TableCheck

Security Assessment Report (Flask App)

February 2021, 2025

Document Control

Date	Change	Changed by	Version
21 Feb 2024	Initial Report	Avelino Dahotoy III	1.0

Contents

Executive Summary

The Executive Summary outlines the key findings derived from the SCA (Software Composition Analysis) and SAST (Static Application Security Testing) scans conducted on the application repository, as well as the results observed during runtime testing. It provides an overview of the key security and vulnerability issues identified, with a breakdown of the SAST and SCA scan results included in the summary. Detailed findings, including specific vulnerabilities and their potential impact, will be addressed in the Findings and Recommendations section for further review and action.

Risk Assignment

Testing discovered a total of 7 unique findings. After a thorough analysis, these findings have been rated at the following risk levels:

Key Findings Summary:

RISK LEVEL	COUNT
CRITICAL	3
HIGH	1
MEDIUM	1
LOW	1

Key Findings by Class and Severity:

Finding	Severity
SQL Injection Vulnerability	Critical
Unrestricted File Uploads	Critical
Exposed and Hardcoded Secret Key and Password	High
Log files exposure and insecure logging (/logs Route)	High
Broken Access Control (/users Route)	Medium
Project Dependency File must contain version limitations	Low

Software Composition Analysis (SCA):

Critical	High	Medium	Low
0	1	0	1

Static Application Security Testing (SAST):

Critical	High	Medium	Low
0	1	0	0

Risk Rating are calculated using CVSS Version 3.1 Calculator
Scoring Matrix: https://nvd.nist.gov/vuln-metrics/cvss

Vulnerability Assessment

Key Findings:

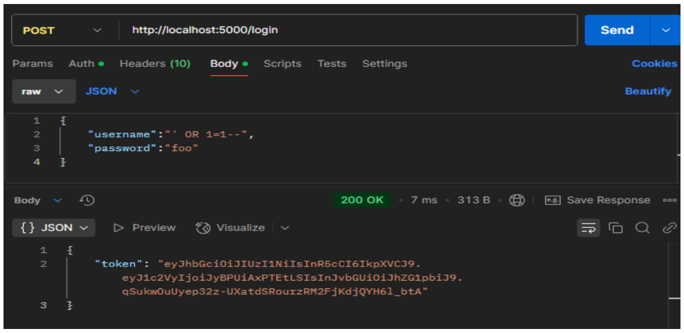
Risk Level	Critical
Vulnerability Class	SQL Injection Vulnerability
CVSS v3.1 Rating	9.2 (AV:L, AC:L, PR:N, UI:N, S:U, C:H,I:L, AH)
Location	http://localhost:5000/login

Description

Query is not parameterized and easy for attacker to pass in SQLi attack.

Evidence

Steps to replicate: Send an HTTP **POST** Request to http://localhost:5000/login with payload : {"username":"' OR 1=1--","password":"foo"}



POST http://localhost:5000/login

```
{
    "username":"' OR 1=1--",
    "password":"foo"
}
```

```
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
    eyJ1c2VyIjoiJyBPUiAxPTEtLSIsInJvbGUiOiJhZG1pbiJ9.
    qSukwOuUyep32z-UXatdSRourzRM2FjKdjQYH6l_btA"
}
```

Recommendations	• Use parameterized query to prevent SQL injection • Add firewall SQLi injection protection.
References	https://owasp.org/www-community/attacks/SQL_Injection https://attack.mitre.org/techniques/T1190/ **SOC 2 Type 2**: Criterion CC3.1 (Risk Mitigation) **ISO 27001**: Control A.14.2.5 (Secure system engineering principles)

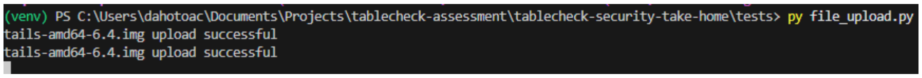
Risk Level	Critical
Vulnerability Class	Unrestricted File Uploads
CVSS v3.1 Rating	9.2 (AV:L, AC:L, PR:N, UI:N, S:U, C:H,I:L, AH)
Location	http://localhost:5000/upload

Description

- No file type validation. Attacker can upload malicious files since the endpoint is not restricting any file types.
- No file upload limit set. Attacker can send a large volume of file upload request to overload the server's resources.
- No file size limit
- No authentication or authorization when uploading files

Evidence

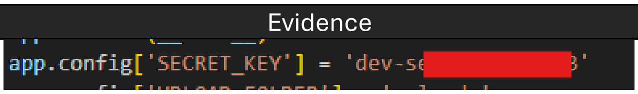
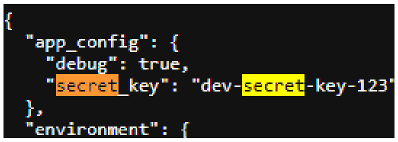
Steps to reproduce: Try upload multiple files with different file types.

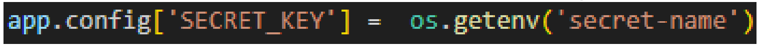


```
(venv) PS C:\Users\dahotoac\Documents\Projects\tablecheck-assessment\tablecheck-security-take-home\tests> py file_upload.py
tails-amd64-6.4.img upload successful
tails-amd64-6.4.img upload successful
```

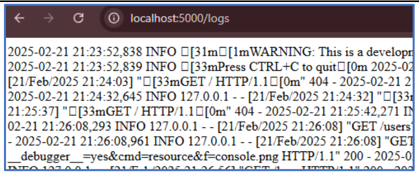
Recommendations	- Set file upload limit so that the attackers can’t abuse the file upload feature. - Set file type validation and add rate limit to the file upload endpoint.
References	https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload **ISO 27001**: Control A.14.2.6 (Secure development) **SOC 2 Type 2**: Criterion CC6.4 (Security controls)

Risk Level	Critical
Vulnerability Class	Exposed and Hardcoded Secret Key and Password
CVSS v3.1 Rating	9.2 (AV:L, AC:L, PR:N, UI:N, S:U, C:H,I:L, AH)
Location	- https://github.com/TableCheck-Labs/tablecheck-security-take-home/blob/main/app/app.py - http://localhost:5000/export
Description	
- Secrets should not be hardcoded in the code as it can be committed to the repository and it stays in the commit history. People who have access the repository even in view mode can see the secrets. - HTTP responses should not include environment secrets.	
Evidence	


`app.config['SECRET_KEY'] = 'dev-se[redacted]3'`

`{`
`  "app_config": {`
`    "debug": true,`
`    "secret_key": "dev-secret-key-123"`
`  },`
`  "environment": {`	

Recommendations	- Use environmental variables or a secret manager.  `app.config['SECRET_KEY'] =  os.getenv('secret-name')`
References	- **ISO 27001**: Control A.9.4.3 (Password management system) - **SOC 2 Type 2**: Criterion CC6.4 (Security controls)

Risk Level	High
Vulnerability Class	Log files exposure and insecure logging (/logs Route)
CVSS v3.1 Rating	8.1 (AV:L, AC:L, PR:H, UI:N, S:U, C:H,I:L, AH)
Location	http://localhost:5000/logs
Description	
- No authentication and authorization when access log files - Log files should be sanitized and not contain sensitive or confidential information - Logs file may contain valuable information that attackers can use to exploit your system.	
Evidence	
Step to reproduce: Access http://localhost:5000/logs without any authentication.	



```
2025-02-21 21:23:52,838 INFO □[31m□[1mWARNING: This is a developr
2025-02-21 21:23:52,839 INFO □[33mPress CTRL+C to quit□[0m 2025-02
[21/Feb/2025 21:24:03] "□[33mGET / HTTP/1.1□[0m" 404 - 2025-02-21 2
2025-02-21 21:24:32,645 INFO 127.0.0.1 - - [21/Feb/2025 21:24:32] "□[33
21:25:37] "□[33mGET / HTTP/1.1□[0m" 404 - 2025-02-21 21:25:42,271 IN
02-21 21:26:08,293 INFO 127.0.0.1 - - [21/Feb/2025 21:26:08] "GET /users
- 2025-02-21 21:26:08,961 INFO 127.0.0.1 - - [21/Feb/2025 21:26:08] "GET
__debugger__=yes&cmd=resource&f=console.png HTTP/1.1" 200 - 2025-0
```

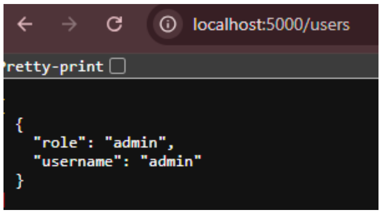
Recommendations	• Sanitize log message not directly passing the app message to the log file. • Logs should only be accessible to the right people. This may expose app logic and information that can be used to exploit your system
References	

Risk Level	Medium
Vulnerability Class	Broken Access Control (/users Route)
CVSS v3.1 Rating	5.0 (AV:L, AC:H, PR:H, UI:N, S:C, C:L,I:L, AL)
Location	http://localhost:5000/users

Description

- Users table or information should not be available to public. User information can be used in various types of attacks. (e.g. Phishing)
- Roles are sensitive because they define the level of access a user can do and can be identified and targeted.

Evidence



```
{
  "role": "admin",
  "username": "admin"
}
```

Recommendations	• Implement authentication and authorization methods in the /users' route. • User roles should only be visible to the admin or user himself.
References	https://owasp.org/Top10/A01_2021-Broken_Access_Control/

Risk Level	Medium
Vulnerability Class	Insecure Template Rendering
CVSS v3.1 Rating	5.0 (AV:L, AC:H, PR:H, UI:N, S:C, C:L,I:L, AL)
Location	http://localhost:5000/template

Description

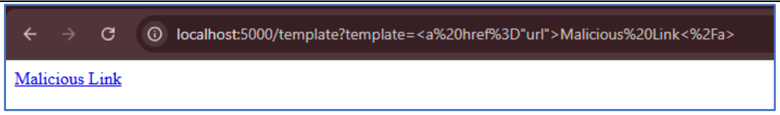
- Attacker can push malicious HTML payload and render it to the users

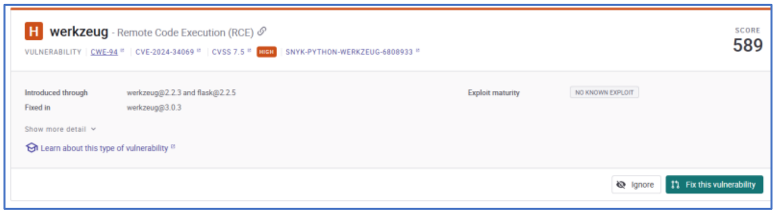
Evidence

Steps to reproduce:

Run the app.

Access Location URL and pass a parameters template with URL encoded value of a link

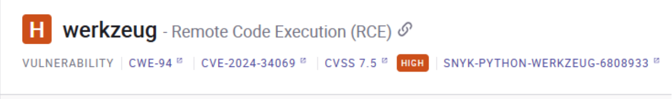
 localhost:5000/template?template=<a%20href%3D"url">Malicious%20Link<%2Fa> Malicious Link	
Recommendations	• If it is really necessary to allow users to upload template, add input encoding and validation. • Add firewall rules to detect XSS injection.
References	**ISO 27001**: Control A.14.2.5 (Secure system engineering principles) https://owasp.org/www-community/attacks/xss/

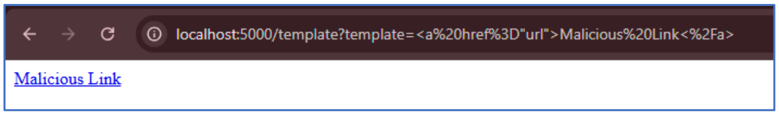
Risk Level	Low
Vulnerability Class	Project Dependency File must contain version limitations
CVSS v3.1 Rating	3.9 (AV:L, AC:H, PR:H, UI:N, S:C, C:N,I:L, AL)
Location	https://github.com/TableCheck-Labs/tablecheck-security-take-home/blob/main/app/requirements.txt
Description	
Although this not a high risk it's best practice to define versions of the dependencies. Declaring non versioned dependencies may lead to engineers installing a vulnerable version of it.	
Evidence	
SCA findings in Snyk:  H werkzeug - Remote Code Execution (RCE) VULNERABILITY | CWE-94 | CVE-2024-34069 | CVSS 7.5 HIGH | SNYK-PYTHON-WERKZEUG-6808933 SCORE 589 Introduced through werkzeug@2.2.3 and flask@2.2.5 Fixed in werkzeug@3.0.3 Exploit maturity NO KNOWN EXPLOIT Show more detail Learn about this type of vulnerability Ignore Fix this vulnerability	
Recommendations	Declare minimum versions to install `Werkzeug==3.0.6 # or Werkzeug>=2.3.0`
References	https://security.snyk.io/vuln/SNYK-PYTHON-WERKZEUG-8309092 https://cwe.mitre.org/data/definitions/94.html https://www.cve.org/CVERecord?id=CVE-2024-34069

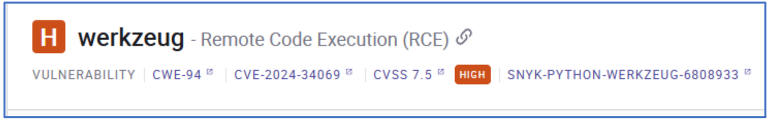
Snyk Findings (SCA)

Software Composition Analysis by **Snyk** (Free version)

Risk Level	High
Vulnerability Class	werkzeug Remote Code Execution (RCE)
CVSS v3.1 Rating	7.5
Location	https://github.com/TableCheck-Labs/tablecheck-security-take-home/blob/main/app/requirements.txt
Description	
Affected versions of this package are vulnerable to Remote Code Execution (RCE) due to insufficient hostname checks and the use of relative paths to resolve requests	
Evidence	

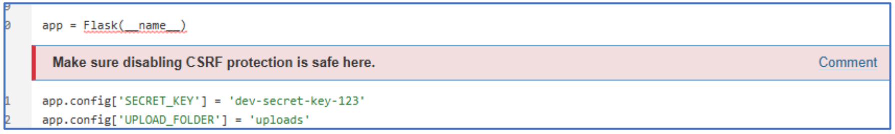
 werkzeug - Remote Code Execution (RCE) VULNERABILITY CWE-94 CVE-2024-34069 CVSS 7.5 HIGH SNYK-PYTHON-WERKZEUG-6808933	
Recommendations	Update the version to 3.0.6
References	https://cwe.mitre.org/data/definitions/94.html https://www.cve.org/CVERecord?id=CVE-2024-34069

Risk Level	Medium
Vulnerability Class	Insecure Template Rendering
CVSS v3.1 Rating	5.0 (AV:L, AC:H, PR:H, UI:N, S:C, C:L,I:L, AL)
Location	http://localhost:5000/template
Description	
• Attacker can push malicious HTML payload and render it to the users	
Evidence	
Steps to reproduce: Run the app. Access Location URL and pass a parameters template with URL encoded value of a link  localhost:5000/template?template=<a%20href%3D"url">Malicious%20Link<%2Fa> Malicious Link	
Recommendations	• If it is really necessary to allow users to upload template, add input encoding and validation. • Also restrict users from rendering links if possible • Add firewall rules to detect XSS injection.
References	**ISO 27001**: Control A.14.2.5 (Secure system engineering principles)

Risk Level	Low
Vulnerability Class	zipp Infinite loop
CVSS v3.1 Rating	6.9
Location	Requirements.txt
Description	
Infinite loop where an attacker can cause the application to stop responding by initiating a loop through functions affecting the Path module, such as joinpath, the overloaded division operator, and iterdir.	
Evidence	
 werkzeug - Remote Code Execution (RCE) VULNERABILITY CWE-94 CVE-2024-34069 CVSS 7.5 HIGH SNYK-PYTHON-WERKZEUG-6808933 https://security.snyk.io/vuln/SNYK-PYTHON-WERKZEUG-6808933	
Recommendations	Update the version to 3.0.6
References	https://cwe.mitre.org/data/definitions/94.html https://www.cve.org/CVERecord?id=CVE-2024-34069

Sonarqube Findings (SAST)

Risk Level	High
Vulnerability Class	Cross-Site Request Forgery (CSRF)

<table>
<tr><td>CVSS v3.1 Rating</td><td></td></tr>
<tr><td>Location</td><td><b>app.py</b></td></tr>
<tr><td colspan="2">Description</td></tr>
<tr><td colspan="2">No CSRF configuration enabled for Flask App</td></tr>
<tr><td colspan="2">Evidence</td></tr>
<tr><td colspan="2">
app = Flask(__name__)
Make sure disabling CSRF protection is safe here. Comment
app.config['SECRET_KEY'] = 'dev-secret-key-123'
app.config['UPLOAD_FOLDER'] = 'uploads'</td></tr>
<tr><td>Recommendations</td><td>• Set the CSRF configuration <b>WTF_CSRF_ENABLED = True</b> and add logic to you code to use CSRF tokens.
• It is recommended to not disable the CSRF protection on specific views or forms:</td></tr>
<tr><td>References</td><td>https://owasp.org/www-community/attacks/xss/
https://cwe.mitre.org/data/definitions/79.html</td></tr>
</table>

Appendix I – Security Automation and Recommendations

Pre-commit Hooks

These hooks are great way to enforce security controls at the early development phase. Here are the recommended pre-commit hooks:

- Secret Scanning – Prevents engineer from committing hardcoded secret to the repo.
- Static Analysis (e.g pylint) – Catches common security flaw in the code.

Recommendations:

Although there are many pre-commit hooks available, I recommend using the ones above as the minimum controls. This is because SAST and SCA tools can be integrated into the repo pipelines, allowing security personnel to track the status of these vulnerabilities. Below are the use cases and DevSecOps tools I recommend.

- **Static Application Security Testing (SAST)**:
  - **SonarQube** is an open-source tool that can be self-hosted. It is highly effective at detecting code flaws, misconfigurations, and even identifying committed secrets. It can be integrated into the development pipeline to prevent branch merging when issues are detected.
- **Software Composition Analysis (SCA):**
  - **Snyk (SCA)** – A free tool provided by Snyk, it scans project dependencies to detect vulnerabilities. It also includes code analysis to identify committed secrets.
  - **Dependency-Track**- This is an open-source tool that analyzes Software Bill of Materials, detect unsecure project dependencies and even detect dependency licensing issues.
- **DAST**:
  - **Owasp (ZAP)** – Scans the project in the runtime stage. And detects security issues and configuration.
- **All in one DevSecops Tool**:
  - **Akido**: An all-in-one DevSecOps tool that can be integrated with other commonly used DevSecOps tools in the market, such as SonarQube. Although it requires a license, it is highly powerful, featuring a denoise function that excludes false positive findings, allowing you to focus on critical vulnerabilities.