Access Control Testing

Access control testing involves verifying that the security measures that regulate who or what can view or use resources in a computing environment are functioning as intended. The goal is to ensure that users have the appropriate level of access to resources based on their permissions and that unauthorized access is effectively prevented.

Creating Access Control Testing

When conducting access control testing, you typically:

- Ensure that authentication mechanisms are secure and cannot be bypassed.
- Check that authorization schemes implement role-based or other access control models correctly.
- Verify that privilege escalation cannot occur where a lower-privileged user improperly gains higher access rights.
- Confirm that access to sensitive operations (like administrative functions) is appropriately restricted.

What Will We Do?

In this lesson, we will use prompt engineering to aid in creating access control tests. We will be supplied Python code with no access control unit tests. We must prompt the AI to create some tests to ensure that our code is free from security vulnerabilities.

Instructions

Below is a Python script with known access control vulnerabilities.

```
from flask import Flask, request, session, redirect, url_for

app = Flask(__name__)
app.secret_key = 'super_secret_key'

# This is a mock database of users with their roles
users_db = {
    'admin': {'password': 'adminpass', 'role': 'admin'},
    'user': {'password': 'userpass', 'role': 'user'}
}

@app.route('/login', methods=['POST'])
def login():
    username = request.form['username']
    password = request.form['password']
    user = users_db.get(username)
    # Simple authentication (passwords should be hashed in production)
    if user and user['password'] == password:
        session['username'] = username
        session['role'] = user['role']
        return redirect(url_for('index'))
    return 'Invalid credentials', 401
```

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

@app.route('/')
def index():
    if 'username' in session:
        return f'Hello, {session["username"]}!'
    return 'You are not logged in.'

@app.route('/admin')
def admin_panel():
    # Vulnerable access control: no role check
    if 'username' in session:
        return 'Admin Panel - only for admins'
    return redirect(url_for('index'))

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

Your task is to prompt the AI chat to develop unit tests for the Python script. **You will know you have successfully completed the task when you obtain the following results from the machine:**

- Using the Python unittest framework
- Using the Python requests library
- Using the Python threading library
- A test that starts the Flask server in a new thread
- A test that stops the Flask server and waits for the thread to finish
- A test that ensures Admin have access to the admin panel
- A test that ensures regular user should not have access to the admin panel
- A test that ensures unauthenticated users should not have access to the admin panel
- A test that ensures that after logging out, the Admin should not have access to the admin panel

Note: Depending on your prompt, you may receive different results than anticipated. That is okay. However, there should be at least four (4) tests.

If you would like some help creating prompts to send to the AI, open the hint dropdown below.

Prompt Hint

The AI remembers previous messages so you can chat back and forth like a conversation. You may need to use multiple prompts, but combining prompts saves computation time and energy. Here are some examples below:

- Create unit tests for the following Python code using the unittest, requests, and threading libraries: <paste in the code from above>

- There should be a test which starts a Flask server in a new thread and a test which stops a Flask server and waits for the thread to finish.
- Add tests which ensure Admins have access to the admin panel, while regular and unauthenticated users do not have access to the admin panel and add a test that checks that after logging out, the Admin should not have access to the admin panel.

If you need the AI to forget past prompts, click the trash can icon and the chat will reset.

Prompt

Create unit tests for the following Python code using the unittest, requests, and threading libraries:

```
from flask import Flask, request, session, redirect, url_for

app = Flask(__name__)
app.secret_key = 'super_secret_key'

# This is a mock database of users with their roles
users_db = {
    'admin': {'password': 'adminpass', 'role': 'admin'},
    'user': {'password': 'userpass', 'role': 'user'}
}

@app.route('/login', methods=['POST'])
def login():
    username = request.form['username']
    password = request.form['password']
    user = users_db.get(username)
    # Simple authentication (passwords should be hashed in production)
    if user and user['password'] == password:
        session['username'] = username
        session['role'] = user['role']
        return redirect(url_for('index'))
    return 'Invalid credentials', 401

@app.route('/logout')
```

```
def logout():
    session.clear()
    return redirect(url_for('index'))


@app.route('/')
def index():
    if 'username' in session:
        return f'Hello, {session["username"]}!'
    return 'You are not logged in.'


@app.route('/admin')
def admin_panel():
    # Vulnerable access control: no role check
    if 'username' in session:
        return 'Admin Panel - only for admins'
    return redirect(url_for('index'))


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

There should be a test which starts a Flask server in a new thread and a test which stops a Flask server and waits for the thread to finish.

Add tests which ensure Admins have access to the admin panel, while regular and unauthenticated users do not have access to the admin panel and add a test that checks that after logging out, the Admin should not have access to the admin panel.

Response

To create unit tests for the given Python code, we need to use the unittest, requests, and threading libraries. We will use the FlaskTestClient provided by Flask for testing Flask routes.

Here's an outline of the unit tests that need to be created:

1. **Testing Flask server setup and teardown:**
    - Test that starts a Flask server in a new thread.
    - Test that stops the Flask server and waits for the thread to finish.
2. **Testing access to admin panel:**
    - Test that ensures admins have access to the admin panel.

- Test that ensures regular users and unauthenticated users do not have access to the admin panel.
- Test that checks that after logging out, the Admin should not have access to the admin panel.

These tests will require mocking user sessions and making requests to the server using the requests library.

Here is a basic outline of how the unit tests could be implemented in Python using unittest:

```
import unittest

import requests

import threading

from flask import Flask


class TestFlaskApp(unittest.TestCase):

    def setUp(self):

        self.server_thread = threading.Thread(target=app.run, kwargs={'debug': False})

        self.server_thread.start()


    def tearDown(self):

        requests.get('http://127.0.0.1:5000/shutdown')

        self.server_thread.join()


    def test_admin_access(self):

        # Test admin access to admin panel

        with requests.Session() as session:

            response = session.post('http://127.0.0.1:5000/login', data={'username': 'admin', 'password': 'adminpass'})

            self.assertEqual(response.status_code, 302)  # Redirect expected

            response = session.get('http://127.0.0.1:5000/admin')

            self.assertEqual(response.text, 'Admin Panel - only for admins')


    def test_user_access(self):

        # Test non-admin access to admin panel

        with requests.Session() as session:
```

```
        response = session.post('http://127.0.0.1:5000/login', data={'username': 'user', 'password': 'userpass'})
        self.assertEqual(response.status_code, 302)  # Redirect expected
        response = session.get('http://127.0.0.1:5000/admin')
        self.assertEqual(response.url, 'http://127.0.0.1:5000/')  # Redirected to index

    def test_logout_admin(self):
        # Test admin access after logout
        with requests.Session() as session:
            response = session.post('http://127.0.0.1:5000/login', data={'username': 'admin', 'password': 'adminpass'})
            self.assertEqual(response.status_code, 302)  # Redirect expected
            session.get('http://127.0.0.1:5000/logout')
            response = session.get('http://127.0.0.1:5000/admin')
            self.assertEqual(response.url, 'http://127.0.0.1:5000/')  # Redirected to index

if __name__ == '__main__':
    unittest.main()
```

Please note that you may need to modify the shutdown mechanism in the Flask app code in order to cleanly stop the server during testing. Also, make sure to adjust port numbers and URLs in the tests to match your Flask server configuration.