



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

Document Control	1
Executive Summary.....	2
Risk Assignment	2
Vulnerability Assessment	3
Key Findings:	3
Snyk Findings (SCA).....	6
Sonarqube Findings (SAST).....	7
Appendix I – Security Automation and Recommendations	8
Pre-commit Hooks	8
Recommendations:.....	8

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	2
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	Critical
Log files exposure and insecure logging (/logs Route)	High
Broken Access Control (/users Route)	Medium
Insecure Template Rendering	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):

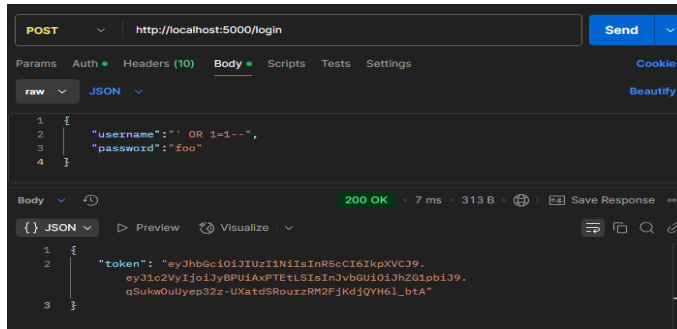
Critical	High	Medium	Low
0	1	0	0

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"}	
	
Recommendations	<ul style="list-style-type: none"> Use parameterized query to prevent SQL injection: conn = get_db() query = "SELECT * FROM users WHERE username=? AND password=?" user = conn.execute(query, (username, password)).fetchone() 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)

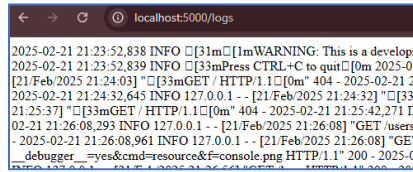
Risk Level	Critical
Vulnerability Class	Unrestricted File Uploads
CVSS v3.1 Rating	9.2 (AV:L, AC:L, PR:N, UI:N, S:U, C:H,I:L, AH)
Location	http://localhost:5000/upload
Description	
<ul style="list-style-type: none"> No file type validation. Attacker can upload malicious files since the endpoint is not 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.	

<pre>(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</pre>	
Recommendations	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> https://github.com/TableCheck-Labs/tablecheck-security-take-home/blob/main/app/app.py http://localhost:5000/export
Description	
<ul style="list-style-type: none"> 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	
<pre>app.config['SECRET_KEY'] = 'dev-secret-123'</pre> <pre>{ "app_config": { "debug": true, "secret_key": "dev-secret-key-123" }, "environment": {</pre>	
Recommendations	<ul style="list-style-type: none"> Use environmental variables or a secret manager. <pre>app.config['SECRET_KEY'] = os.getenv('secret-name')</pre>
References	<ul style="list-style-type: none"> 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	
<ul style="list-style-type: none"> 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.



```

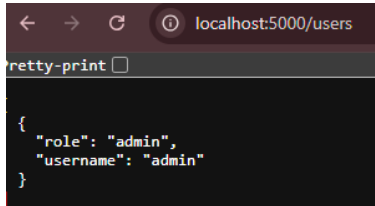
2025-02-21 21:23:52,838 INFO [31m[1mWARNING: This is a develop
2025-02-21 21:23:52,839 INFO [33mPress CTRL+C to quit[0m 2025-02-
[21/Feb/2025 21:24:03] "[35mGET / HTTP/1.1[0m" 404 - 2025-02-21 2
2025-02-21 21:24:32,645 INFO 127.0.0.1 - - [21/Feb/2025 21:24:32] "[33m
21:25:37] "[33mGET / HTTP/1.1[0m" 404 - 2025-02-21 21:25:42,271 I
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	
<ul style="list-style-type: none"> • 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	
 <pre> pretty-print [] { "role": "admin", "username": "admin" } </pre>	
Recommendations	<ul style="list-style-type: none"> • 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	
<ul style="list-style-type: none"> • Cross Site Scripting (XSS) Vulnerability • Attacker can push malicious Script/HTML payload and render it to the users 	
Evidence	
Steps to reproduce: Run the app.	

<div>  werkzeug - Remote Code Execution (RCE) 🔗 </div> <div> VULNERABILITY CWE-94 [🔗] CVE-2024-34069 [🔗] CVSS 7.5 [🔗] HIGH SNYK-PYTHON-WERKZEUG-6808933 [🔗] </div>	
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	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	
<div>  werkzeug - Remote Code Execution (RCE) 🔗 </div> <div> VULNERABILITY CWE-94 [🔗] CVE-2024-34069 [🔗] CVSS 7.5 [🔗] HIGH SNYK-PYTHON-WERKZEUG-6808933 [🔗] </div> 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)
CVSS v3.1 Rating	
Location	app.py
Description	
No CSRF configuration enabled for Flask App	
Evidence	
<pre> 5 app = Flask(__name__) 6 7 # Make sure disabling CSRF protection is safe here. 8 9 app.config['SECRET_KEY'] = 'dev-secret-key-123' 10 app.config['UPLOAD_FOLDER'] = 'uploads' </pre> <div> Make sure disabling CSRF protection is safe here. Comment </div>	
Recommendations	<ul style="list-style-type: none"> Set the CSRF configuration WTF_CSRF_ENABLED = True and add logic to you code to use CSRF tokens. It is recommended to not disable the CSRF protection on specific views or forms:
References	https://owasp.org/www-community/attacks/xss/ https://cwe.mitre.org/data/definitions/79.html

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.