



FGA 0238 - Testes de Software – Turma: 02

Semestre: 2024.1

Equipe: 5 - Os Abandonados

Nomes: Cássio Sousa dos Reis

Matrículas: 221021886

João Lucas Pinto Vasconcelo

190089601

Kauan de Torres Eiras

232014727

Rodrigo Braz Ferreira Gontijo

190116498

PTOSS 5 – Teste de Segurança

1 Aplicação Analisada

1.1. Identificação da Aplicação

Nome: MEC-Energia

Link para o repositório: <https://gitlab.com/lappis-unb/projetos-energia/mec-energia>

1.2. Descrição

O projeto MEC-Energia é um sistema de recomendação de contratos de energia elétrica, desenvolvido para ajudar as instituições de ensino superior (IES) a gerenciar e avaliar a adequação dos seus contratos de energia. Ele funciona a partir do registro das faturas mensais, permitindo a geração de relatórios que sugerem ajustes nos contratos para otimizar o consumo e economizar recursos.

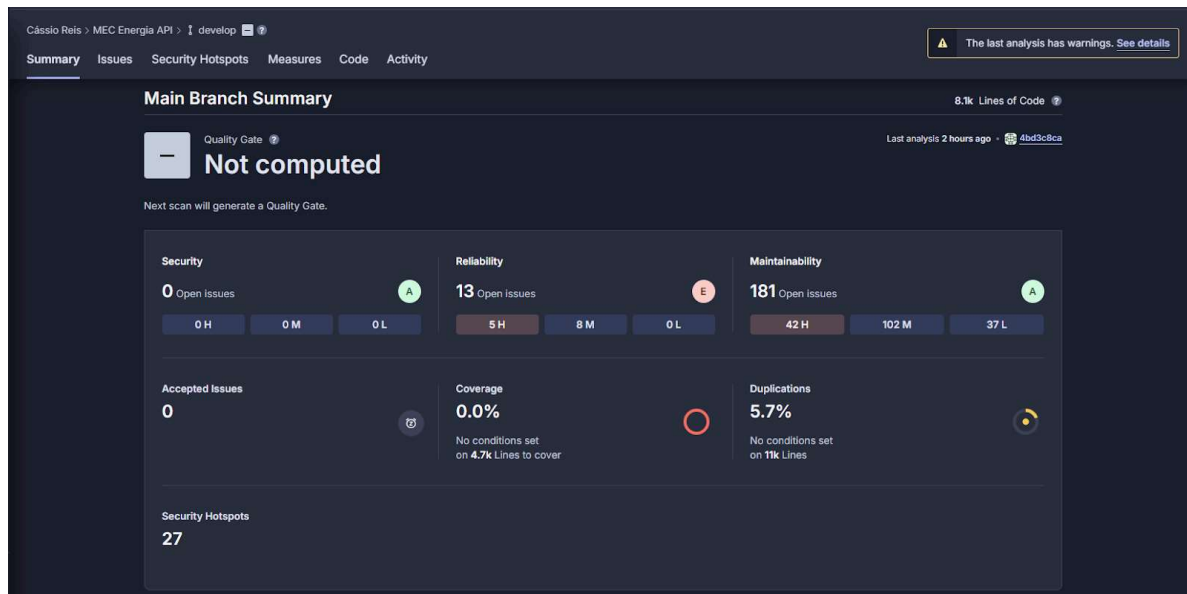
1.3. Linguagens

MEC Energia Web (*frontend*): Typescript

MEC Energia API (*backend*): Python

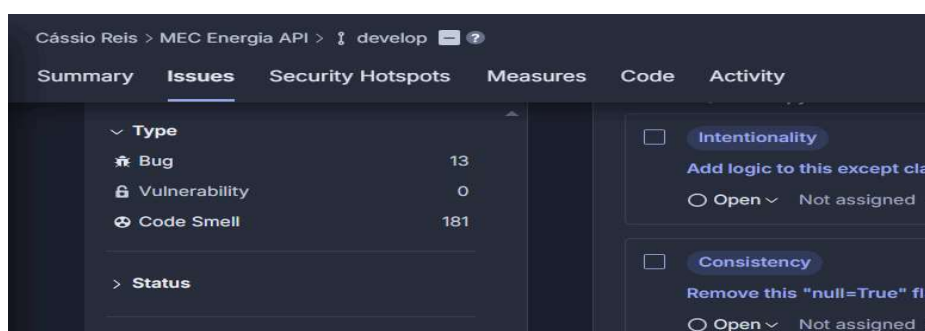
2 Visão Geral do Resultado

A análise foi realizada sobre o MEC Energia API (backend). A imagem abaixo mostra a tela do resultado da análise estática utilizando o SonarCloud:



Segundo a análise estática do SonarCloud, o projeto MEC Energia API tem 27 security hotspots, que podem potencialmente introduzir riscos de segurança, e devem ser revisados manualmente para garantir que não representem uma vulnerabilidade.

Além disso, também foram detectados 13 bugs e 181 code smells, que podem impactar a manutenibilidade e a qualidade geral do projeto, exigindo uma revisão cuidadosa e possível refatoração para melhorar o código. Nenhuma vulnerabilidade foi identificada pela ferramenta, o que é um bom sinal para a segurança do projeto.



O SonarCloud está apresentando uma cobertura de testes de 0,0%, embora a cobertura real do projeto seja de 59%.

3 Vulnerabilidades

Como mostrado nas capturas de tela, a ferramenta não detectou vulnerabilidades.



4 Hot Spots

A seguir, tem-se a lista dos security hotspots detectados na análise. A análise identificou os seguintes hotspots, em que os números entre parênteses indicam a quantidade de cada tipo de hotspot identificado:

Authentication (20):

- “password” detected here, review this potentially hard-coded credential. (20)

Cross-Site Request Forgery (CRSF) (1):

- Make sure allowing safe and unsafe HTTP methods is safe here

Permission (2):

- Copying recursively might inadvertently add sensitive data to the container. Make sure it is safe here.
- The python image runs with root as the default user. Make sure it is safe here.

Weak Cryptography (1):

- Make sure that using this pseudorandom number generator is safe here.

Encryption of Sensitive Data (1)

- Using http protocol is insecure. Use https instead

Insecure Configuration (1)

- Make sure this permissive CORS policy is safe here.

Others (1)

- Make sure automatically installing recommended packages is safe here.

5 Análise das Vulnerabilidades ou Hot Spots

5.1. Authentication

5.1.1. Descrição

“password” detected here, review this potentially hard-coded credential.

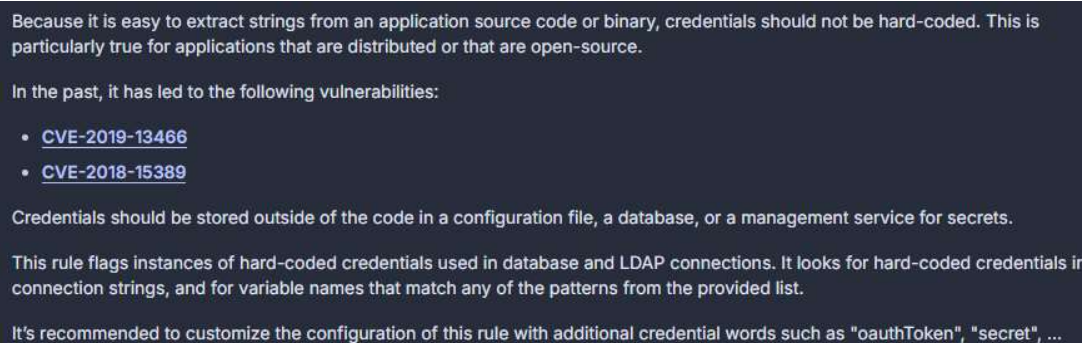
Este hotspot indica credenciais hard-coded, como senhas diretamente no código. Essa prática é insegura, e a exposição dessas credenciais é um risco significativo, pois strings hard-coded podem ser facilmente extraídas de código-fonte ou binários, levando a vulnerabilidades conhecidas, como comprometimento de segurança e acesso não autorizado.

5.1.2. Análise de falso positivo:

A instância detectada não se refere a uma credencial armazenada de forma insegura, mas sim a um uso legítimo que não compromete a segurança do sistema.

O security hotspot selecionado para análise se refere a scripts que criam dados iniciais (seed data) para um ambiente de desenvolvimento ou teste, e a senha é usada apenas nesses contextos e não em produção, logo pode ser considerado um falso positivo.

No entanto, boas práticas recomendam evitar hard-coding de senhas sempre que possível e usar métodos alternativos para carregar essas informações de maneira segura.



Because it is easy to extract strings from an application source code or binary, credentials should not be hard-coded. This is particularly true for applications that are distributed or that are open-source.

In the past, it has led to the following vulnerabilities:

- [CVE-2019-13466](#)
- [CVE-2018-15389](#)

Credentials should be stored outside of the code in a configuration file, a database, or a management service for secrets.

This rule flags instances of hard-coded credentials used in database and LDAP connections. It looks for hard-coded credentials in connection strings, and for variable names that match any of the patterns from the provided list.

It's recommended to customize the configuration of this rule with additional credential words such as "oAuthToken", "secret", ...

Descrição do hotspot no SonarCloud

5.1.3. Solução

Para resolver o hotspot detectado como um falso positivo, é essencial verificar o contexto para garantir que as credenciais detectadas são usadas apenas para fins de teste e não em ambientes de produção. Soluções propostas:

- Separar as credenciais de seed data do código-fonte principal
- usar dados fictícios ou senhas temporárias para testes
- documentar claramente o propósito dessas credenciais

O SonarCloud recomenda as seguintes práticas:

Recommended Secure Coding Practices

- Store the credentials in a configuration file that is not pushed to the code repository.
- Store the credentials in a database.
- Use your cloud provider's service for managing secrets.
- If a password has been disclosed through the source code: change it.

Recomendações SonarCloud

5.2 Weak Cryptography

5.2.1 Descrição

Make sure that using this pseudorandom number generator is safe here.

O uso inadequado de geradores de números pseudoaleatórios (PRNGs) é um problema significativo em contextos que exigem criptografia robusta. PRNGs produzem valores baseados em algoritmos determinísticos que não oferecem a aleatoriedade necessária para garantir a segurança criptográfica. Quando PRNGs são usados para gerar valores críticos, como senhas, tokens ou chaves, a previsibilidade desses valores pode ser explorada por atacantes, resultando em graves falhas de segurança. Este problema está classificado como "Cryptographic Failures", que ocupa o #2 lugar na lista OWASP Top 10 de 2021. Ela abrange falhas relacionadas ao uso inadequado de criptografia, incluindo a escolha de algoritmos e práticas inseguras para proteger dados sensíveis.

5.2.2 Análise de falso positivo

```
↑ Show 1 more lines
2  import random
3
4  from mec_energia import settings
5
6  def generate_random_password():
7      return ''.join(random.choice(string.ascii_lowercase + string.digits) for _ in range(20))
8
9  def create_token_response(token, user_id, user_email, user_first_name, user_last_name, user_type):
10     response = {
11         'token': token,
12         'user': {
```

Make sure that using this pseudorandom number generator is safe here.

Captura de tela do hotspot

Este hotspot de segurança se refere ao uso do módulo `random` do Python para gerar senhas. O módulo **random** é um gerador de números pseudoaleatórios (PRNG) projetado para tarefas gerais de geração de números aleatórios, mas não é seguro para aplicações criptográficas ou de segurança. O código atual utiliza `random.choice()` para selecionar caracteres de uma lista, criando senhas previsíveis e menos robustas contra ataques. Isso pode comprometer a segurança das senhas, tornando-as mais vulneráveis a ataques de força bruta e adivinhação, o que pode levar a comprometimentos de contas e dados sensíveis.

Dado que a função `generate_random_password()` está sendo usada em processos críticos, como a criação de contas de usuário e a redefinição de senhas, o uso de um PRNG não seguro neste contexto constitui uma vulnerabilidade real e não se trata de um falso positivo.

5.2.3. Solução

Para solucionar o problema de segurança relacionado ao uso de geradores de números pseudoaleatórios (PRNGs) inadequados para a geração de senhas, deve-se substituir o PRNG atual por um gerador de números aleatórios criptograficamente seguro. No Python, o módulo `secrets` é recomendado para esse tipo de operação, pois ele garante que os valores gerados sejam criptograficamente seguros e adequados para contextos que exigem alta imprevisibilidade, como na criação de senhas.

A função `generate_random_password()` deve ser modificada para utilizar o módulo `secrets` em vez de `random`. Isso garantirá que as senhas geradas sejam imprevisíveis e resistentes a ataques de força bruta ou de adivinhação.

5.3 Cross-Site Request Forgery (CRSF)

5.3.1 Descrição

A ferramenta apontou uma possível vulnerabilidade de CSRF na função `upload_csv(self, request, *args, **kwargs)`. Isso se deve ao fato de que a função lida com uma operação sensível (POST) que, se não estiver devidamente protegida, pode ser explorada por um atacante.

Make sure allowing safe and unsafe HTTP methods is safe here. [🔗](#)

Allowing both safe and unsafe HTTP methods is security-sensitive [python:S3752](#)

Status: To Review

This Security Hotspot needs to be reviewed to assess whether the code poses a risk.

Review priority:
🔴 High

Category:
Cross-Site Request Forgery (CSRF)

Assignee:
Not assigned

Where is the risk?	What's the risk?	Assess the risk	How can I fix it?	Activity
--------------------	------------------	-----------------	-------------------	----------

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.

Descrição do hotspot no SonarCloud

5.3.2 Análise de falso positivo

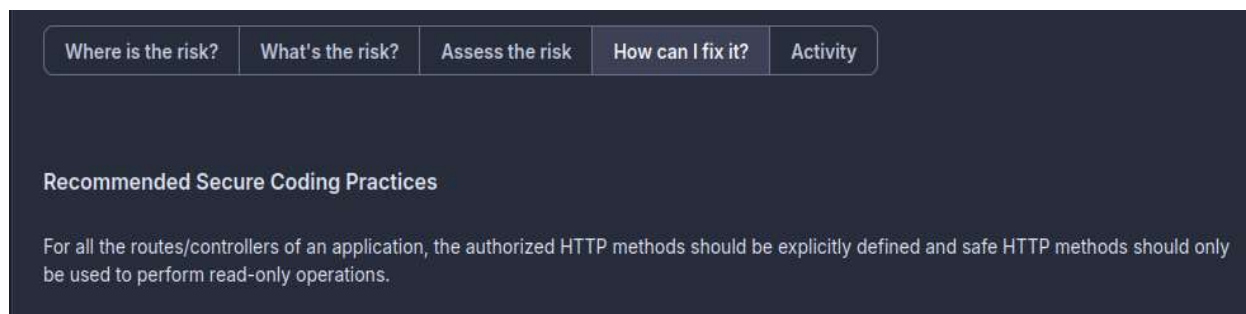
O ponto crítico aqui é se a função “upload_csv” está protegida contra CSRF, que geralmente afeta requisições do tipo POST. No código do Django, framework utilizado no projeto, essa proteção é geralmente habilitada por padrão para todas as requisições POST, a menos que explicitamente desabilitada.

Dado que o método “upload_csv” utiliza o método POST e não há indícios de que o token CSRF foi desativado, a proteção contra CSRF deve estar ativa, o que indica que a ameaça identificada provavelmente é um falso positivo. O Django já deve estar protegendo essa função contra ataques CSRF.

5.3.3. Solução

Seguindo a recomendação do SonarCloud, garanta que todos os controladores e rotas do projeto tenham os métodos HTTP autorizados explicitamente definidos. Além disso, use métodos seguros (como GET) exclusivamente para operações de leitura, enquanto métodos como POST, PUT, e DELETE devem ser utilizados para operações que modificam dados.

Outra solução é verificar se o middleware “CsrfViewMiddleware” do Django está ativo no projeto para garantir que a proteção CSRF esteja aplicada a todas as requisições POST.



Solução recomendada pelo SonarCloud

5.4 Copying recursively might inadvertently add sensitive data to the container

5.4.1 Descrição

A mensagem de alerta "Copying recursively might inadvertently add sensitive data to the container" indica que a instrução 'COPY . .' no Dockerfile pode estar copiando arquivos e diretórios que não deveriam ser incluídos na imagem do contêiner. Isso pode resultar na inclusão acidental de dados sensíveis, como credenciais, arquivos de configuração ou outros dados que não são necessários para a execução da aplicação em produção.

5.4.2 Problema Identificado

A instrução COPY . . copia todos os arquivos e diretórios do contexto de construção para o contêiner, o que pode incluir arquivos indesejados. Isso é especialmente perigoso em ambientes de produção, onde a exposição acidental de dados sensíveis pode levar a vulnerabilidades de segurança.

5.4.3 Proposta

A correção proposta envolve a substituição da instrução COPY . . por instruções de cópia mais específicas, que apenas incluem os arquivos e diretórios necessários para a aplicação. Isso minimiza o risco de inclusão de dados sensíveis e melhora a segurança geral da imagem do contêiner.

5.4.4 Código Corrigido

Aqui está o Dockerfile corrigido, conforme sua proposta:


```
Dockerfile
1 FROM python:3.10.5
2
3 # -----< COPY -----
4
5 COPY contracts/ contracts/
6 COPY cronjob/ cronjob/
7 COPY global_search_recommendation/ global_search_recommendation/
8 COPY mec_energia/ mec_energia/
9 COPY recommendation/ recommendation/
10 COPY recommendation_commons/ recommendation_commons/
11 COPY scripts/ scripts/
12 COPY tariffs/ tariffs/
13 COPY tests/ tests/
14 COPY universities/ universities/
15 COPY users/ users/
16 COPY utils/ utils/
17
18 COPY .coveragerc .coveragerc
19 COPY .editorconfig .editorconfig
20 COPY .pylintrc .pylintrc
21 COPY manage.py manage.py
22 COPY pytest.ini pytest.ini
23 COPY requirements.txt requirements.txt
24
25 COPY cronjob/cronjob /etc/cron.d/mec-cron
26
27 RUN apt-get update && \
28 apt-get install -y libpq-dev cron
29
30 RUN pip install -U --no-cache-dir -r requirements.txt
31
32 # -----< cron -----
33 RUN chmod -R 755 /etc/cron.d/mec-cron && \
34 /usr/bin/crontab /etc/cron.d/mec-cron
35
36 WORKDIR /home/dev/mec-energia-api
```

5.4.5 Justificativa da Correção

Especificidade: Ao especificar quais diretórios e arquivos devem ser copiados, você reduz a chance de incluir arquivos indesejados ou sensíveis.

Segurança: Esta abordagem melhora a segurança da imagem do contêiner, garantindo que apenas os arquivos necessários para a execução da aplicação sejam incluídos.

Manutenção: Um Dockerfile mais explícito e organizado facilita a manutenção e a compreensão do que está sendo incluído na imagem.

5.4.6 Conclusão

A correção proposta para o Dockerfile aborda diretamente o hotspot identificado, garantindo que a aplicação seja construída de forma mais segura. É importante sempre revisar o que está sendo copiado para o contêiner e evitar práticas que possam expor dados sensíveis. Além disso, recomenda-se a utilização de um arquivo '.dockerignore' para excluir arquivos e diretórios que não devem ser incluídos na construção da imagem, como arquivos de configuração, chaves secretas e outros dados

Link para o commit da correção: [Correção de segurança no Dockerfile \(7c7efa1f\)](#) · Commits · João Lucas Vas / MEC Energia API2 · GitLab

5.5 Encryption of Sensitive Data

5.5.1 Descrição

O hotspot detectado refere-se ao uso do protocolo HTTP em vez de HTTPS para comunicação. O uso de HTTP é considerado inseguro, pois não fornece criptografia para os dados em trânsito, o que pode levar a várias vulnerabilidades, incluindo a exposição de dados sensíveis, redirecionamento de tráfego para endpoints maliciosos, execução de código no cliente, e corrupção de informações críticas.

The screenshot shows a SonarCloud security hotspot. At the top, the title is "Using http protocol is insecure. Use https instead" with a link icon. Below it, a subtitle reads "Using clear-text protocols is security-sensitive" followed by a link to "python:S5332". On the right, the "Review priority" is "Low" (yellow smiley icon), the "Category" is "Encryption of Sensitive Data", and the "Assignee" is "Not assigned". The main content area has a "Status: To Review" box with a "Review" button. Below this is a tabbed interface with tabs: "Where is the risk?", "What's the risk?", "Assess the risk", "How can I fix it?", and "Activity". The "What's the risk?" tab is active, showing a detailed explanation of the risk: "Clear-text protocols such as ftp, telnet, or http lack encryption of transported data, as well as the capability to build an authenticated connection. It means that an attacker able to sniff traffic from the network can read, modify, or corrupt the transported content. These protocols are not secure as they expose applications to an extensive range of risks:". This is followed by a bulleted list of risks: sensitive data exposure, traffic redirected to a malicious endpoint, malware-infected software update or installer, execution of client-side code, and corruption of critical information. The text continues: "Even in the context of isolated networks like offline environments or segmented cloud environments, the insider threat exists. Thus, attacks involving communications being sniffed or tampered with can still happen. For example, attackers could successfully compromise prior security layers by:". This is followed by another bulleted list: bypassing isolation mechanisms, compromising a component of the network, and getting the credentials of an internal IAM account (either from a service account or an actual person). The text concludes: "In such cases, encrypting communications would decrease the chances of attackers to successfully leak data or steal credentials from other network components. By layering various security practices (segmentation and encryption, for example), the application will follow the defense-in-depth principle. Note that using the http protocol is being deprecated by major web browsers. In the past, it has led to the following vulnerabilities:". This is followed by a bulleted list of CVEs: CVE-2019-6169, CVE-2019-12327, and CVE-2019-11065.

Descrição do hotspot no SonarCloud

5.5.2 Problema Identificado

Essa detecção é uma vulnerabilidade séria. O uso de HTTP para transmissão de dados permite que atacantes realizem ataques de interceptação de tráfego (man-in-the-middle), onde eles podem monitorar, capturar e manipular os dados transmitidos entre o cliente e o servidor. Ferramentas como Wireshark, tcpdump e Ettercap tornam esse tipo de ataque ainda mais fácil, pois permitem que um atacante capture pacotes de dados em redes desprotegidas e visualize informações sensíveis em texto claro, como credenciais de login, números de cartão de crédito, ou outros dados confidenciais. Além disso, sem criptografia, os dados podem ser redirecionados para servidores controlados por atacantes, onde malware pode ser injetado ou informações podem ser roubadas sem o conhecimento do usuário. Esse tipo de ataque é especialmente perigoso em redes públicas,

como Wi-Fi em cafés ou aeroportos, onde o tráfego de rede pode ser facilmente monitorado por qualquer pessoa conectada à mesma rede.

Sensitive Code Example

```
url = "http://example.com" # Sensitive
url = "ftp://anonymous@example.com" # Sensitive
url = "telnet://anonymous@example.com" # Sensitive

import telnetlib
cnx = telnetlib.Telnet("towel.blinkenlights.nl") # Sensitive

import ftplib
cnx = ftplib.FTP("ftp.example.com") # Sensitive

import smtplib
smtp = smtplib.SMTP("smtp.example.com", port=587) # Sensitive
```

Exemplo de Códigos não seguros

5.5.3 Solução

Para mitigar essa vulnerabilidade, a recomendação é substituir HTTP por HTTPS, alterando todas as URLs no código que usam HTTP para HTTPS. HTTPS usa o protocolo TLS (Transport Layer Security) para criptografar os dados transmitidos, dificultando que eles sejam interceptados ou manipulados por terceiros. Implementar práticas seguras, como o uso de TLS, faz com que as conexões sejam autenticadas e criptografadas de ponta a ponta.

Configurações Adicionais:

- Configurar a aplicação para bloquear conteúdo misto ao renderizar páginas da web. Isso previne que partes de uma página sejam carregadas usando HTTP enquanto o restante usa HTTPS.
- Criptografar a comunicação de componentes na nuvem sempre que possível.

5.6 Insecure Configuration: Permissive CORS Policy

5.6.1 Descrição

O hotspot detectado refere-se ao uso de uma política permissiva para o CORS (Cross-Origin Resource Sharing), na configuração `CORS_ORIGIN_ALLOW_ALL = True`. Isso permite que qualquer origem tenha acesso aos recursos da aplicação, o que pode expor a aplicação a riscos significativos de segurança. CORS é um mecanismo de segurança que restringe como recursos em uma página web podem ser solicitados de outro domínio fora do domínio ao qual o recurso pertence. Quando mal configurado, ele pode permitir que sites não confiáveis acessem dados sensíveis.

5.6.2 Riscos de Segurança

A configuração atual do CORS representa uma vulnerabilidade significativa. Quando a política CORS é excessivamente permissiva, qualquer site pode fazer solicitações à sua API, o que pode levar a uma série de ataques, incluindo:

- **Ataques de Cross-Site Scripting (XSS):** Um atacante pode usar um site malicioso para fazer solicitações à sua API em nome de um usuário legítimo, potencialmente acessando dados sensíveis.
- **Roubo de dados:** Um site malicioso pode capturar informações privadas ou confidenciais que seriam de outra forma protegidas.
- **Execução de comandos maliciosos:** Se o site malicioso tiver permissões inadequadas, ele pode executar comandos potencialmente prejudiciais na API.

Programas como **Burp Suite** e **OWASP ZAP** podem ser usados para explorar essas falhas, permitindo que um atacante teste quais sites conseguem interagir com sua API, expondo vulnerabilidades potencialmente devastadoras.

5.6.3 Solução

Para mitigar essa vulnerabilidade, é crucial implementar uma política CORS mais restritiva. Aqui estão as etapas recomendadas:

1. Restringir a Origem Permitida:

- Não se deve permitir todas as origens (`CORS_ORIGIN_ALLOW_ALL = True`). Ao invés disso, é importante limitar o acesso apenas a domínios confiáveis. No Django, isso pode ser feito com `CORS_ORIGIN_ALLOW_ALL = False`.

2. Validação Rigorosa do Header Origin:

- Evite usar asterisco (*), que representa todos ou tudo, ou retornar o conteúdo do header Origin sem validação. O ideal é que haja uma lista de permissões com endereços de origens confiáveis que podem acessar a API.

6 Conclusão sobre o teste de segurança

A análise de segurança do MEC Energia API foi realizada por meio da análise estática utilizando o SonarCloud. Esse tipo de teste examina o código-fonte sem executá-lo, identificando potenciais vulnerabilidades, bugs e code smells que poderiam comprometer a segurança e a qualidade da aplicação.

A segurança da aplicação MEC Energia API, conforme revelado pela análise estática e as análises dos security hotspots, mostra um cenário misto. Embora não tenham sido detectadas vulnerabilidades críticas, os hotspots apontam para áreas como autenticação, uso inadequado de PRNGs, e configurações permissivas de CORS, que necessitam de atenção. Embora algumas dessas detecções sejam falsos positivos, como o hotspot de CSRF e de senhas hardcoded, outras representam riscos reais que devem ser mitigados para evitar futuras explorações.

Em suma, o teste de análise estática provou ser eficaz na detecção de uma variedade de problemas que podem afetar tanto a segurança quanto a qualidade do código do projeto analisado. A adoção de medidas corretivas baseadas nos resultados dessa análise é essencial para fortalecer a segurança da aplicação e garantir sua operação segura em ambientes de produção.