Flask

NCAE

Outline

- Introduction to Flask webservers
- Flask + SQL
- Flask templating
- Common Flask/Python vulns

Introduction to Flask Servers

Example Code

- `pip3 install flask` (dependency)
- Endpoint ➜ function (like `hello_world()`)
- `Flask()` object is webserver, `app.run()` starts
- `@app.route(“/”)` ➜ when endpoint `/` is requested, run the next defined function

```python
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello_world():
    return "<p>Hello, World!</p>"

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

Flask vs. Werkzeug

- Flask is a Python webserver
- Relies on Python Werkzeug library (may see this mentioned in multiple places)
- Mostly under the hood
- Werkzeug also provides simple HTTP server along with library
- *Note - Django is another common Python webserver*

Flask Features

- You can specify valid HTTP methods for each endpoint
- You can have variable paths, like below, and that variable is passed in as function argument
- You can add extra decorators (like token_required) so that custom function is ran before get_botnet_order
    - These functions usually enforce authentication, such as checking that a session cookie is valid

```
# GET botnet order information
@app.route('/api/botnet-orders/<int:order_id>', methods=['GET'])
@token_required
def get_botnet_order(session_data, order_id):
    # get user_id from order_id
```

Flask Features

- Also a generic way to run a function before or after EVERY endpoint

```python
from flask import Flask

app = Flask(__name__)
@app.before_request
def before_request_func():
    print("before_request executing!")

@app.after_request
def after_request_func(response):
    print("after_request executing!")
    return response

@app.route("/")
def index():
    print("Index running!")

if __name__ == "__main__":
    app.run()
```

Flask Blueprints

- If you have MANY endpoints, you get a LARGE `server.py` file
- Blueprints were added to allow for splitting code between files & organize
- Bottom code snippet is saved in `./example_blueprint.py`

```python
from flask import Flask
from example_blueprint import example_blueprint

app = Flask(__name__)
app.register_blueprint(example_blueprint)
```

```python
from flask import Blueprint

example_blueprint = Blueprint('example_blueprint', __name__)

@example_blueprint.route('/')
def index():
    return "This is an example app"
```

Flask Requests and Responses

- Endpoint functions can access HTTP request data through the request variable
  - This includes headers, GET parameters, POST parameters, cookies, etc.
- Endpoints send responses as the return value
- You can return a string, or create a `Response()` object and set attributes for more control

```python
@app.route('/getpost', methods=['GET'])
def getpost():
    # get variables
    postid = request.args["id"]
```

```python
@app.route('/flag', methods=['POST'])
def main():
    resp = make_response("Nope")
    resp.status_code = 401
```

Flask + SQL

SQLite

- Like any webserver, Flask can use any DBMS (MySQL, PostgreSQL, etc.)
- SQLite is commonly used
- SQLite is not a service, all the data is stored in a file
- Connect: `conn = sqlite3.connect('local.db')`
- Cursor: `cur = con.cursor()`
- Run query: `cur.execute('CREATE TABLE …')`
- Commit: `conn.commit()` (only when making changes)
- Get SELECT data: `rows = cur.fetchall()`

Preventing SQL Injection

Vulnerable

- `db.exec('SELECT * FROM users WHERE userId=' +input_id+ ';');`
- `db.exec(f'SELECT * FROM users WHERE userId={input_id};');`
- `db.exec('SELECT * FROM users WHERE userId=%s;' % input_id);`

Safe

- `db.exec('SELECT * FROM users WHERE userId=%s;', (input_id,));`

Flask Templating

Templating

- POV - you're a dev using Flask and want to render HTML pages, how??
- Use Jinja2 templating engine for both static HTML pages and dynamic HTML pages
- Static - `render_template('page.html')` (*`page.html` must be located in `./templates/page.html` folder*)
- Dynamic - `render_template('page.html', user=request.json()['user'])`
  - Inside `page.html`, you'll see code like `{{ user }}` where data is pasted in
  - Jinja2 is safe from XSS ***by default*** unless it uses `{{ user | safe }}`, then it DOESN'T ESCAPE
- Other methods (hard-coded HTML, opening/returning files) can be unsafe

Server-Side Template Injection

- Since dynamic HTML pages contain code like `{{ user }}` where `user` is an actual Python variable, can be dangerous
- Never let user-controlled data be treated as template
- Example 1 - `render_template_string(user_input)`
- Example 2 - modify a file, then `render_template('that_file.html')`
- Allows attacker to run arbitrary Python and achieve RCE

Common Flask/Python Vulns

Flask Debug Mode

- `app.run(debug=True)` is BAD
- Verbose error messages (not basic 500 Internal Server messages)
- Error messages leak source code (BAD)
- Opens up `/console` endpoint
  - 9 digit pin allows anyone to run arbitrary Python (RCE)
  - Well-known method of obtaining pin using LFI (link)

POST Parameter Data Types

- `param1 = request.json()['param1']`
- `param1` could be boolean, string, integer, array, or object
- Unexpected data type leads to unexpected behavior
- Check type with `isinstance(request.json()['param1'], str)`
- *Note - `request.form['param1']` I believe is always string*

Flask Cookies

- Flask has built-in session cookies that use cookie name session by default
- Cookies are made up of 3 parts - 1x base64-encoded JSON section + 2x signature pieces, split up by periods
  - `eyJsb2dnZWRfaW4iOmZhbHNlfQ.XDuWxQ.E2Pyb6x3w-NODuflHoGnZOEpbH8`
- Signature is verified using a secret, initialized as `app.secret_key = '<secret>'` (link)
- If secret is not random/guessable, you can forge session cookies (link)
- JWT is not Python- or Flask-specific, but commonly used
- Similar format, but 2x base64-encoded JSON sections + 1x signature piece
  - `eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c`
- If you see this, ensure secret is random + consult HackTricks

Path Traversal

- Biggest unintuitive behavior that leads to path traversal
- `os.path.join()` combines 2 strings
- ie, `os.path.join('/etc', 'shadow') = '/etc/shadow'`
- HOWEVER
- `os.path.join('/etc', '/tmp/hakt.txt') = '/tmp/hakt.txt'`
- Absolute address ALWAYS takes precedence
- No `../` necessary

Deserialization

- Serializing objects allows you to transfer objects
- JSON, YAML, XML, and CSV are all serialization formats
- Python has unique one called **pickle**
- Deserialization pickles is called unpickling
- Unpickling arbitrary data can lead to RCE
- If user-supplied pickles are ran through `pickle.loads()`, VERY BAD RCE

SSRF

- Assuming you already know what SSRF is
- The easiest way in Python to send arbitrary HTTP is using `requests` module
- If you see `import requests` in any of the Python code files for flask server, RED ALERT
- Check out where requests is being used

Other Flask Server Examples

- https://github.com/BYU-CSA/BYUCTF-2023/tree/main/urmombotnetdotnet.com
- https://github.com/BYU-CSA/BYUCTF-2023/tree/main/notes
- https://github.com/BYU-CSA/old-ctf-challenges/tree/master/web/socialmedia2
- https://github.com/BYU-CSA/old-ctf-challenges/tree/master/web/fragment