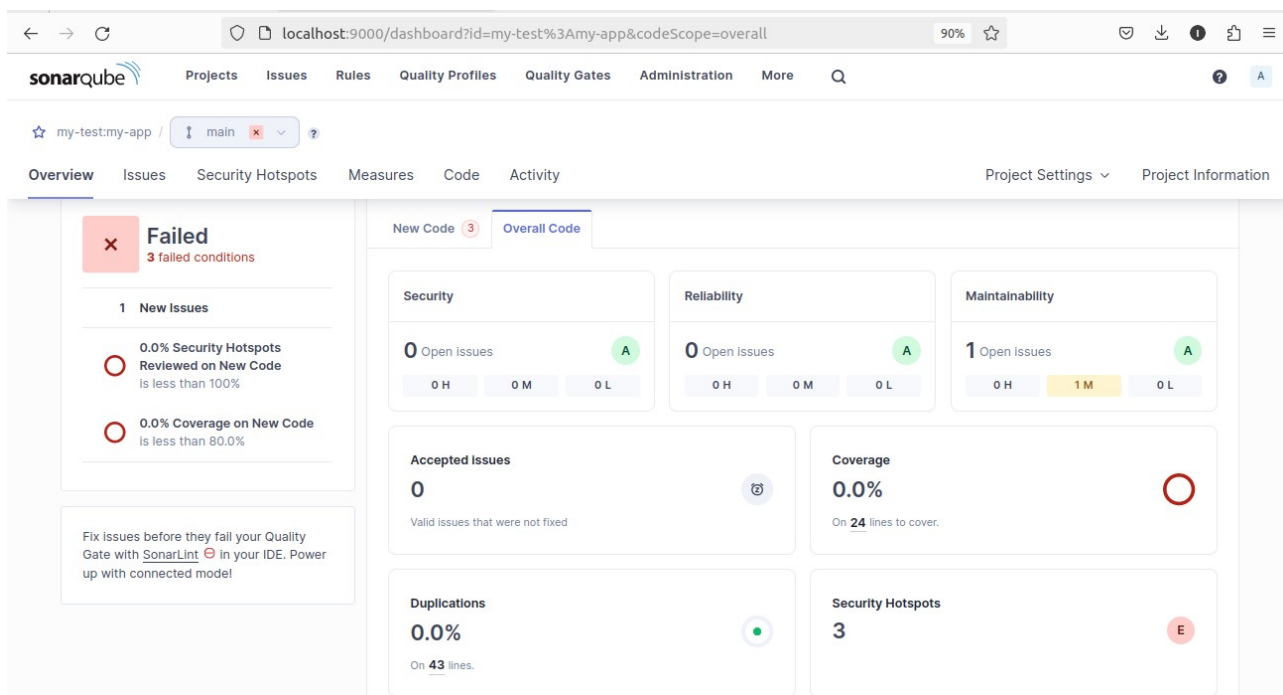


# Средства статического и динамического анализа ПО и инфраструктуры

В результате сканирования исходного кода было найдено следующие 3 уязвимости:

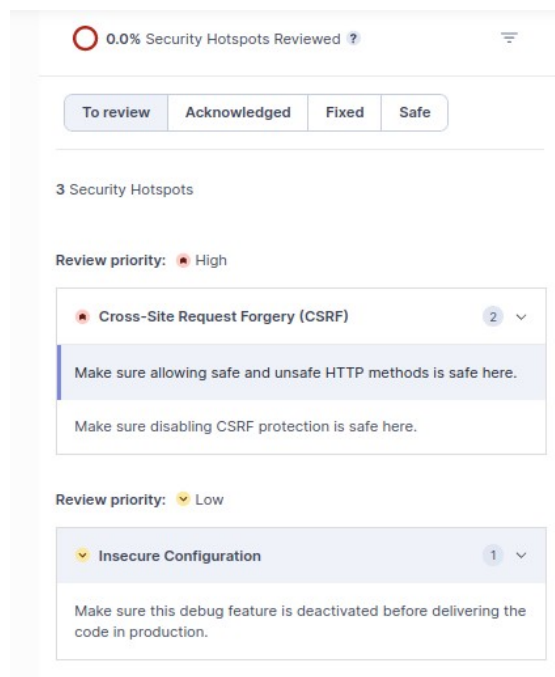


Cross-Site Request Forgery (CSRF):

- ◆ Make sure allowing safe and unsafe HTTP methods is safe here
- ◆ Make sure disabling CSRF protection is safe here

Insecure Configuration:

- ◆ Make sure this debug feature is deactivated before delivering the code in production



Далее подробно разберём каждую из них.

## Cross-Site Request Forgery (CSRF). Make sure allowing safe and unsafe HTTP methods is safe here.

An HTTP method is safe when used to perform a read-only operation, such as retrieving information. In contrast, an unsafe HTTP method is used to change the state of an application, for instance to update a user's profile on a web application.

Common safe HTTP methods are GET, HEAD, or OPTIONS.

Common unsafe HTTP methods are POST, PUT and DELETE.

Allowing both safe and unsafe HTTP methods to perform a specific operation on a web application could impact its security, for example CSRF protections are most of the time only protecting operations performed by unsafe HTTP methods.

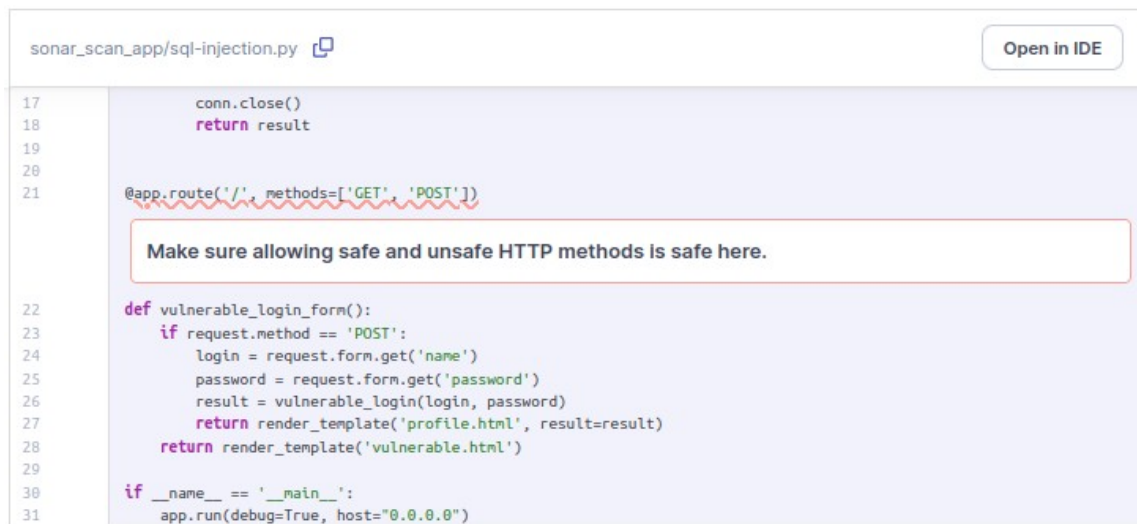
«Метод HTTP безопасен, если используется для выполнения операции только для чтения, например получения информации. Напротив, небезопасный метод HTTP используется для изменения состояния приложения, например, для обновления профиля пользователя в веб-приложении.


Распространенными безопасными методами HTTP являются GET, HEAD или OPTIONS.

Распространенными небезопасными методами HTTP являются POST, PUT и DELETE.

Разрешение как безопасным, так и небезопасным методам HTTP выполнять определенную операцию в веб-приложении может повлиять на его безопасность, например, защита CSRF в большинстве случаев защищает только операции, выполняемые небезопасными методами HTTP.»

Данная уязвимость была обнаружена в следующем блоке кода:



```
sonar_scan_app/sql-injection.py  Open in IDE  
17     conn.close()  
18     return result  
19  
20  
21 @app.route('/', methods=['GET', 'POST'])  
22  
23     def vulnerable_login_form():  
24         if request.method == 'POST':  
25             login = request.form.get('name')  
26             password = request.form.get('password')  
27             result = vulnerable_login(login, password)  
28             return render_template('profile.html', result=result)  
29             return render_template('vulnerable.html')  
30  
31 if __name__ == '__main__':  
    app.run(debug=True, host="0.0.0.0")
```

SonarQube предоставляет следующие рекомендации для устранения уязвимости:

For [Flask](#):

```
@methods.route('/compliant1')  
def view():  
    return Response("...", 200)
```

```
@methods.route('/compliant2', methods=['GET'])  
def view():  
    return Response("...", 200)
```

## Cross-Site Request Forgery (CSRF). Make sure disabling CSRF protection is safe here

A cross-site request forgery (CSRF) attack occurs when a trusted user of a web application can be forced, by an attacker, to perform sensitive actions that he didn't intend, such as updating his profile or sending a message, more generally anything that can change the state of the application.

The attacker can trick the user/victim to click on a link, corresponding to the privileged action, or to visit a malicious web site that embeds a hidden web request and as web browsers automatically include cookies, the actions can be authenticated and sensitive.

«Атака подделки межсайтовых запросов (CSRF) происходит, когда злоумышленник может заставить доверенного пользователя веб-приложения выполнить конфиденциальные действия, которые он не намеревался, например обновление своего профиля или отправку сообщения, в более общем смысле - все, что может изменить состояние приложения.

Злоумышленник может обманом заставить пользователя/жертву щелкнуть ссылку, соответствующую привилегированному действию, или посетить вредоносный веб-сайт, на котором встроен скрытый веб-запрос, а поскольку веб-браузеры автоматически включают файлы cookie, действия могут быть аутентифицированы и конфиденциальны.»

Данная уязвимость была обнаружена в следующем блоке кода:

sonar\_scan\_app/sql-injection.py 

Open in IDE

```
1 import sqlite3
2 from flask import Flask, render_template, request
3
4
5 app = Flask(__name__)
6
7 def vulnerable_login(username, password):
8     conn = sqlite3.connect('database.db')
9     cursor = conn.cursor()
10    query = f"SELECT * FROM users WHERE username='{username}' AND password='{password}'"
11    cursor.execute(query)
12    success_login = cursor.fetchone()
13    if success_login:
14        query_token = f"SELECT flag FROM token"
15        cursor.execute(query_token)
```

SonarQube предоставляет следующие рекомендации для устранения уязвимости:

For a **Flask** application,

- the `CSRFProtect` module should be used (and not disabled further with `WTF_CSRF_ENABLED` set to `false`):

```
app = Flask(__name__)
csrf = CSRFProtect()
csrf.init_app(app) # Compliant
```

- and it is recommended to not disable the CSRF protection on specific views or forms:

```
@app.route('/example/', methods=['POST']) # Compliant
def example():
    return 'example '

class unprotectedForm(FlaskForm):
    class Meta:
        csrf = True # Compliant

    name = TextField('name')
    submit = SubmitField('submit')
```

## Insecure Configuration. Make sure this debug feature is deactivated before delivering the code in production.

Development tools and frameworks usually have options to make debugging easier for developers. Although these features are useful during development, they should never be enabled for applications deployed in production. Debug instructions or error messages can leak detailed information about the system, like the application's path or file names.

«Инструменты и платформы разработки обычно имеют опции, упрощающие отладку для разработчиков. Хотя эти функции полезны во время разработки, их никогда не следует включать для приложений, развернутых в рабочей среде. Инструкции по отладке или сообщения об ошибках могут привести к утечке подробной информации о системе, такой как путь к приложению или имена файлов.»

Данная уязвимость была обнаружена в следующем блоке кода:

sonar\_scan\_app/sql-injection.py 

Open in IDE

```
21 @app.route('/', methods=['GET', 'POST'])
22 def vulnerable_login_form():
23     if request.method == 'POST':
24         login = request.form.get('name')
25         password = request.form.get('password')
26         result = vulnerable_login(login, password)
27         return render_template('profile.html', result=result)
28     return render_template('vulnerable.html')
29
30 if __name__ == '__main__':
31     app.run(debug=True, host="0.0.0.0")
```

Make sure this debug feature is deactivated before delivering the code in production.

SonarQube предоставляет следующие рекомендации для устранения уязвимости:

```
from flask import Flask

app = Flask()
app.debug = False
app.run(debug=False)
```

Используя рекомендации SonarQube, были исправлены блоки исходного кода. После дополнительного сканирования видим, что уязвимости в коде были устранены.

←→↻

localhost:9000/dashboard?id=my-test%3Amy-app

67%

☆

🔒

⬇

🔔

📄

☰

sonarqube

ProjectsIssuesRulesQuality ProfilesQuality GatesAdministrationMore

Q

my-test-my-app

main

OverviewIssuesSecurity HotspotsMeasuresCodeActivity

Project SettingsProject Information

main

53 Lines of Code • Version not provided • Set as homepage

The last analysis has warnings. See details

Quality Gate Status

Failed

2 failed conditions

1 New Issues

0.0% Coverage on New Code

is less than 80.0%

Fix issues before they fail your Quality Gate with SonarLint in your IDE. Power up with connected mode!

New CodeOverall Code

New Code: Since May 6, 2024 Started 9 hours ago

New issues FAILED

1

Required = 0

Accepted issues

0

Valid issues that were not fixed

Coverage FAILED

0.0%

Required ≥ 80.0%

On 30 New Lines to cover.

Duplications

0.0%

Required ≤ 3.0%

On 53 New Lines.

Security Hotspots

0

A