gain unauthorized access to the admin account
- Escalate privileges without valid OTP ownership
- Fully compromise application security

This completely defeats the purpose of two-factor authentication.

→ CVSS 3.1 Base Score

8.6 – High

Vector:

CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:N

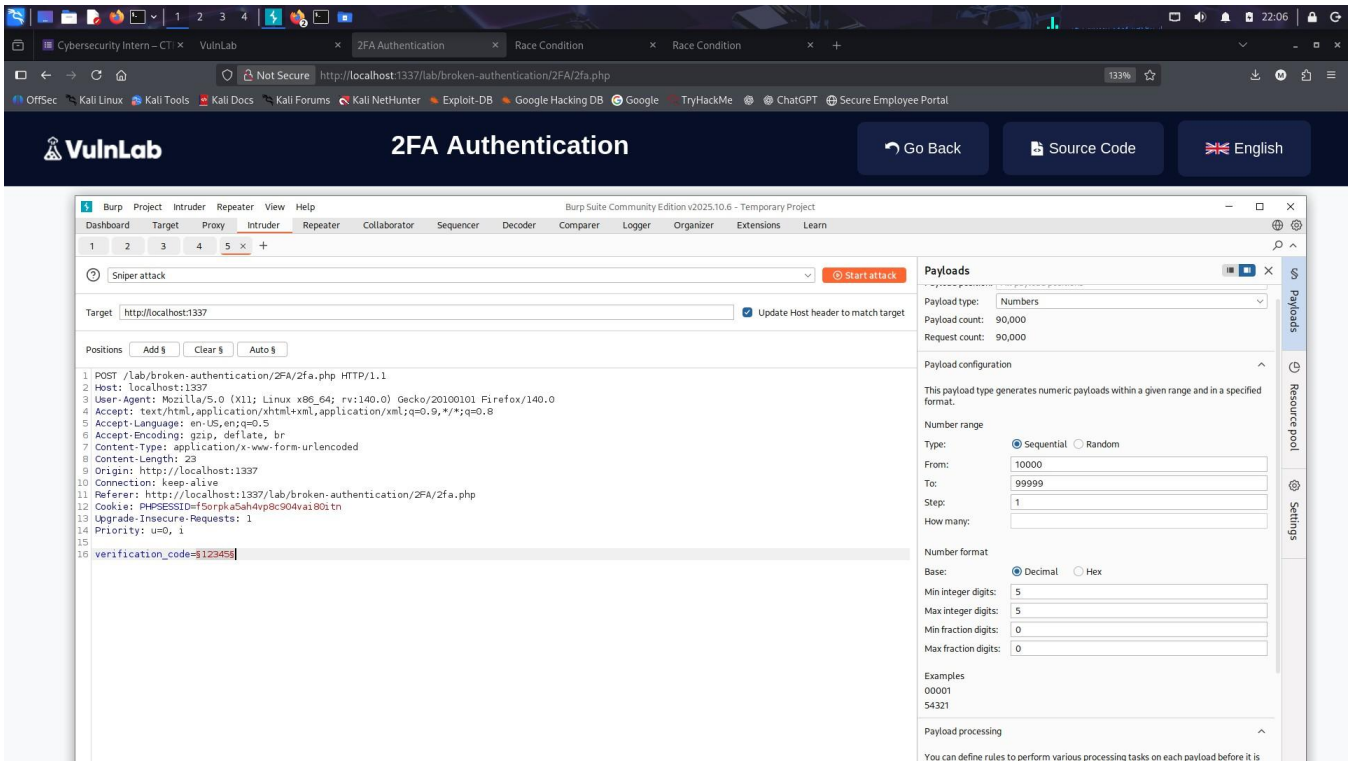
Risk Rating

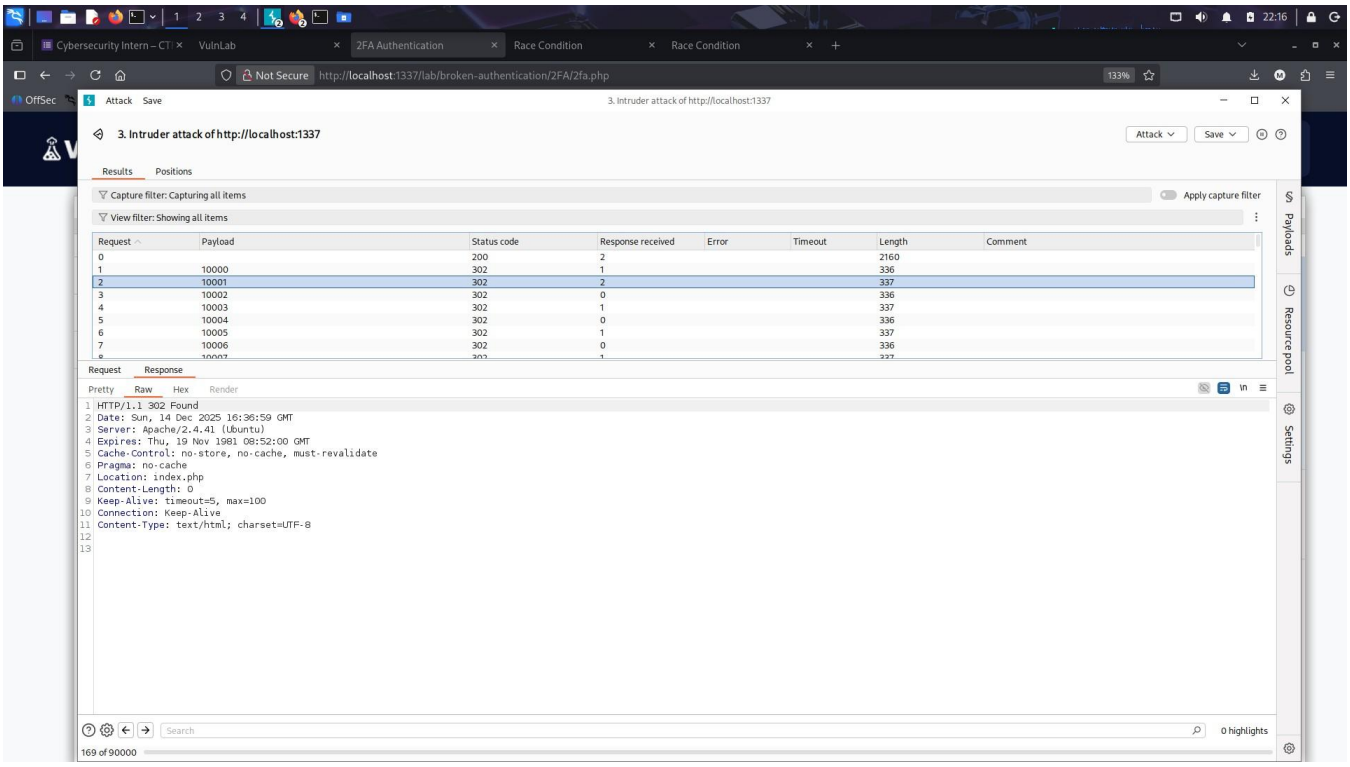
High

→ Remediation / Mitigation

- Implement strict rate limiting on OTP verification
- Enforce account lockout after failed attempts
- Add exponential backoff delays between attempts
- Use cryptographically secure OTP generators
- Bind OTP attempts to user identity and IP
- Implement CAPTCHA after multiple failures
- Log and alert on OTP abuse attempts

→ Evidence/screenshot





NEXT CATEGORY- RACE CONDITION

VULN45-RACE CONDITION IN THE REGISTRATION FORM

→ Description

The application enforces a business rule that an email address can be used only once during registration. However, the registration logic does not handle concurrent requests safely. By sending multiple registration requests simultaneously with the same email address, multiple accounts can be created, bypassing the uniqueness restriction.

→ Affected Component

User registration functionality

→ Proof of Concept (PoC)

Steps to Reproduce:

- Prepare a valid registration request with the email: test@hack.com
- Send the request to Burp Repeater.
- Duplicate the request into multiple tabs.
- Group the requests and send them in parallel.
- Observe that all requests return: HTTP/1.1 200 OK

This confirms that multiple accounts were created using the same email address.

→ Impact Analysis

An attacker can:

- Create unlimited duplicate accounts
- Bypass uniqueness constraints
- Abuse referral systems
- Evade fraud detection
- Disrupt account integrity

This can lead to financial abuse and loss of trust in the system.

→ CVSS 3.1 Base Score

6.5 – Medium

Vector:

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:H/A:N

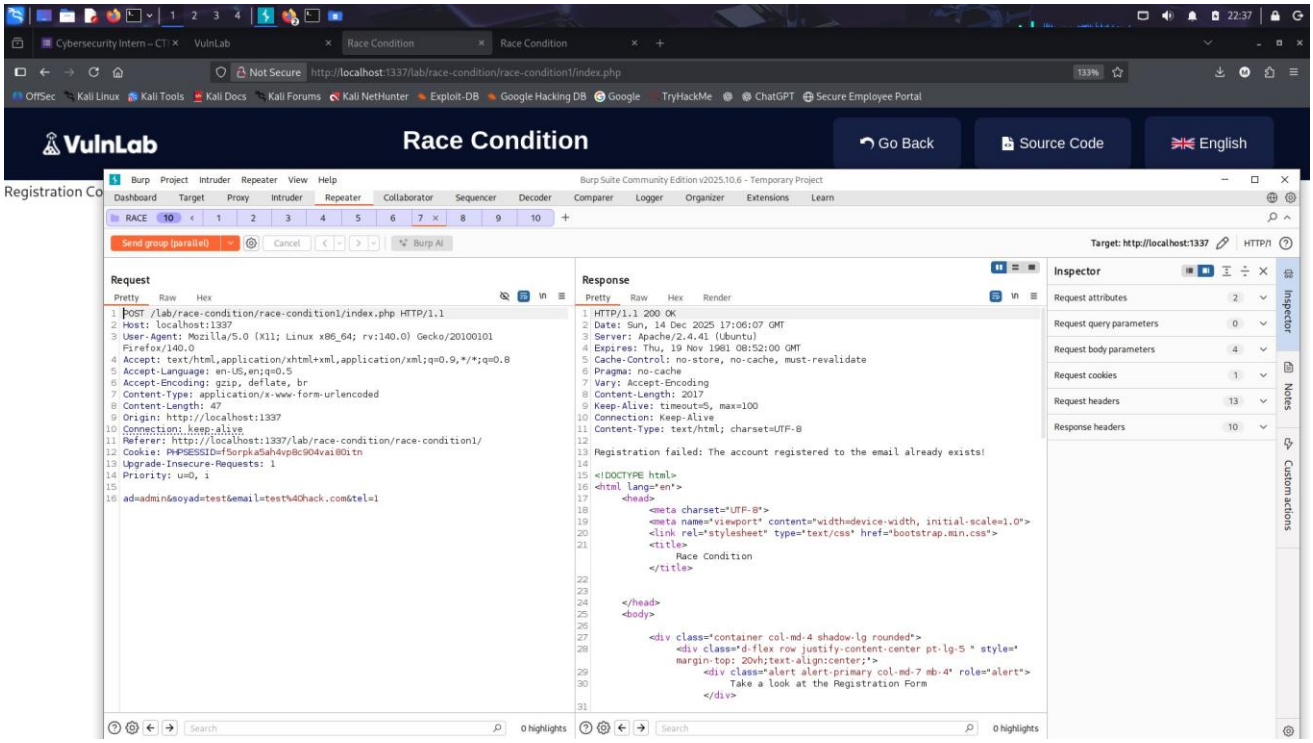
Risk Rating

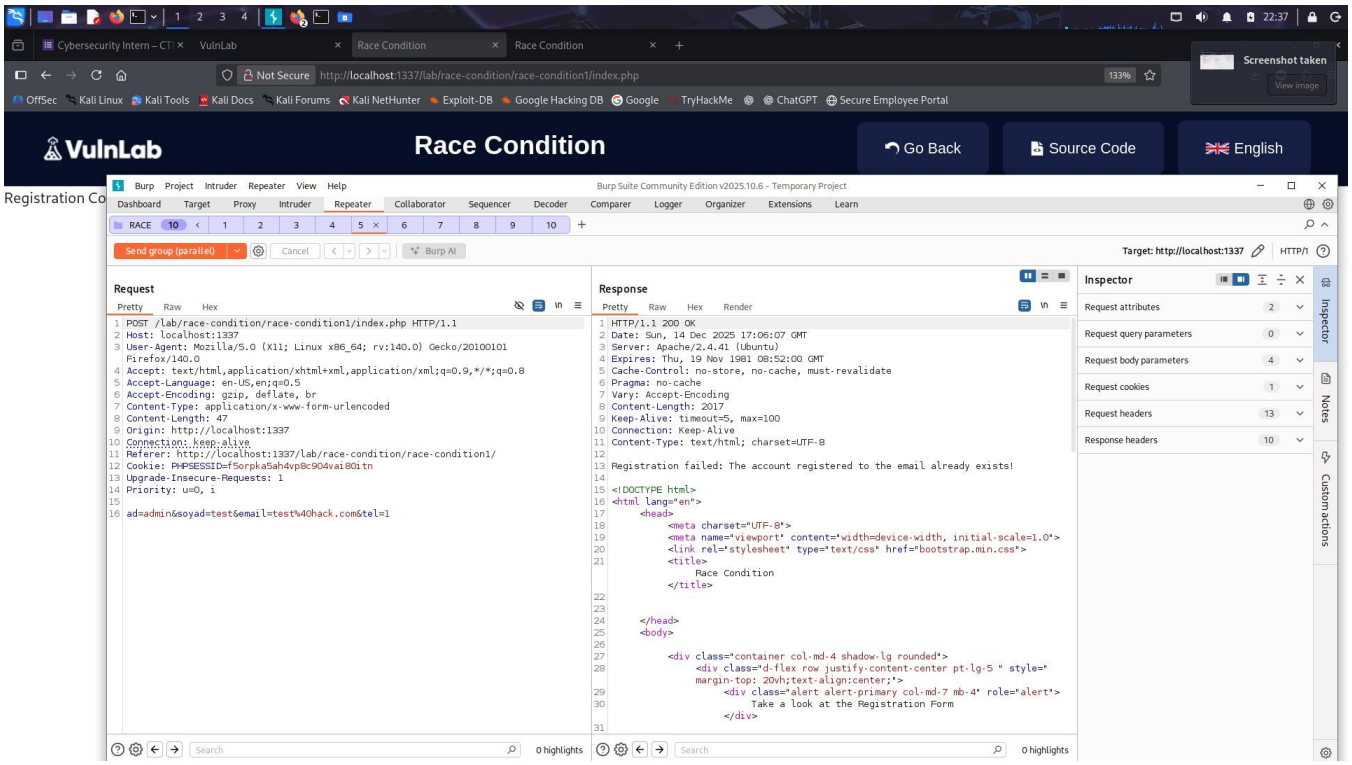
Medium

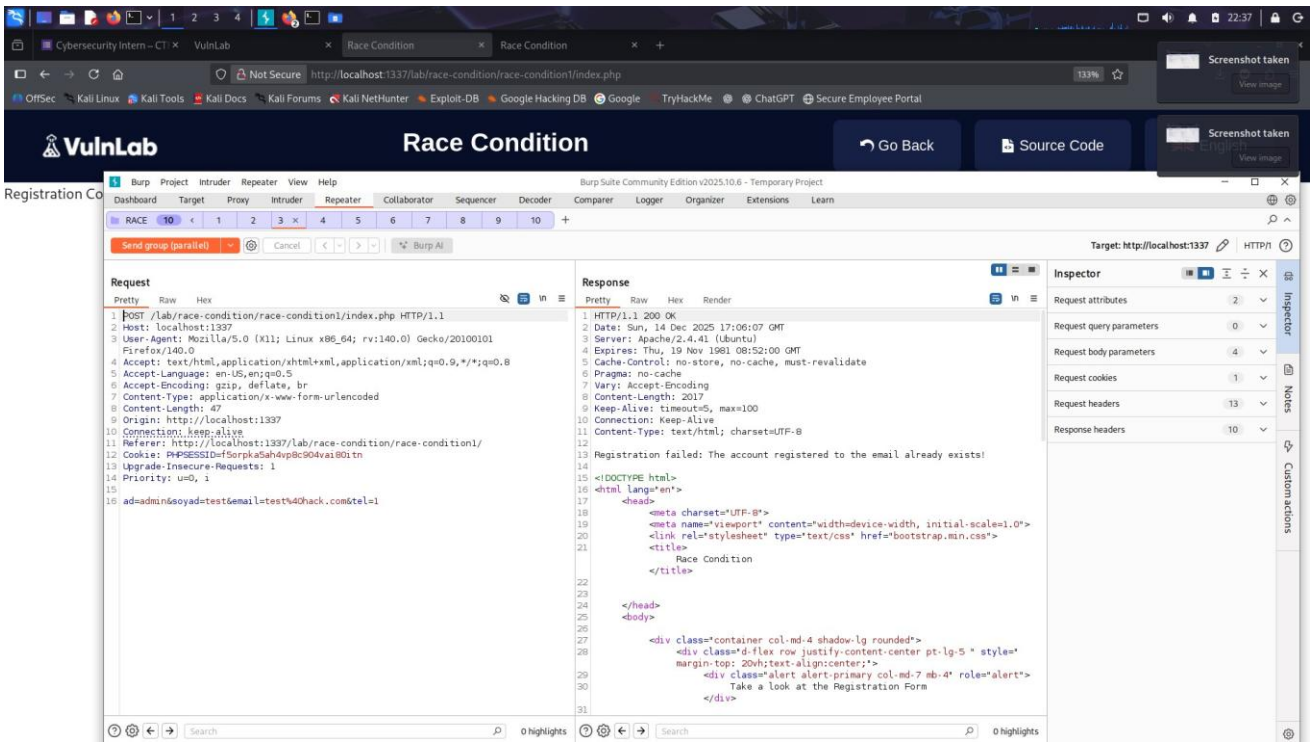
→ Remediation / Mitigation

- Implement server-side locking mechanisms
- Use database-level UNIQUE constraints
- Ensure atomic operations during registration
- Apply transaction isolation
- Reject duplicate requests at commit time

→ Evidence




VulnLab
Race Condition
Go Back
Source Code
English
Registration Co
Request
Response
Inspector
POST /lab/race-condition/race-condition1/index.php HTTP/1.1
HTTP/1.1 200 OK
Registration failed: The account registered to the email already exists!
Take a look at the Registration Form


VulnLab
Race Condition
Go Back
Source Code
Registration Co
Request
Response
Inspector
POST /lab/race-condition/race-condition1/index.php HTTP/1.1
HTTP/1.1 200 OK
Registration failed: The account registered to the email already exists!
Take a look at the Registration Form

VULN46- DISCOUNT CODE APPLICATION IN SHOPPING CART

→ Description

The application provides a discount code feature that is intended to be applied only once per order or user session. However, the server does not properly enforce atomic validation when processing concurrent requests. By sending multiple discount code application requests simultaneously, the same discount code can be applied multiple times, resulting in excessive and unauthorized discounts.

This occurs due to the absence of concurrency control and improper synchronization in the discount validation logic.

→ Affected Component

Shopping Cart – Discount Code Application Functionality

→ Proof of Concept (PoC)

Steps to Reproduce:

- Add any product to the shopping cart.

- Enter a valid discount code: sbrvtn50
- Intercept the discount code application request using Burp Suite.
- Send the request to Burp Repeater.
- Duplicate the request into multiple tabs (e.g., 10–15).
- Group all requests and send them in parallel.
- Observe that all requests return: HTTP/1.1 200 OK

The discount code is applied multiple times successfully.

→ Impact Analysis

An attacker can:

- Apply the same discount code multiple times
- Reduce the total cart value to near zero
- Cause direct financial loss
- Abuse promotional campaigns
- Bypass business logic controls

This vulnerability can lead to revenue loss and abuse of promotional mechanisms.

→ CVSS 3.1 Base Score

7.1 – High

Vector String:

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:H/A:N

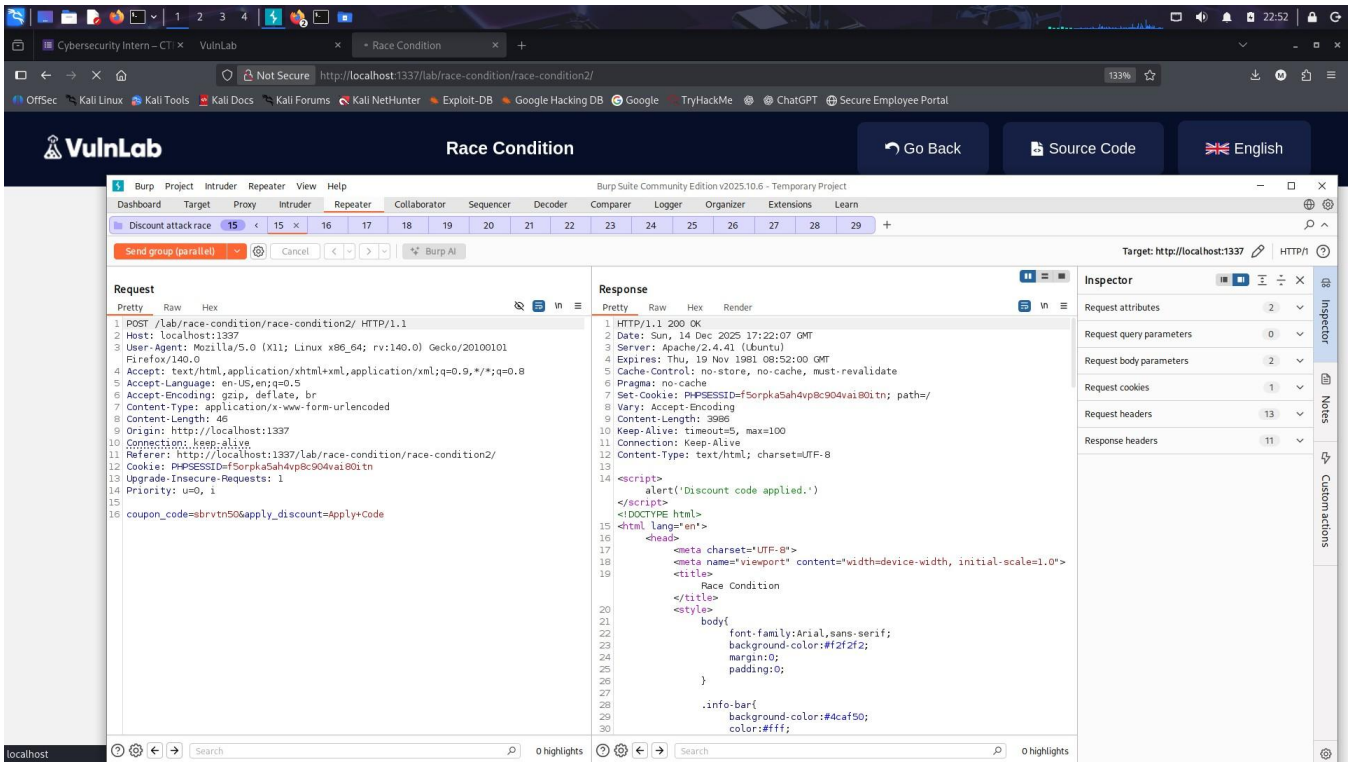
Risk Rating

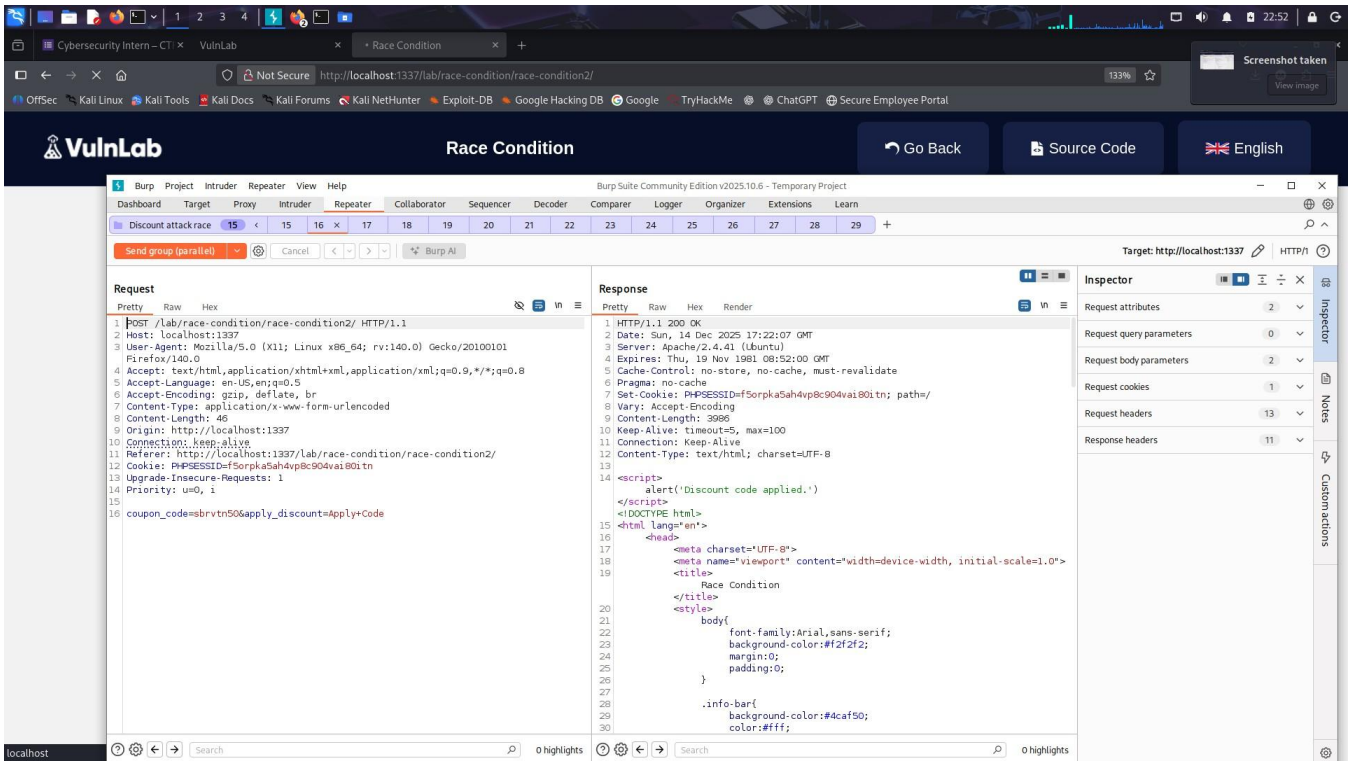
High

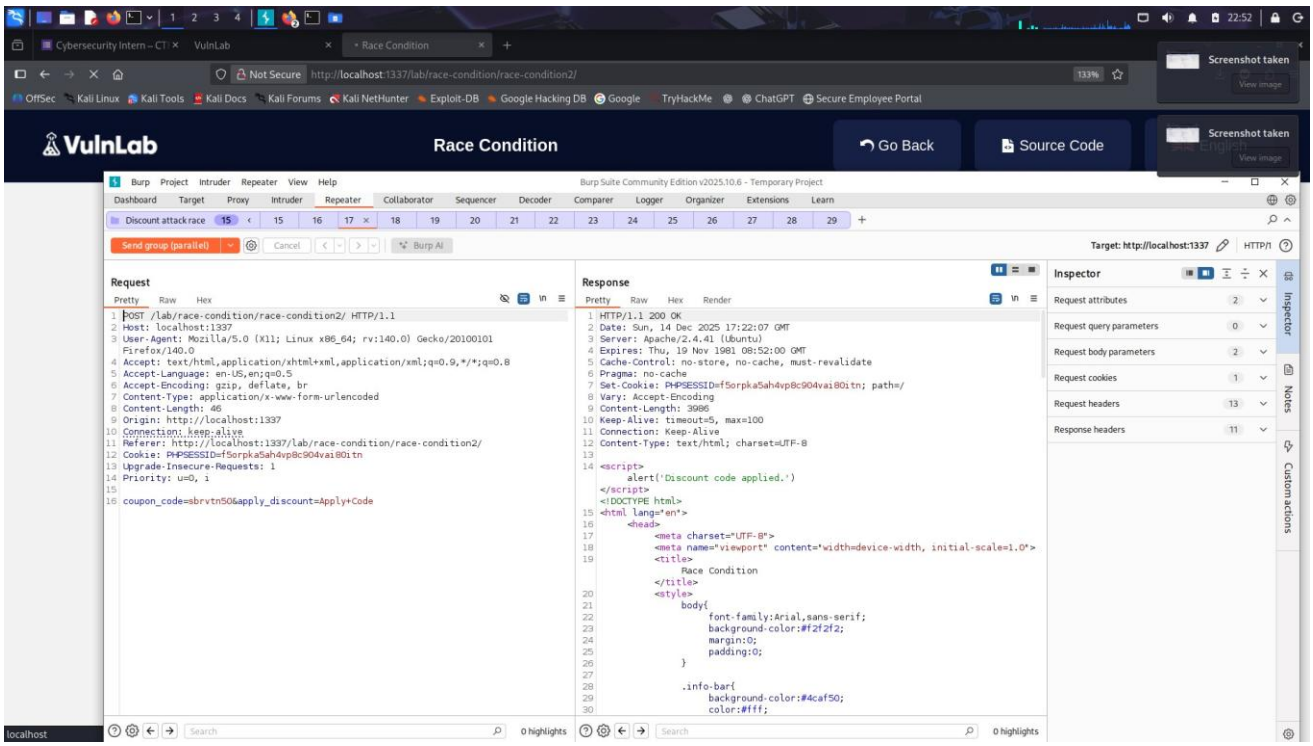
→ Remediation / Mitigation

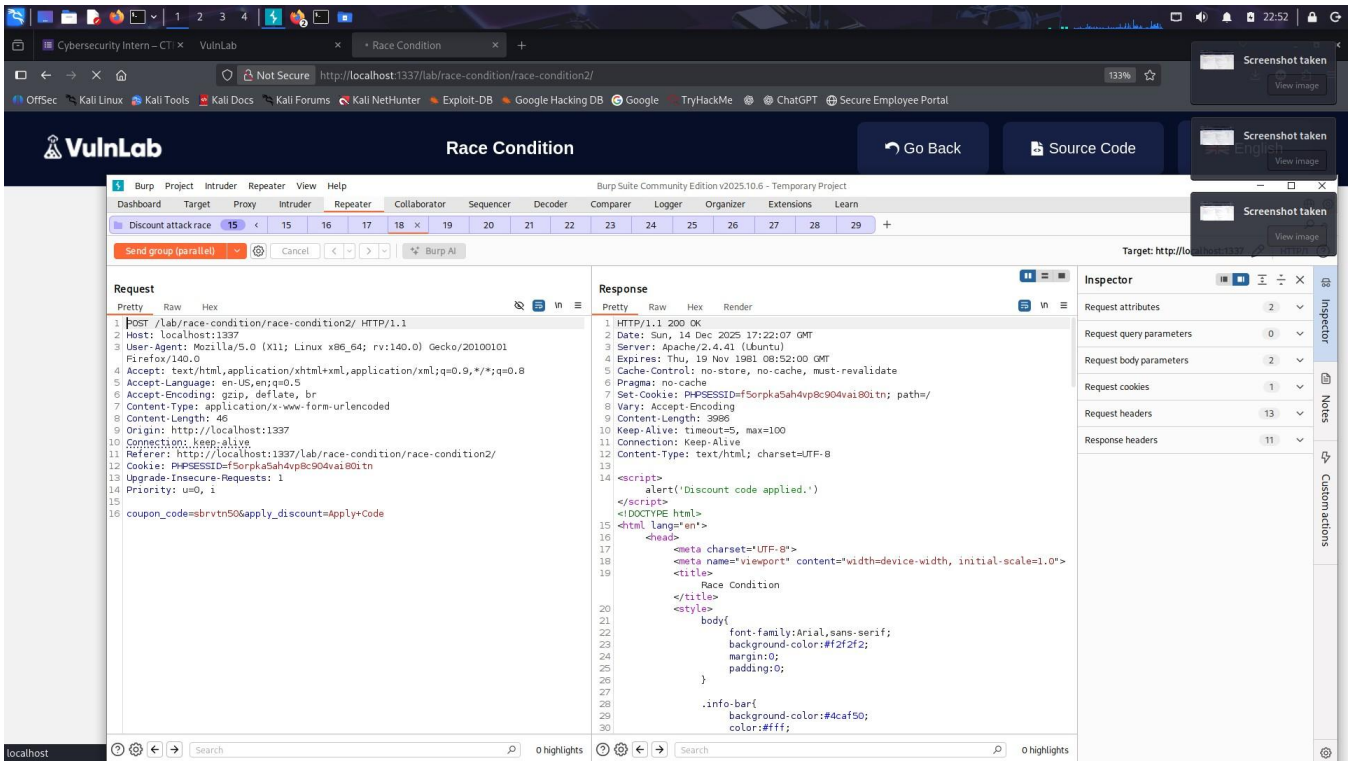
- Enforce atomic server-side checks for discount application
- Use database transactions with proper isolation levels
- Lock discount usage per user/session/order
- Maintain server-side flags to prevent reuse
- Validate discount usage at final checkout, not only on apply

→ Evidence / Screenshots




VulnLab
Race Condition
Go Back
Source Code
English
Request
POST /lab/race-condition/race-condition2/ HTTP/1.1
Host: localhost:1337
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:140.0) Gecko/20100101 Firefox/140.0
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: 46
Origin: http://localhost:1337
Connection: keep-alive
Referer: http://localhost:1337/lab/race-condition/race-condition2/
Cookie: PHPSESSID=f5orpka5ah4vp8c904vai80itn
Upgrade-Insecure-Requests: 1
Priority: u=0, i
coupon_code=sbrvtn50&apply_discount=Apply+Code
Response
HTTP/1.1 200 OK
Date: Sun, 14 Dec 2025 17:22:07 GMT
Server: Apache/2.4.41 (Ubuntu)
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Set-Cookie: PHPSESSID=f5orpka5ah4vp8c904vai80itn; path=/
Vary: Accept-Encoding
Content-Length: 3986
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
<script>
alert('Discount code applied.')
</script>
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>
Race Condition
</title>
<style>
body{
font-family:Arial,sans-serif;
background-color:#f2f2f2;
margin:0;
padding:0;
}
.info-bar{
background-color:#4caf50;
color:#fff;
Inspector
Request attributes
Request query parameters
Request body parameters
Request cookies
Request headers
Response headers


VulnLab
Race Condition
Go Back
Source Code
Request
POST /lab/race-condition/race-condition2/ HTTP/1.1
Host: localhost:1337
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:140.0) Gecko/20100101 Firefox/140.0
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: 46
Origin: http://localhost:1337
Connection: keep-alive
Referer: http://localhost:1337/lab/race-condition/race-condition2/
Cookie: PHPSESSID=f5orpka5ah4vp8c904vai80itn
Upgrade-Insecure-Requests: 1
Priority: u=0, i
coupon_code=sbrvtn50&apply_discount=Apply+Code
Response
HTTP/1.1 200 OK
Date: Sun, 14 Dec 2025 17:22:07 GMT
Server: Apache/2.4.41 (Ubuntu)
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Set-Cookie: PHPSESSID=f5orpka5ah4vp8c904vai80itn; path=/
Vary: Accept-Encoding
Content-Length: 3986
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
<script>
alert('Discount code applied.')
</script>
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>
Race Condition
</title>
<style>
body{
font-family:Arial,sans-serif;
background-color:#f2f2f2;
margin:0;
padding:0;
}
.info-bar{
background-color:#4caf50;
color:#fff;
Inspector
Request attributes
Request query parameters
Request body parameters
Request cookies
Request headers
Response headers



NEXT CATEGORY-SERVER-SIDE TEMPLATE INJECTION(SSTI)

VULN 47- SSTI-BASIC

→ Description

The application renders user-supplied input directly inside a server-side template without proper sanitization or sandboxing. As a result, template expressions provided by the user are evaluated by the template engine on the server, allowing arbitrary template execution.

This confirms the presence of a Server-Side Template Injection (SSTI) vulnerability.

→ Affected Endpoint

SSTI – Basic (Introduction / Input field where user introduces themselves)

→ Proof of Concept (PoC)

Payload Used:

{{6*6}}

Steps to Reproduce:

1. Navigate to the SSTI-Basic lab page.
2. Enter the payload {{6*6}} into the input field.
3. Submit the form.
4. Observe that the output displays: 36

Result: The server evaluates the template expression instead of treating it as plain text.

→ Impact Analysis
Successful exploitation confirms arbitrary template execution on the server. Depending on the template engine and configuration, this can lead to:

- Disclosure of sensitive server-side data
- Access to application configuration
- Server-side file read/write
- Remote Code Execution (RCE) in advanced cases

This vulnerability directly compromises server-side integrity.

→ CVSS 3.1 Base Score

8.1 – High

Vector String:

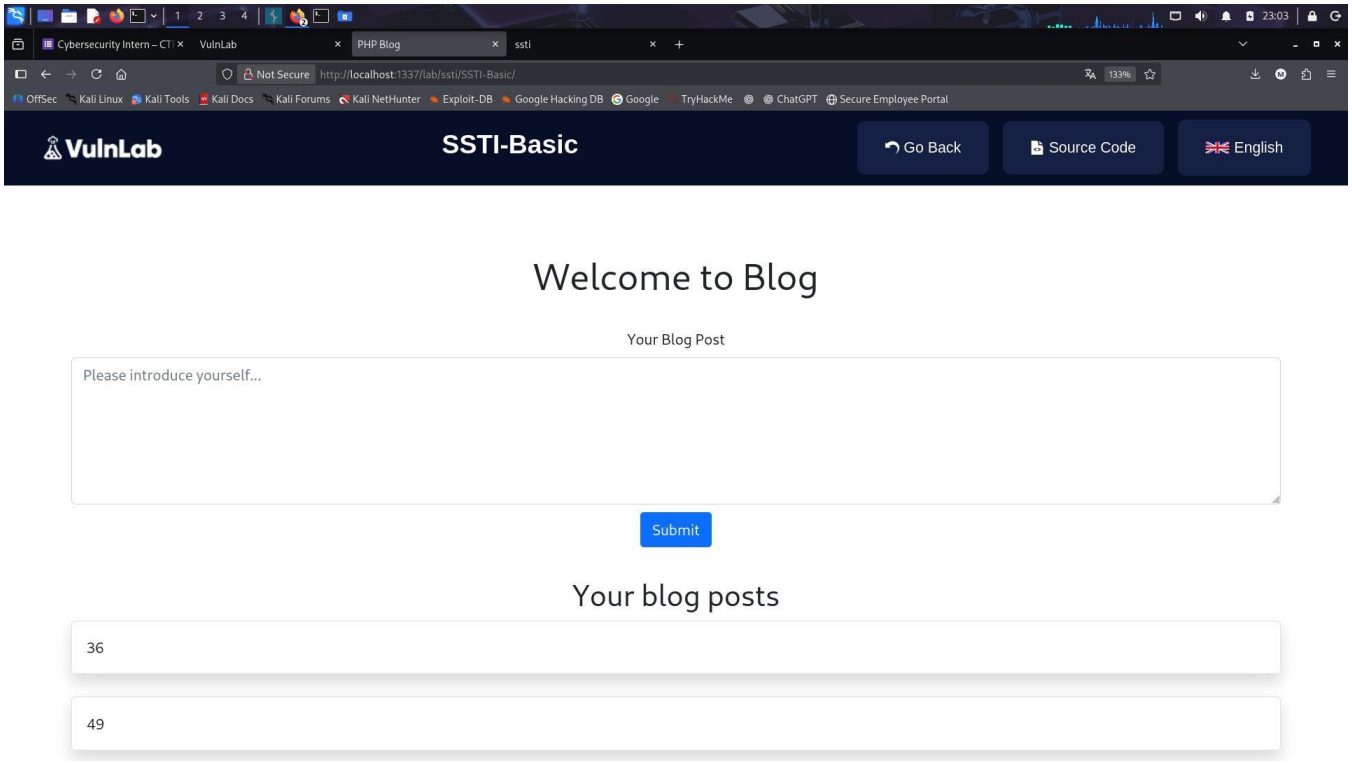
CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N

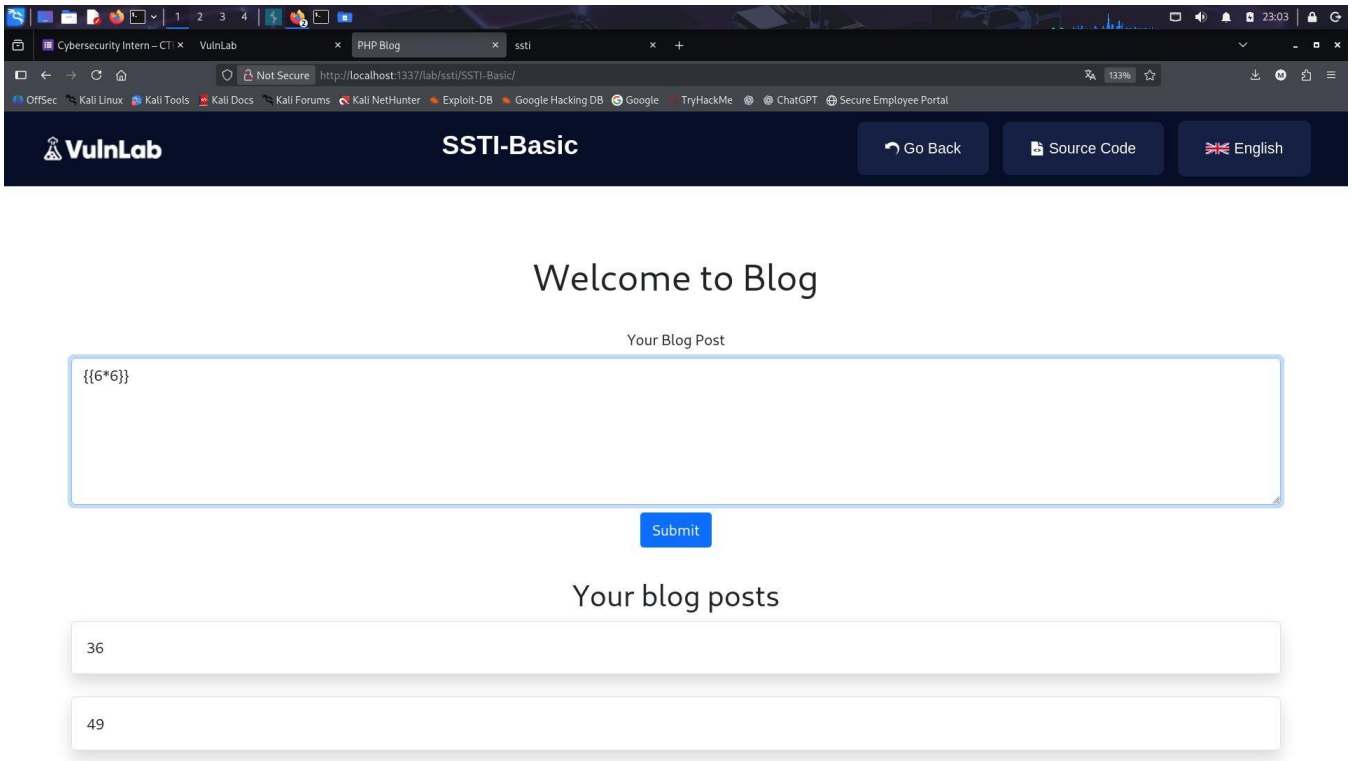
Risk Rating : High

→ Remediation / Mitigation

- Never render untrusted user input inside templates
- Use strict allowlists for template variables
- Disable expression evaluation where not required
- Apply context-aware output encoding
- Use sandboxed template environments

→ Evidence / Screenshot





VULN 48- SSTI-BLACKLIST

→ Description

The application attempts to prevent Server-Side Template Injection by blacklisting specific template syntax such as {{ and }}. However, this blacklist-based protection is ineffective and can be bypassed by crafting alternative template expressions that reconstruct the blocked tokens at runtime.

User-controlled input is still processed by the server-side template engine, resulting in template parsing and execution.

→ Affected Component

SSTI – Blacklist (User input field processed by template engine)

→ Proof of Concept (PoC)

Payload Used:

{}}{

Steps to Reproduce:

1. Navigate to the SSTI – Blacklist lab.
2. Enter the payload {}}{ into the vulnerable input field.
3. Submit the request.
4. Observe the server response.

Observed Result: The server returns the following error:

ERROR: Unexpected token "end of template" of value "" in "{{" at line 1

Why This Confirms the Vulnerability

The error message explicitly references {{, which means:

- The blacklist filter was bypassed.
- The template engine successfully reconstructed and parsed the opening template tag.
- The error occurred only because the template expression was syntactically incomplete, not because the payload was blocked.
- This confirms that user input reaches the template engine and is evaluated.
- This behavior definitively proves the presence of an SSTI vulnerability despite blacklist protections.

→ Impact Analysis

A successful blacklist bypass allows an attacker to:

- Execute arbitrary template expressions
- Read sensitive server-side data
- Access application configuration
- Potentially achieve Remote Code Execution (RCE), depending on the template engine
- Blacklist-based defenses provide no real protection against SSTI attacks.

→ CVSS 3.1 Base Score

8.1 – High

Vector String:

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N

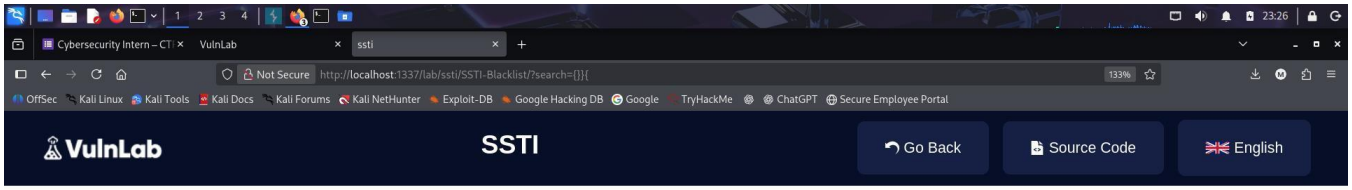
Risk Rating

High

→ Remediation / Mitigation

- Do not use blacklist-based filtering for template engines
- Avoid rendering user input directly in templates
- Use strict allowlists for permitted variables
- Disable template expression evaluation where not required
- Employ sandboxed or logic-less template engines

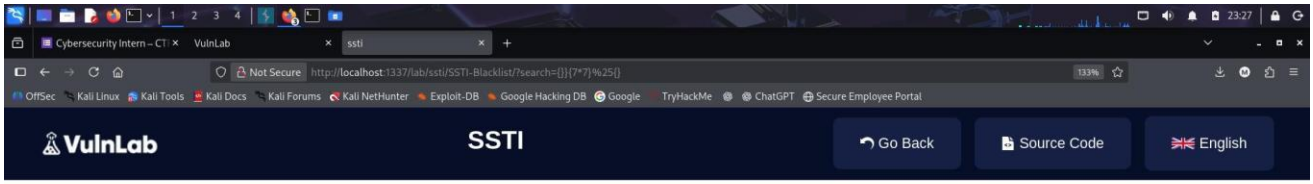
→ Evidence / Screenshot



{}}{

Search

ERROR:Unexpected token "end of template" of value "" in "{{" at line 1.



{}}{7*7}%{}

Search

ERROR:Unexpected "}" in "{{7*7}%{}" at line 1.

NEXT CATEGORY- CAPTCHA BYPASS

VULN 49- CAPTCHA BYPASS

CAPTCHA Bypass via CAPTCHA Reuse

→ Description

The application implements a CAPTCHA mechanism to prevent automated submissions. However, the CAPTCHA value remains valid even after successful verification. An attacker can reuse a previously solved CAPTCHA token to submit unlimited requests, effectively bypassing CAPTCHA protection.

→ Proof of Concept (PoC)

- Open the CAPTCHA-protected form.
- Solve the CAPTCHA once and submit a valid request.
- Intercept the request using Burp Suite and send it to Repeater.
- Modify only the user_comment parameter.
- Reuse the same captcha value.
- Send the request multiple times.

Observed Result:

Each request returns HTTP 302 Found, confirming successful submission without solving a new CAPTCHA.

Example Request:

POST /lab/captcha-bypass/broken-captcha HTTP/1.1

user_comment=Hacked&captcha=NE4655

→ Impact Analysis

- CAPTCHA protection can be completely bypassed
- Enables automated spam, brute-force attacks, and abuse
- Defeats bot-mitigation controls
- Increases risk of DoS, credential stuffing, and form abuse

→ CVSS 3.1 Score

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:L/A:N

Base Score: 5.3 (Medium)

(You can justify Medium because it impacts integrity, not direct data loss.)

Risk Severity : Medium

Root Cause:

CAPTCHA value is not invalidated after successful verification

CAPTCHA token remains reusable across multiple requests

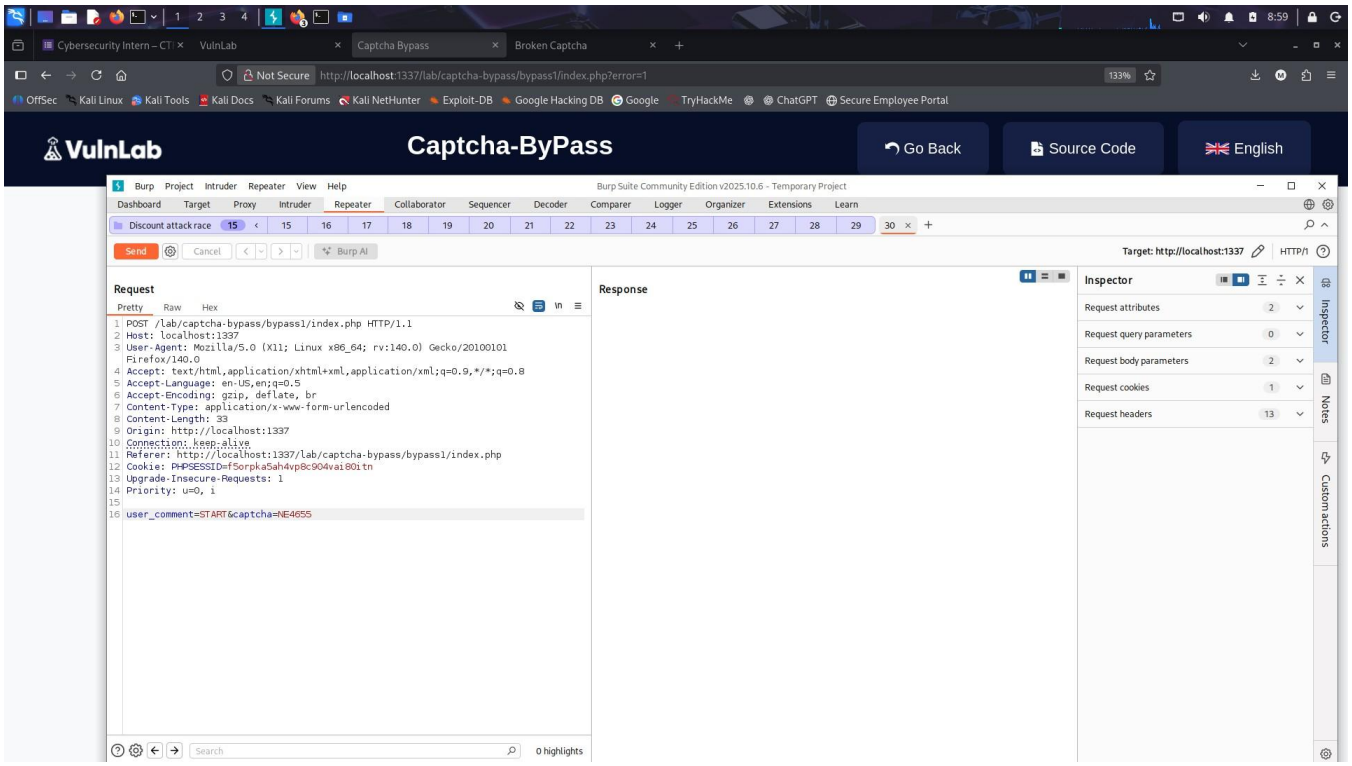
→ Remediation / Mitigation

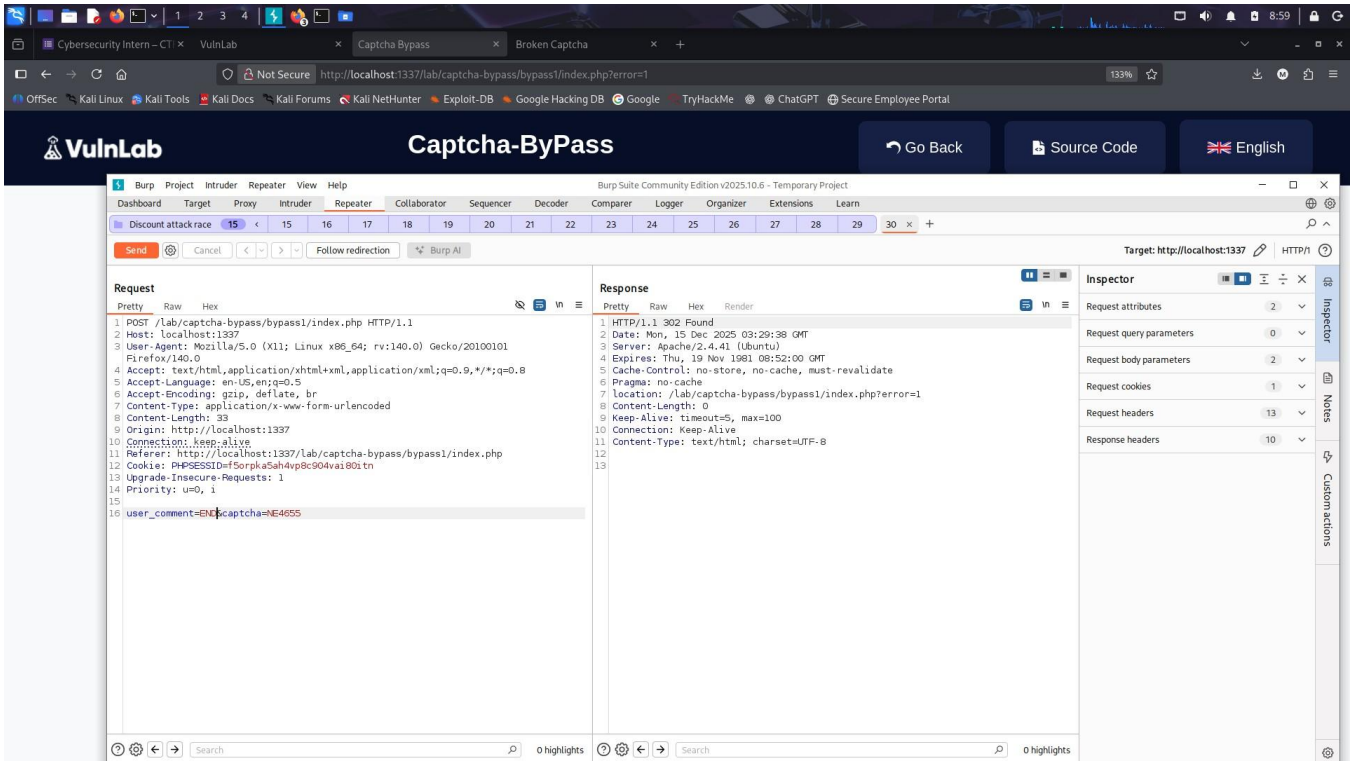
- Invalidate CAPTCHA token immediately after successful use
  Bind CAPTCHA to:
  - Session
  - Single request

    - Timestamp
- Implement server-side CAPTCHA regeneration
- Add rate limiting and request throttling
- Secure Example:

```
unset($_SESSION['captcha']);
```

→ Screenshots / Evidence





VULN 50- BROKEN CAPTCHA

CAPTCHA Bypass via Client-Side Manipulation (Logic Flaw)

→ Description

The application uses a CAPTCHA mechanism based on simple arithmetic values submitted by the client. The server trusts CAPTCHA operands and results provided in the HTTP request instead of validating them securely on the server side. As a result, an attacker can arbitrarily modify the CAPTCHA parameters and submit a forged correct answer, bypassing CAPTCHA validation entirely.

→ Proof of Concept (PoC)

- Open the CAPTCHA-protected form.
- Intercept the submission request using Burp Suite.
- Observe CAPTCHA-related parameters in the request (e.g., numbers and result).
- Modify the CAPTCHA operands and result to any values that satisfy the calculation.
- Forward the modified request.
  Example Manipulation:

captcha_num1=1 captcha_num2=1 captcha_result=2

- Submit the request.

Observed Result: The server accepts the request and returns a success response, confirming CAPTCHA bypass.

→ Impact Analysis

- CAPTCHA protection is completely ineffective
- Allows automated submissions without solving real challenges
- Enables spam, brute-force attempts, and abuse of application functionality
- Weakens overall application security controls

→ CVSS 3.1 Base Score

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:L/A:N Base Score: 5.3 (Medium)

Risk Severity : Medium

Root Cause:

CAPTCHA logic is calculated using client-supplied values

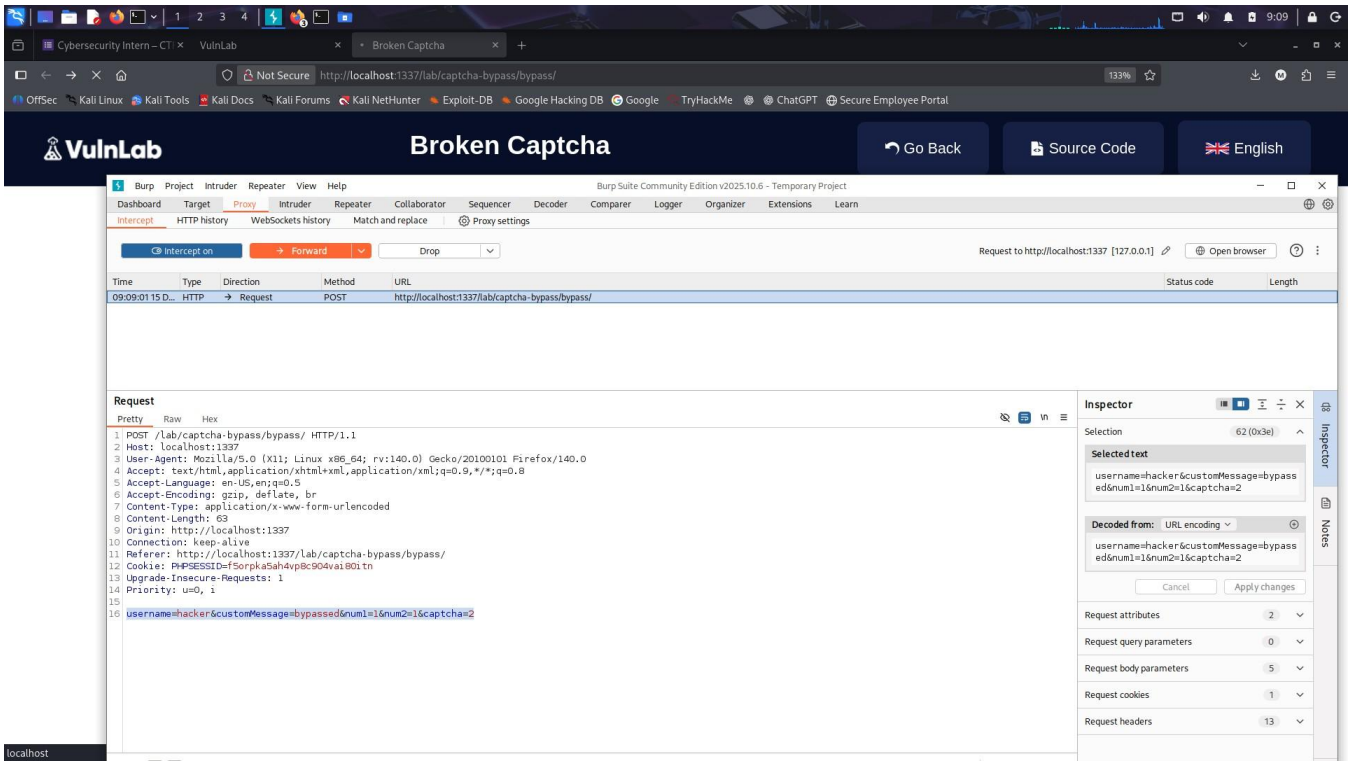
No server-side verification of CAPTCHA challenge integrity

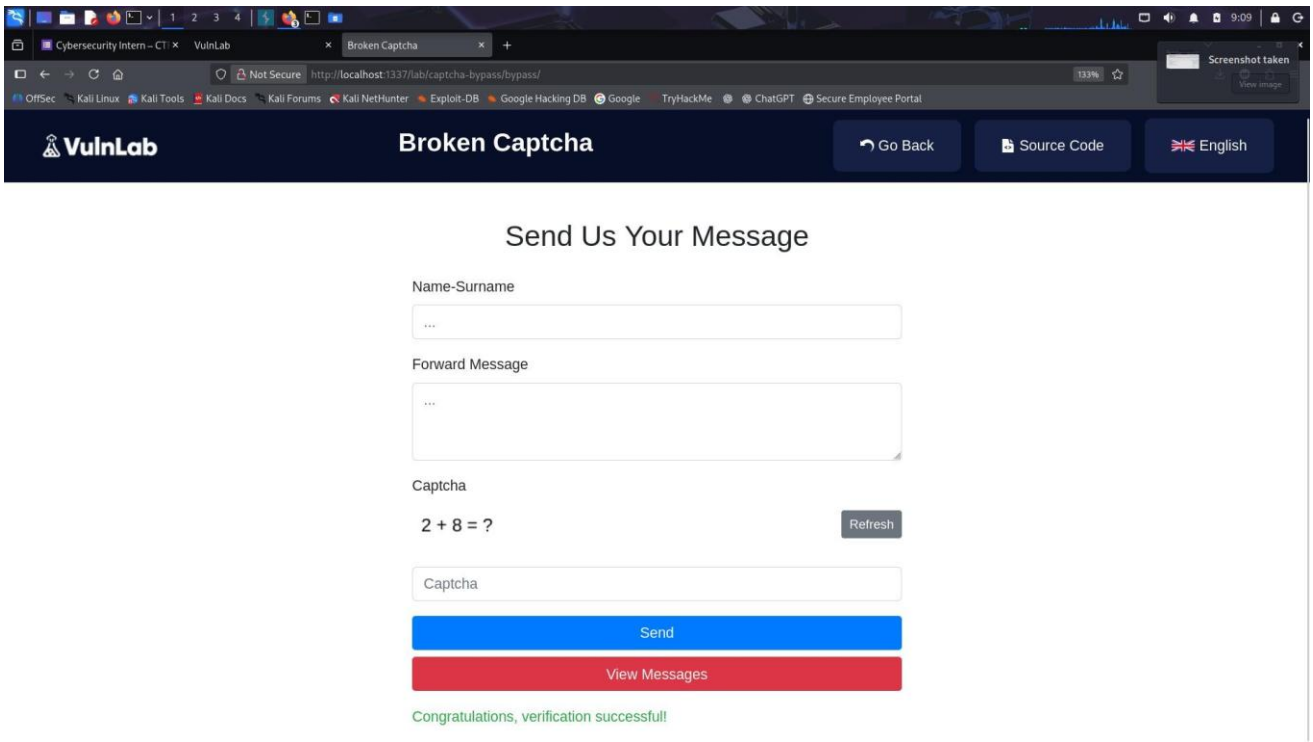
Application assumes CAPTCHA parameters sent by client are trustworthy

→ Remediation / Mitigation

- Perform CAPTCHA generation and validation entirely on the server.
- Do not accept CAPTCHA operands or results from client input
- Use secure CAPTCHA libraries (Google reCAPTCHA, hCaptcha, etc.)
- Bind CAPTCHA to session and invalidate after use

→ Screenshots / Evidence


VulnLab
Broken Captcha
Go Back
Source Code
English
Not Secure
http://localhost:1337/lab/captcha-bypass/bypass/
Burp Suite Community Edition v2025.10.6 - Temporary Project
Intercept on
Forward
Drop
Request to http://localhost:1337 [127.0.0.1]
Open browser
09:09:01 15 D... HTTP Request POST http://localhost:1337/lab/captcha-bypass/bypass/
Request
Pretty Raw Hex
POST /lab/captcha-bypass/bypass/ HTTP/1.1
Host: localhost:1337
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:140.0) Gecko/20100101 Firefox/140.0
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: 63
Origin: http://localhost:1337
Connection: keep-alive
Referer: http://localhost:1337/lab/captcha-bypass/bypass/
Cookie: PHPSESSID=f5orpka5ah4vp8c904vai8O1tn
Upgrade-Insecure-Requests: 1
Priority: u=0, i
username=hacker&customMessage=bypassed&num1=1&num2=1&captcha=2
Inspector
Selection 62 (0x3e)
Selected text
username=hacker&customMessage=bypassed&num1=1&num2=1&captcha=2
Decoded from: URL encoding
username=hacker&customMessage=bypassed&num1=1&num2=1&captcha=2
Cancel
Apply changes
Request attributes 2
Request query parameters 0
Request body parameters 5
Request cookies 1
Request headers 13
Inspector
Notes
localhost


VulnLab
Broken Captcha
Go Back
Source Code
English
Not Secure
http://localhost:1337/lab/captcha-bypass/bypass/
Screenshot taken
Send Us Your Message
Name-Surname
Forward Message
Captcha
2 + 8 = ?
Refresh
Captcha
Send
View Messages
Congratulations, verification successful!

NEXT CATEGORY- API HACKING

VULN51 - API DOCUMENTATION

Broken Object Level Authorization (BOLA) / Excessive Data Exposure via API

→ Description

The application exposes internal API endpoints without proper authorization checks. By directly accessing an API endpoint, an attacker can retrieve sensitive information such as admin user ID and credentials. Using this information, privileged API actions (like deleting files) can be performed without authentication or authorization.

This is a classic Broken Access Control issue in APIs.

→ Proof of Concept (PoC)

Step 1: Enumerate the API

While analyzing application behavior, the following endpoint was discovered by modifying the URL path:

GET /api/users.json

Step 2: Access Sensitive Data

The endpoint returned sensitive user data, including:

- Admin user ID
- Admin credentials / privileged identifiers
- No authentication or authorization token was required.

Step 3: Abuse Admin Privileges

Using the retrieved admin user ID, a privileged API request was crafted to delete a restricted file.

Example action:

Target file: delete_me.jpg

Action performed: DELETE

Result: File successfully deleted

This confirms full privilege abuse via API.

→ Impact Analysis

- Unauthorized access to admin-level data
- Exposure of sensitive credentials and identifiers
- Ability to perform destructive actions (file deletion)
- Complete compromise of application integrity
- High risk of data loss and privilege escalation

→ CVSS 3.1 Base Score

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

Base Score: 9.8 (Critical)

Risk Rating : Critical

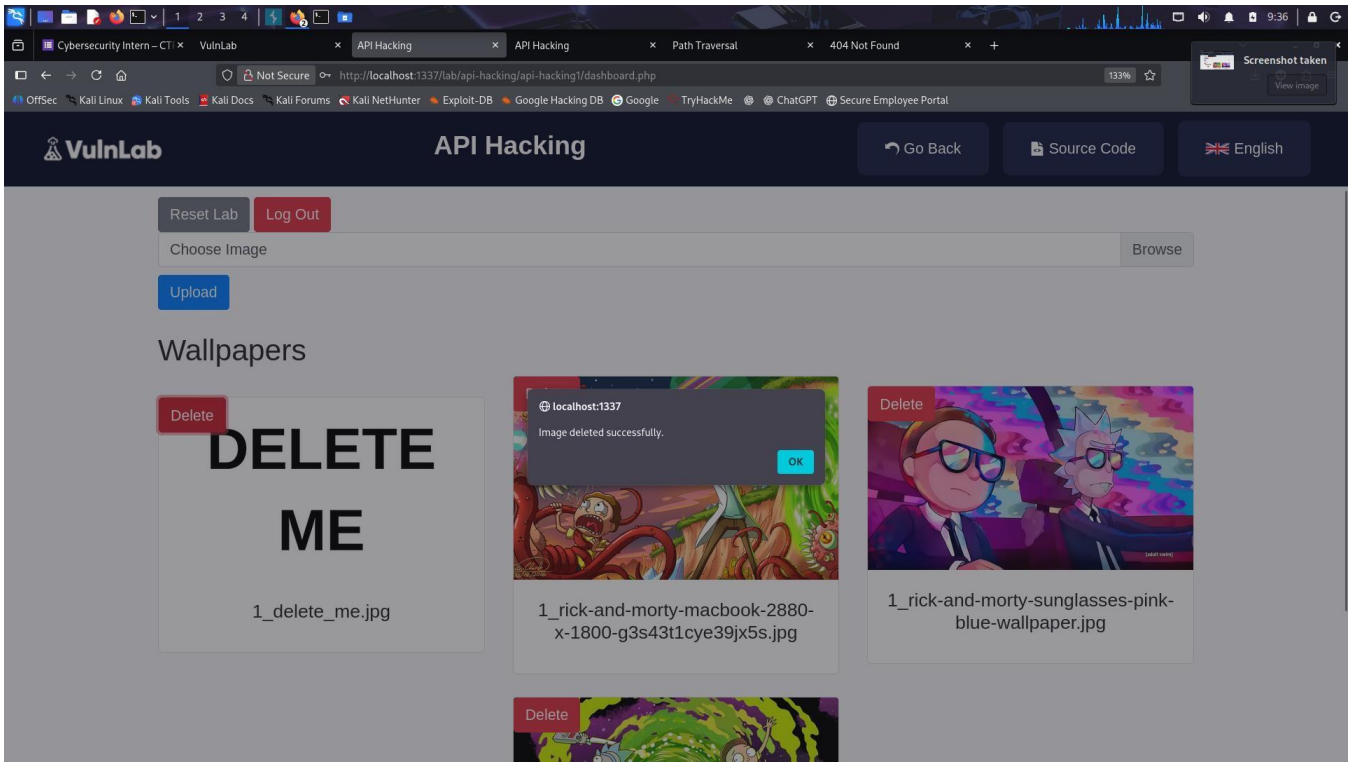
Root Cause:

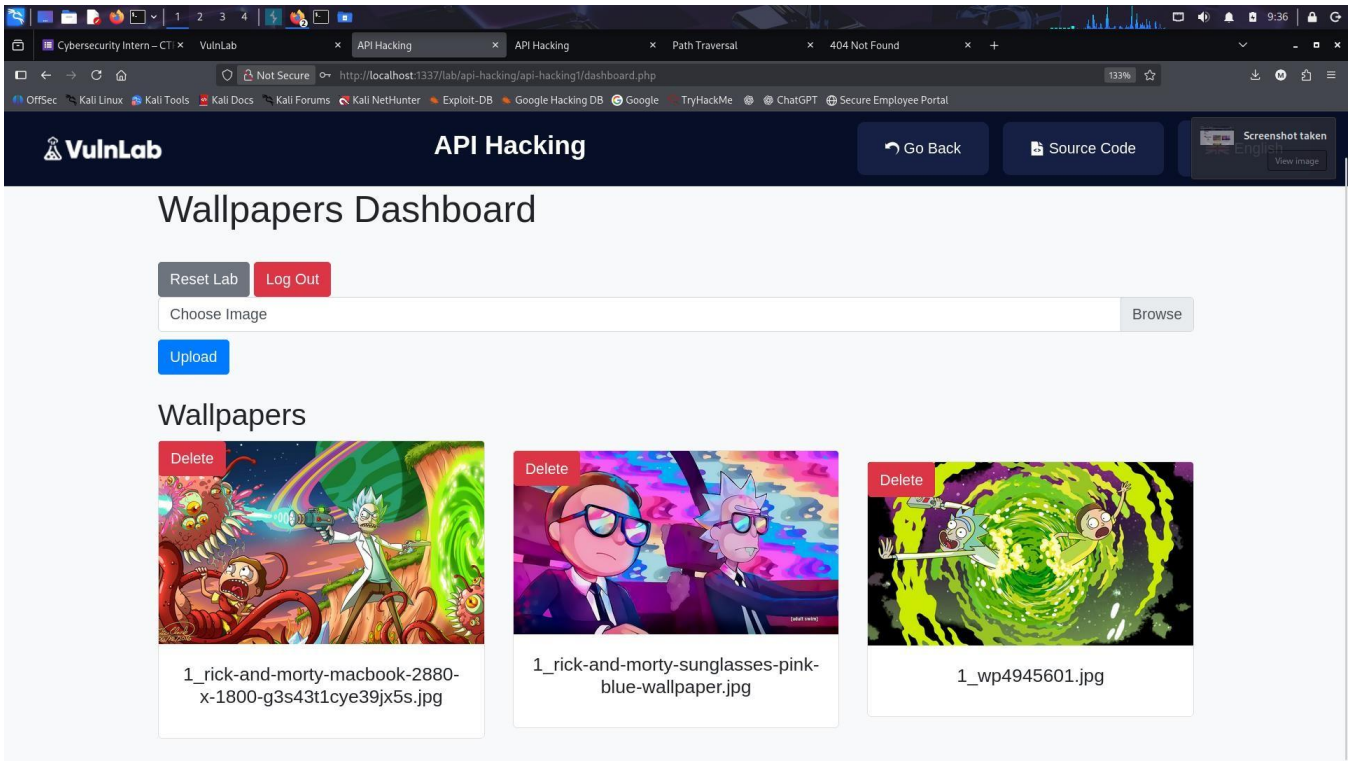
- Missing authentication on API endpoints
- No authorization checks for sensitive resources
- Direct object references exposed via predictable API paths
- Excessive data exposure in API responses

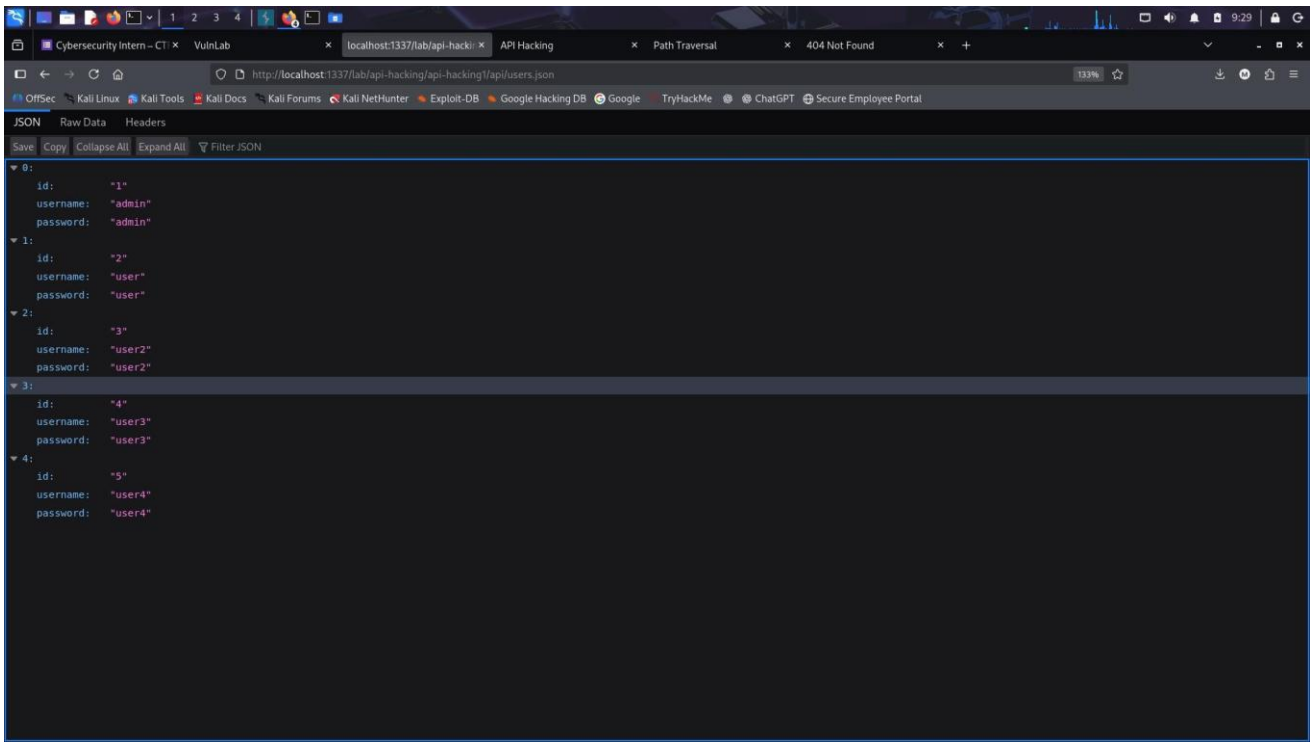
→ Remediation / Mitigation

- Enforce authentication on all API endpoints
- Implement strict role-based access control (RBAC)
- Do not expose sensitive user data (IDs, credentials) in API responses
- Validate user permissions before performing destructive actions
- Use access tokens (JWT/OAuth) and server-side authorization checks

→ Screenshots / Evidence to Attach




VulnLab
API Hacking
Go Back
Source Code
Screenshot taken
Wallpapers Dashboard
Reset Lab
Log Out
Choose Image
Browse
Upload
Wallpapers
Delete
1_rick-and-morty-macbook-2880-x-1800-g3s43t1cye39jx5s.jpg
Delete
1_rick-and-morty-sunglasses-pink-blue-wallpaper.jpg
Delete
1_wp4945601.jpg


JSON
Raw Data
Headers
Save
Copy
Collapse All
Expand All
Filter JSON
0:
id: "1"
username: "admin"
password: "admin"
1:
id: "2"
username: "user"
password: "user"
2:
id: "3"
username: "user2"
password: "user2"
3:
id: "4"
username: "user3"
password: "user3"
4:
id: "5"
username: "user4"
password: "user4"

NEXT CATEGORY /LAST CATEGORY- PATH TRAVERSAL

VULN 52- PATH TRAVERSAL BLACKLIST

Path Traversal (Blacklist Bypass)

→ Description

The application attempts to prevent directory traversal using a blacklist-based filter. However, the filter can be bypassed by using obfuscated traversal sequences. By manipulating the productId parameter with non-standard traversal patterns, an attacker can access arbitrary files on the server outside the intended directory.

→ Affected Endpoint

GET /lab/path-traversal/path-traversal-2/product-detail.php?productId=<payload>

→ Proof of Concept (PoC)

Injected Payload:

....//....//....//....//etc/passwd

Modified Request Line:

GET /lab/path-traversal/path-traversal-2/product-detail.php?productId=....//....//....// //etc/passwd HTTP/1.1

Result:The server returned HTTP/1.1 200 OK, indicating the blacklist filter was bypassed and the request was processed successfully, confirming path traversal via obfuscated sequences.

Steps to Reproduce

1. Navigate to the Path Traversal Blacklist lab page.
2. Intercept the request in Burp Suite.
3. Modify the productId parameter using the obfuscated traversal payload shown above.
4. Forward the request to the server.
5. Observe a 200 OK response, confirming traversal bypass.

→ Impact Analysis

Successful exploitation allows attackers to:

- Read sensitive system files (e.g., /etc/passwd)
- Access application configuration files
- Potentially obtain credentials or secrets
- Aid further attacks such as privilege escalation or RCE

→ CVSS 3.1 Base Score

7.5 (High)

Vector:

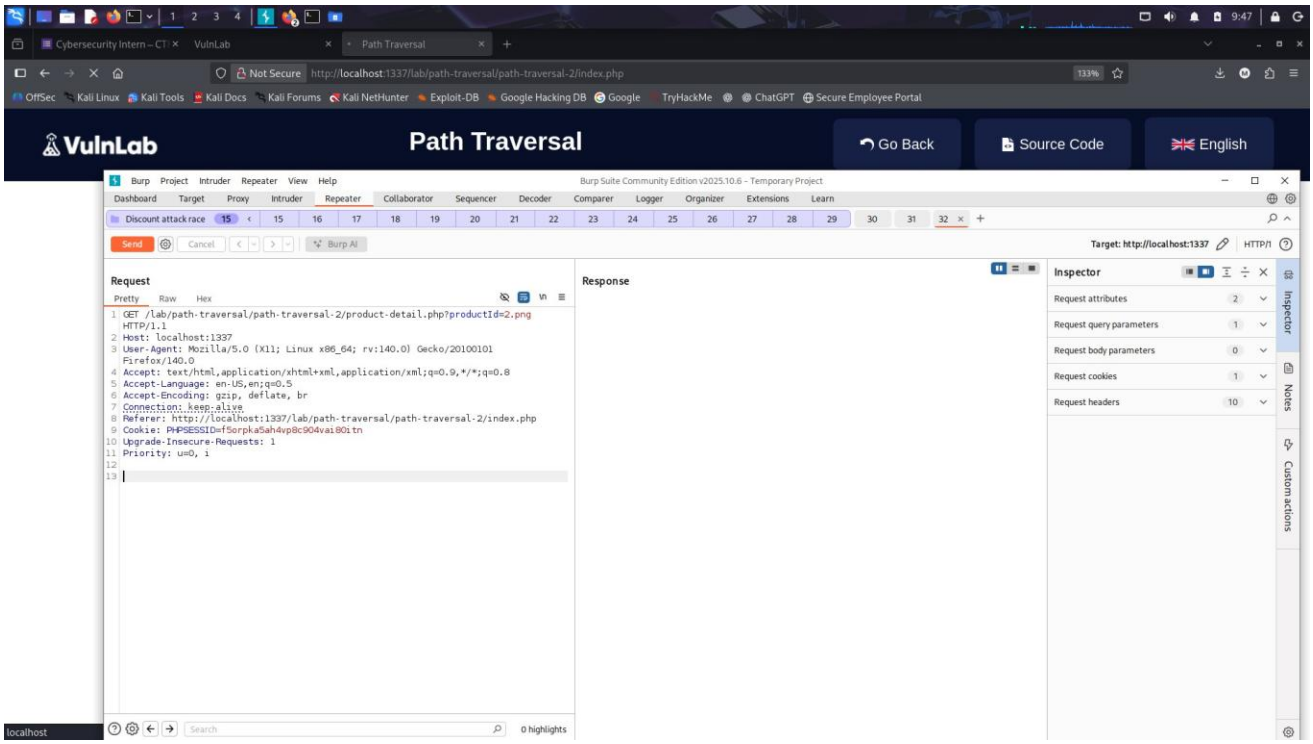
CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N

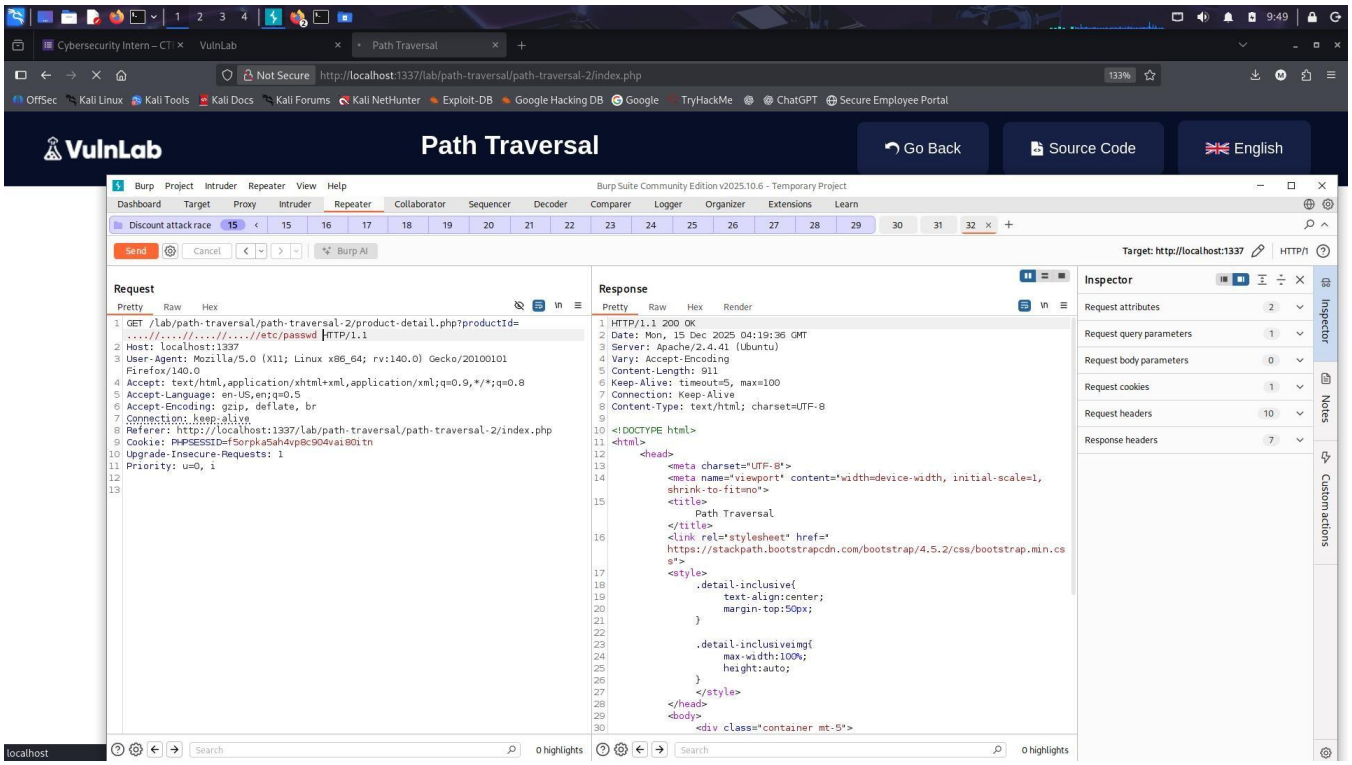
Risk Rating : High

→ Remediation / Mitigation

- Avoid blacklist-based filtering for file paths.
- Use strict allowlists of permitted files or IDs.
- Normalize and canonicalize paths before access.
- Enforce server-side path validation using realpath() or equivalent.
- Store files outside the web root where possible.

→ Evidence / Screenshot





Conclusion

This assessment provided comprehensive, hands-on exposure to real-world web application security flaws across **15 vulnerability categories and 52 practical labs**. Each lab was approached from an attacker's perspective, focusing on understanding the root cause of the vulnerability, reliably reproducing it, and documenting its real-world impact using industry-standard VAPT methodology. The exercise strengthened my ability to think beyond automated tools, analyze application logic, manipulate requests, and validate security weaknesses under realistic constraints. More importantly, it reinforced disciplined reporting practices, including clear PoCs, impact assessment, CVSS scoring, and actionable remediation guidance. Completing this assessment end-to-end reflects not just technical capability, but consistency, persistence, and a strong security mindset with the qualities essential for effective work in VAPT, SOC, and offensive security roles.