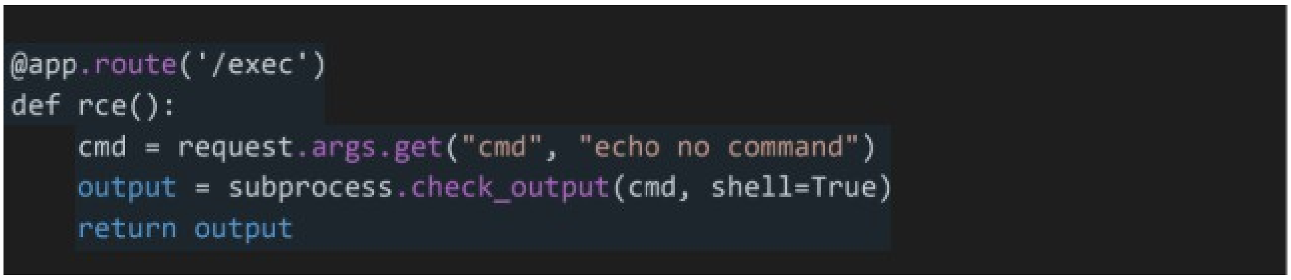
Name: S.Dhanish Ahamed
Reg No: 25BCE0001

Python Code Review

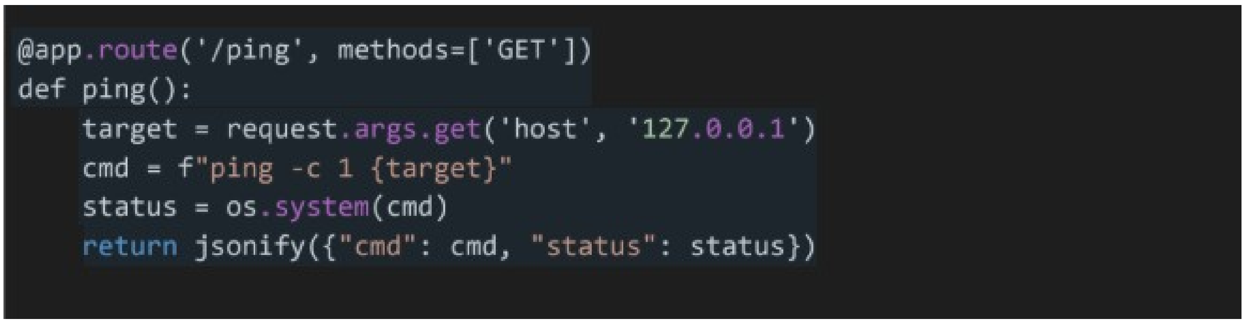
1) Remote Code Execution & Command Injection

- The /exec endpoint takes a cmd parameter and passes it directly to subprocess.check_output with shell=True. An attacker can run any command.
- Fix: Remove this webpoint entirely. There is rarely need for such functionality in a web app.



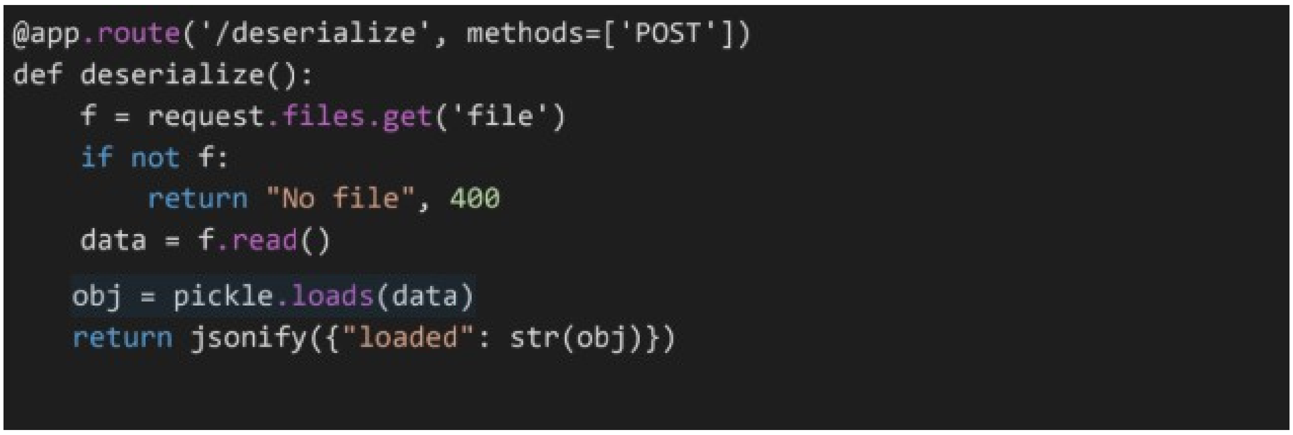
```
@app.route('/exec')
def rce():
    cmd = request.args.get("cmd", "echo no command")
    output = subprocess.check_output(cmd, shell=True)
    return output
```

- The /ping endpoint inserts user input directly into a shell command string. An attacker can append commands using delimiters.
- Fix: Validate that the input is an ip address using ipaddress library



```
@app.route('/ping', methods=['GET'])
def ping():
    target = request.args.get('host', '127.0.0.1')
    cmd = f"ping -c 1 {target}"
    status = os.system(cmd)
    return jsonify({"cmd": cmd, "status": status})
```

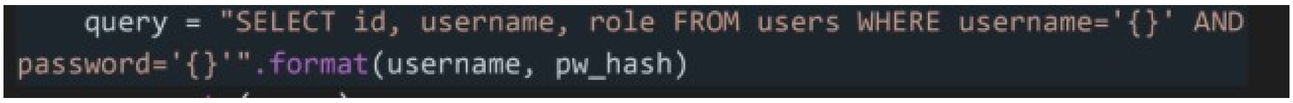
- The /deserialize endpoint accepts a file and passes its contents to “pickle.loads”. Pickle allows arbitrary code execution during deserialization. An attacker can craft a malicious pickle file that runs code when loaded
- Fix: Never use pickle on untrusted data



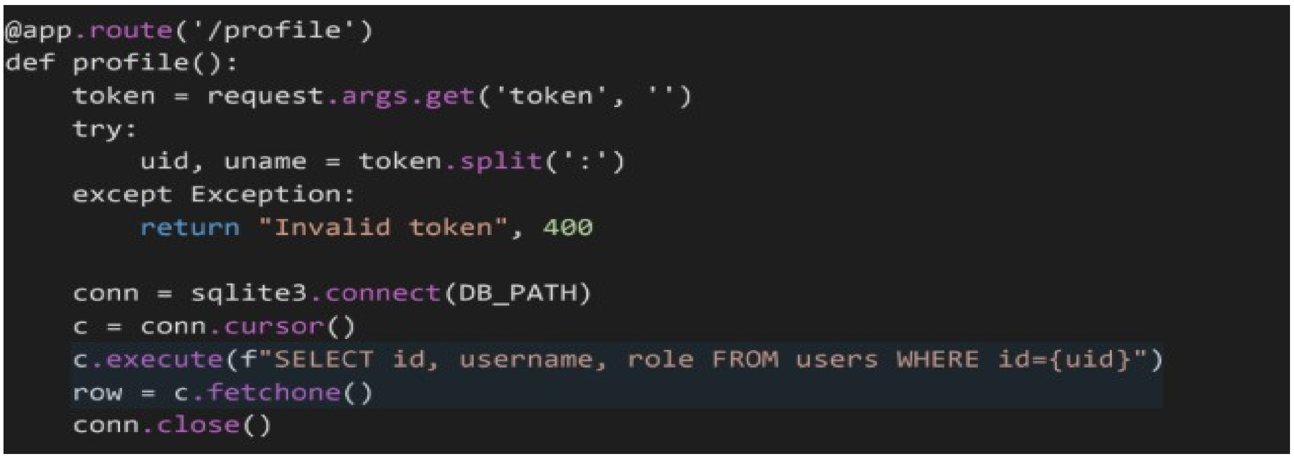
```python
@app.route('/deserialize', methods=['POST'])
def deserialize():
    f = request.files.get('file')
    if not f:
        return "No file", 400
    data = f.read()
    obj = pickle.loads(data)
    return jsonify({"loaded": str(obj)})
```

2) SQL Injection

- The application constructs SQL queries using string formatting with unsanitized user input
- The login query inserts username directly into the SQL string.
- An attacker can login as any user without a password (e.g. ‘ OR ‘1’=’1’)
- The /profile endpoint injects the uid from the token into a query
- /super_admin endpoint injects username
- Use parametrized queries to fix this vulnerability

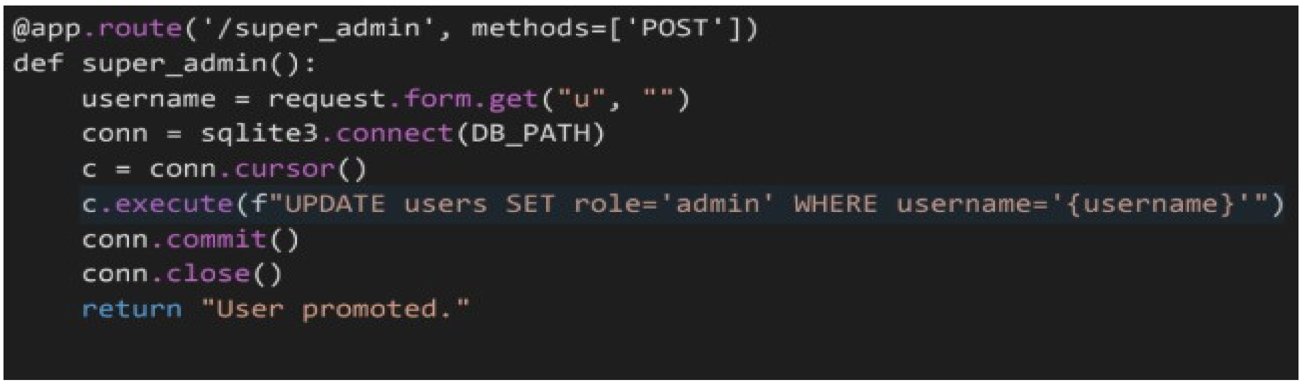


```python
    query = "SELECT id, username, role FROM users WHERE username='{}' AND
password='{}'".format(username, pw_hash)
```



```python
@app.route('/profile')
def profile():
    token = request.args.get('token', '')
    try:
        uid, uname = token.split(':')
    except Exception:
        return "Invalid token", 400

    conn = sqlite3.connect(DB_PATH)
    c = conn.cursor()
    c.execute(f"SELECT id, username, role FROM users WHERE id={uid}")
    row = c.fetchone()
    conn.close()
```

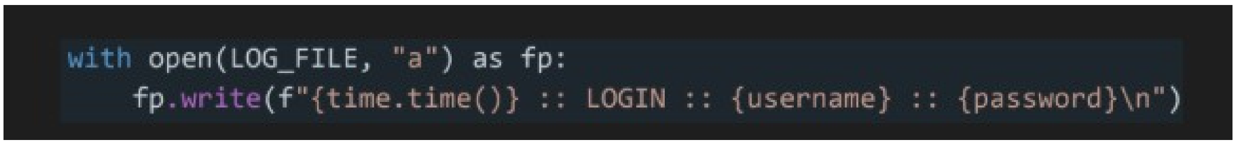


```python
@app.route('/super_admin', methods=['POST'])
def super_admin():
    username = request.form.get("u", "")
    conn = sqlite3.connect(DB_PATH)
    c = conn.cursor()
    c.execute(f"UPDATE users SET role='admin' WHERE username='{username}'")
    conn.commit()
    conn.close()
    return "User promoted."
```

3) Broken Authentication and Session management

1) Logging Credentials: The application writes plain-text passwords to a log file.

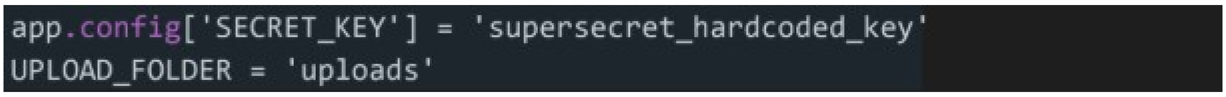
- Risk: If the log file is accessed (e.g., via the Local File Inclusion vulnerability below), attackers gain all user passwords.
- Fix: Never log passwords or sensitive data.



```python
with open(LOG_FILE, "a") as fp:
    fp.write(f"{time.time()} :: LOGIN :: {username} :: {password}\n")
```

2) Hardcoded Secret Key: The application uses a hardcoded secret key.

- Risk**:** If tokens were signed (which they aren't here, but in general), anyone with the code could forge valid sessions.
- Fix: Load secrets from environment variables.

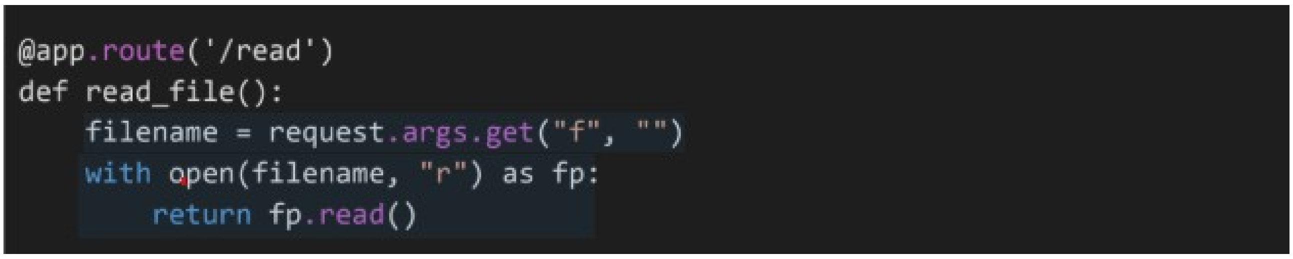


```python
app.config['SECRET_KEY'] = 'supersecret_hardcoded_key'
UPLOAD_FOLDER = 'uploads'
```

4. File System Vulnerabilities

1) Arbitrary File Read (Path Traversal): The /read endpoint opens any filename provided by the user.

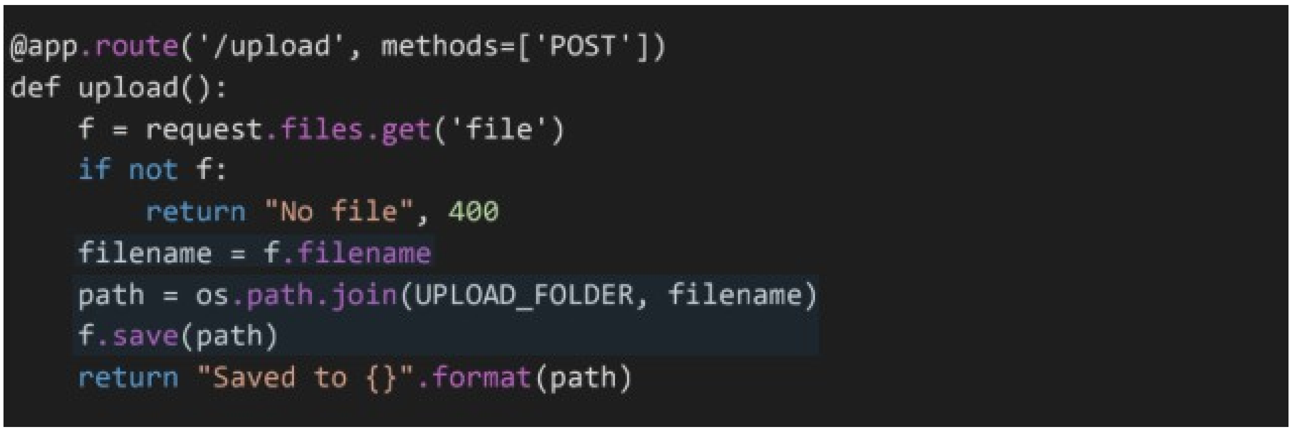
- Risk: Attackers can read sensitive system files (e.g., /etc/passwd, auth.log, source code).
- Fix: Validate that the requested file exists within a specific allowed directory and does not contain traversal characters (..)



```python
@app.route('/read')
def read_file():
    filename = request.args.get("f", "")
    with open(filename, "r") as fp:
        return fp.read()
```

2) Unrestricted File Upload: The /upload endpoint saves files using the user-provided filename without validation.

- Risk: Attackers can upload malicious scripts (e.g., webshells) or overwrite critical system files by using paths like ../../overwrite_me.
- Fix: Use werkzeug.utils.secure_filename and validate the file extension/content type.



```
@app.route('/upload', methods=['POST'])
def upload():
    f = request.files.get('file')
    if not f:
        return "No file", 400
    filename = f.filename
    path = os.path.join(UPLOAD_FOLDER, filename)
    f.save(path)
    return "Saved to {}".format(path)
```