



**FGA-0238 - Testes de Software**

<b>Turma</b>	01	<b>Semestre</b>	2024.1
--------------	----	-----------------	--------

<b>Equipe</b>	NeverMind
---------------	-----------

<b>Nome</b>	<b>Matrícula</b>
Elias Faria de Oliveira	221007706
João Eduardo Pereira Rabelo	180053299
Yan Lucas Souza Guimarães	222006220
Daniel dos Santos Barros de Sousa	211030980

# 1. Aplicação Analisada

## 1. Identificação da Aplicação

Aplicação: MEC Energia

Link: [MEC Energia API - lappis-unb](#)

## 2. Descrição

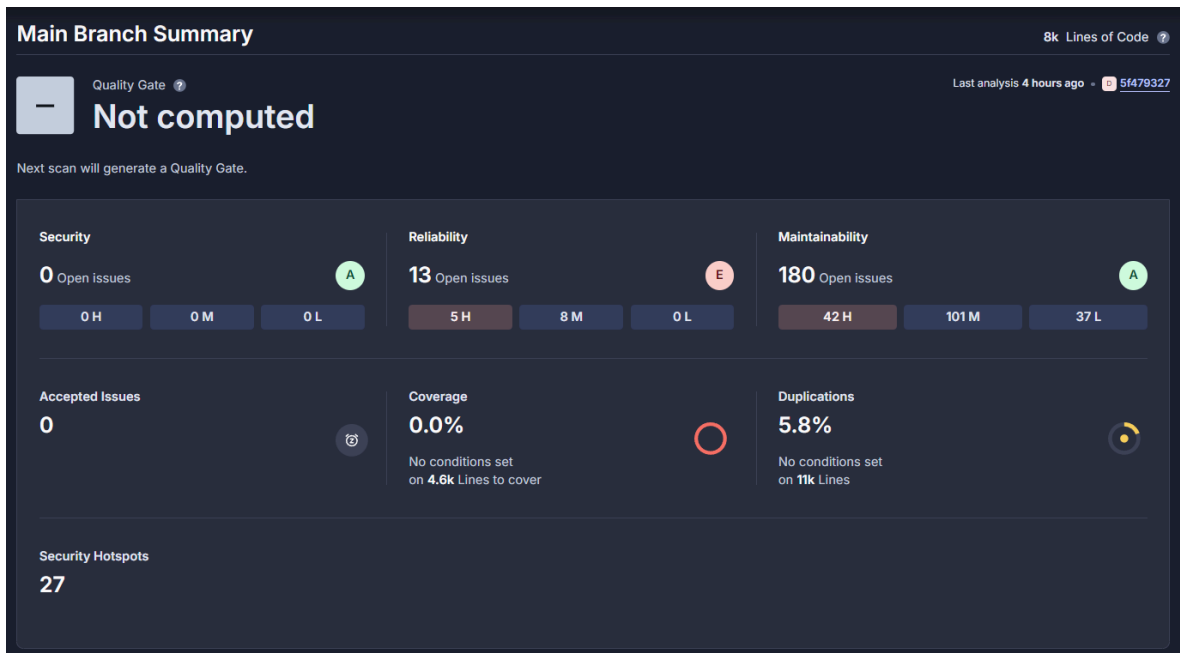
O projeto **MEC Energia API** é uma aplicação desenvolvida pelo Laboratório Avançado de Produção, Pesquisa e Inovação em Software (LAPPIS) da Universidade de Brasília. Esta API faz parte de uma iniciativa para gerenciar e otimizar o consumo de energia elétrica em instituições educacionais. A API oferece funcionalidades para monitoramento, análise e controle do uso de energia, permitindo que as instituições implementem práticas mais eficientes e sustentáveis. Desenvolvida principalmente em Python, a API facilita a integração com outros sistemas e aplicações voltadas à gestão energética.

## 3. Linguagens

O projeto **MEC Energia API** utiliza principalmente as seguintes linguagens:

- **Python:** A linguagem principal usada para o desenvolvimento da API.
- **YAML:** Utilizada para configuração, como arquivos de pipeline e definições de infraestrutura.
- **HTML:** Pode ser utilizada em templates ou documentação.

## 2. Visão Geral do Resultado



**Hotspots de Segurança:** A análise identificou hotspots de segurança, que são áreas do código que podem potencialmente conter vulnerabilidades, exigindo uma revisão mais detalhada. Um exemplo relevante é a função `generate_random_password()`, que utiliza um gerador de números pseudo aleatórios. Essa implementação levanta preocupações sobre a segurança da senha gerada, pois o uso de geradores de números pseudo aleatórios (PRNGs) pode não ser seguro para fins criptográficos, o que torna as senhas previsíveis e suscetíveis a ataques.

**Vulnerabilidades:** Embora os hotspots indiquem possíveis problemas, é necessário investigar se essas questões representam vulnerabilidades reais ou falsos positivos. No caso da função mencionada, o risco está na previsibilidade do PRNG, o que poderia comprometer a segurança dos usuários. Para garantir a segurança, seria recomendável substituir o PRNG por um gerador de números verdadeiramente aleatórios adequado para criptografia, como `secrets` em Python.

**Qualidade do Código:** A análise geral inclui também informações sobre bugs, vulnerabilidades gerais, dívida técnica e a densidade de linhas duplicadas, o que contribui para a compreensão da qualidade do código como um todo. A dívida técnica, por exemplo, indica o esforço necessário para corrigir esses problemas e melhorar a manutenção do software.

### 3. Vulnerabilidades

Acessando o Sonar Cloud, foi possível verificar as seguintes vulnerabilidades no projeto:

- Foram detectadas 16 vulnerabilidades associadas a credenciais presentes no código da aplicação. Essas vulnerabilidades podem resultar em possíveis usos indevidos da aplicação, especialmente se ocorrer uma extração ou invasão. Caso um atacante obtenha acesso, às credenciais poderão estar expostas, permitindo ações que, de outro modo, estariam restritas a usuários com permissões de administrador, o que comprometeria severamente a segurança da aplicação.

```
# Redefinição de senha
RESET_PASSWORD_TOKEN_TIMEOUT= inteiro(os.getenv(
'REINICIAR_TEMPO_LIMITE_DO_TOKEN_DE_SENHA'))
REENVIAR_EMAIL_REINICIAR_TEMPO_LIMITE_DE_SENHA= inteiro(os.getenv(
'REENVIAR_EMAIL_REINICIAR_TEMPO_LIMITE_DE_SENHA'))
MEC_ENERGIA_SENHA_ENDPOINT_PRIMEIRO_ACESSO='definir-senha'
MEC_ENERGIA_SENHA_ENDPOINT_ADMIN_RESET='redefinir-senha'
MEC_ENERGIA_SENHA_ENDPOINT_RESET_USUÁRIO='definir-senha'
```

- Na seção de criptografia, foi identificada a função "generate random password", a qual revela uma vulnerabilidade associada à criação de senhas durante o processo de cadastro do usuário. A prática atual exige que o usuário altere a senha gerada inicialmente, porém, isso acarreta um risco significativo: a possibilidade de que alguns usuários não realizem a alteração da senha após o cadastro.

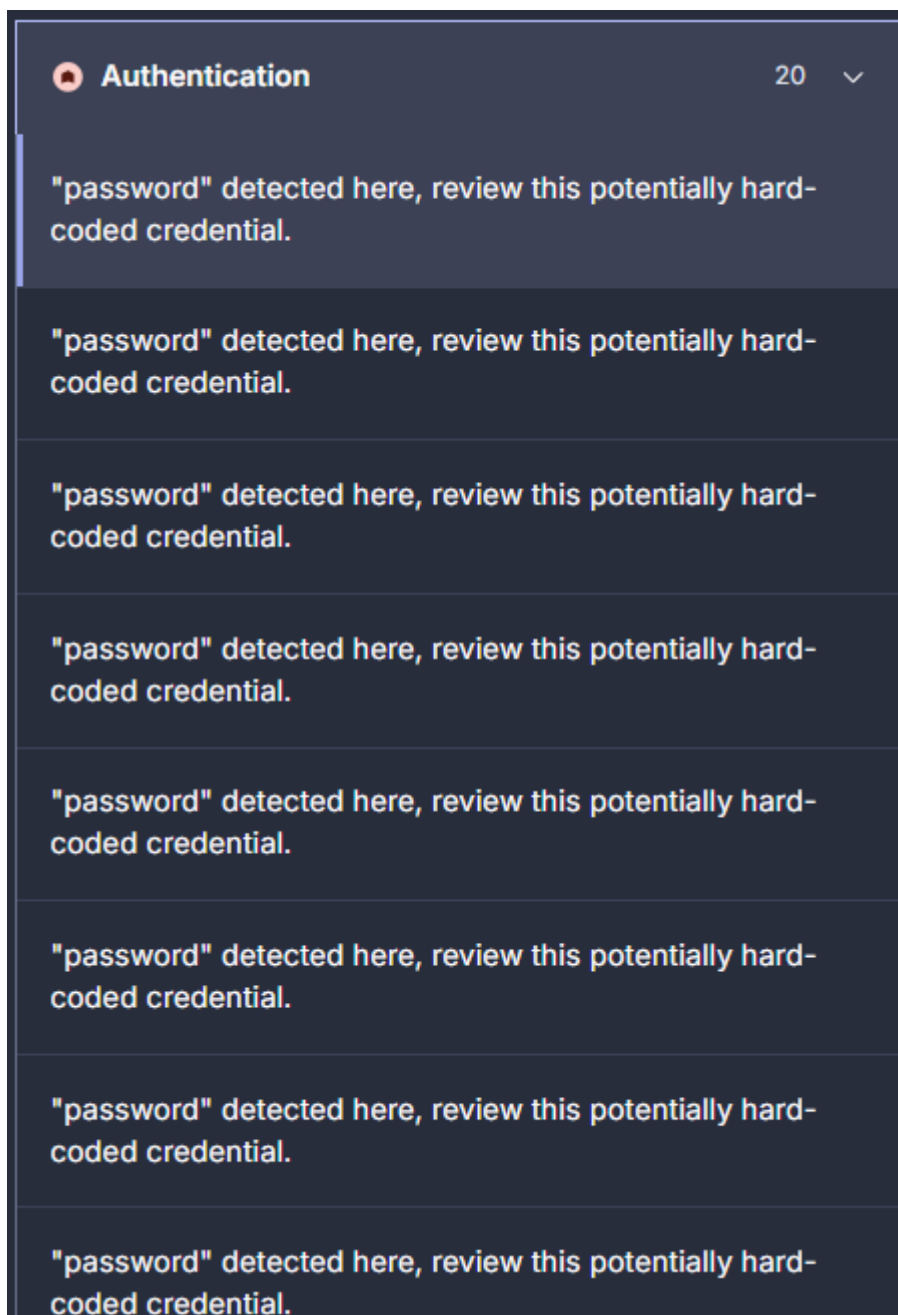
```
def generate_random_password():
    return ''.join(random.choice(string.ascii_lowercase + string.digits) for _ in
range(20))
```

## 4. Hot Spots

Link para acesso: [Sonar Cloud](#)

Acessando o Sonar Cloud, foi possível verificar os seguintes “Hot Spots” no projeto, seguindo os seguintes padrões de prioridade:

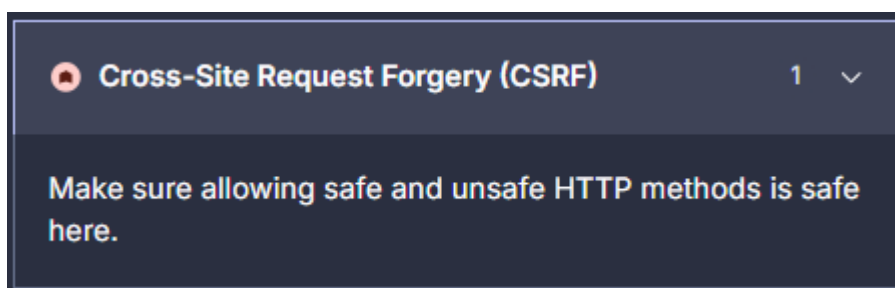
- **Alta Prioridade**
  - a. Foram encontrados 20 Hot Spots referentes a Autenticação, a maioria sendo de campos “password” que são inseridos de maneira crua no código, contudo, os citados, são apenas para testes e seeds para importação de campos para testes.



- b. Este outro único "Hot Spot" de testes alerta sobre o risco de incluir credenciais diretamente no código-fonte de uma aplicação. Como é fácil extrair strings do código ou de um binário, credenciais não devem ser hard-coded (inseridas diretamente no código). Isso é especialmente importante para aplicações distribuídas ou de código aberto. No passado, essa prática resultou em vulnerabilidades como CVE-2019-13466 e CVE-2018-15389.

As credenciais devem ser armazenadas fora do código, em arquivos de configuração, bancos de dados ou serviços de gerenciamento de segredos. Esta regra identifica credenciais hard-coded usadas em conexões com bancos de dados e LDAP, procurando por credenciais nas strings de conexão e em variáveis com nomes que correspondem a padrões específicos.

Recomenda-se personalizar a configuração dessa regra, adicionando palavras-chave relacionadas a credenciais, como o "authToken" e "secret", para aumentar a detecção.



- **Média Prioridade**

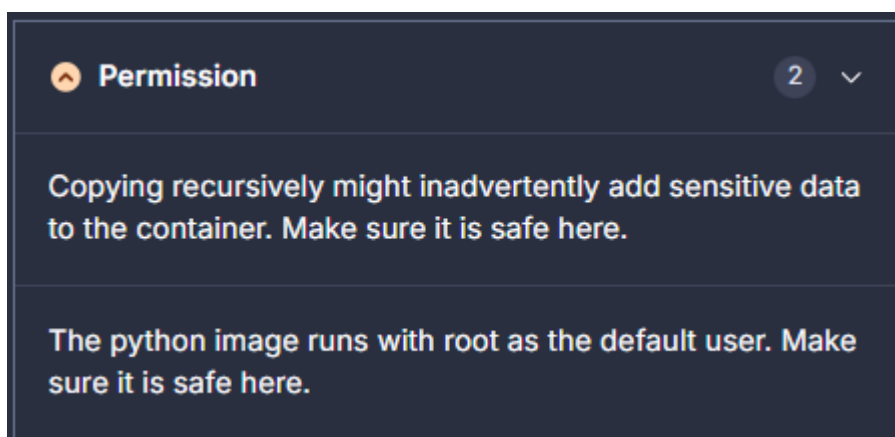
- a. Este “Hot Spot” se refere a dois erros de permissão identificados no código:

- **Cópia de Arquivos Inesperados em Imagens Docker:**

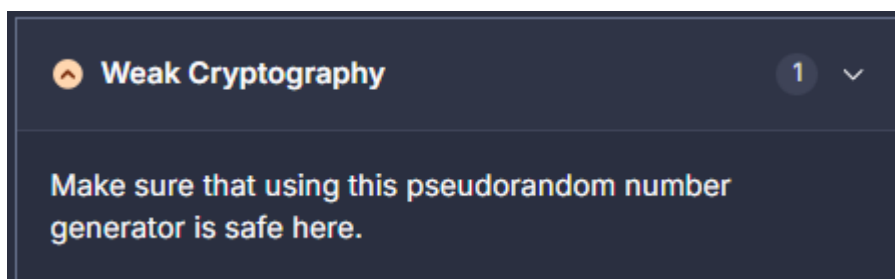
- Durante a construção de uma imagem Docker, o uso das diretivas `COPY` ou `ADD` pode copiar acidentalmente arquivos inesperados do diretório de contexto para o sistema de arquivos da imagem. Isso pode comprometer a confidencialidade dos dados.

- **Execução de Containers como Usuário Privilegiado:**

- Rodar containers com permissões de superusuário (root em Linux ou Container Administrador em Windows) enfraquece a segurança, permitindo que um usuário malicioso execute ações administrativas. Isso pode resultar na exfiltração de dados sensíveis, instalação de softwares maliciosos ou até na invasão de outros componentes da infraestrutura, como clusters Kubernetes ou provedores de nuvem.

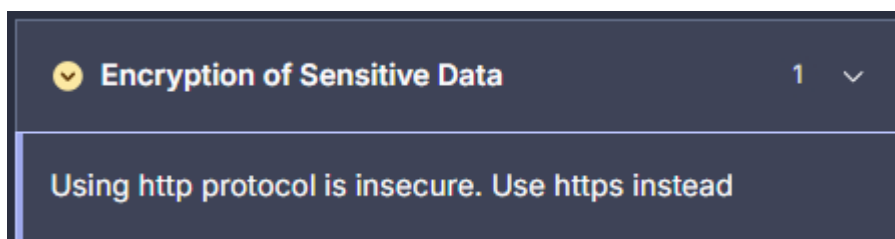


- b. O uso de geradores de números pseudoaleatórios (PRNGs) é sensível à segurança. No passado, isso causou vulnerabilidades como CVE-2013-6386, CVE-2006-3419 e CVE-2008-4102. Quando o software gera valores previsíveis em contextos que requerem imprevisibilidade, um atacante pode adivinhar o próximo valor gerado e usá-lo para se passar por outro usuário ou acessar informações sensíveis.



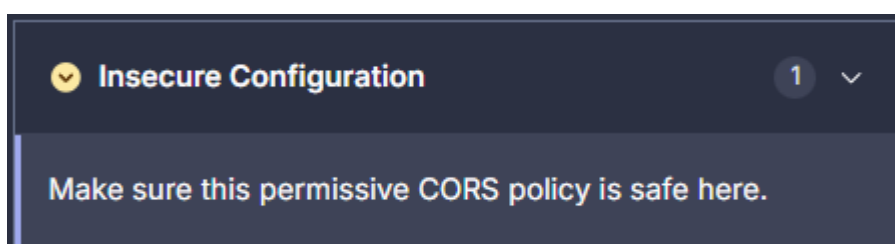
- **Baixa Prioridade**

- a. Protocolos em texto claro, como FTP, Telnet e HTTP, não criptografam dados e não autenticam conexões, permitindo que atacantes leiam, modifiquem ou corrompam informações. Isso pode resultar em exposição de dados sensíveis, redirecionamento para endpoints maliciosos, malware, execução de código não autorizado e corrupção de informações críticas. Mesmo em redes isoladas, criptografar comunicações e aplicar práticas de segurança como segmentação é essencial para reduzir riscos. Vulnerabilidades passadas incluem CVE-2019-6169, CVE-2019-12327 e CVE-2019-11065.

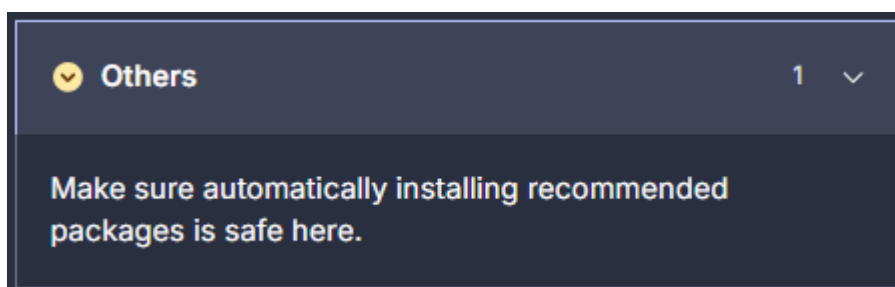




- b. Ter uma política de Cross-Origin Resource Sharing (CORS) permissiva é sensível à segurança. Isso pode levar a vulnerabilidades como CVE-2018-0269 e CVE-2017-14460. A política de mesma origem dos navegadores impede, por padrão, que um frontend JavaScript faça requisições HTTP para recursos com origens diferentes (domínio, protocolo ou porta). A resposta pode incluir cabeçalhos CORS que modificam essa política e relaxam o controle de acesso.

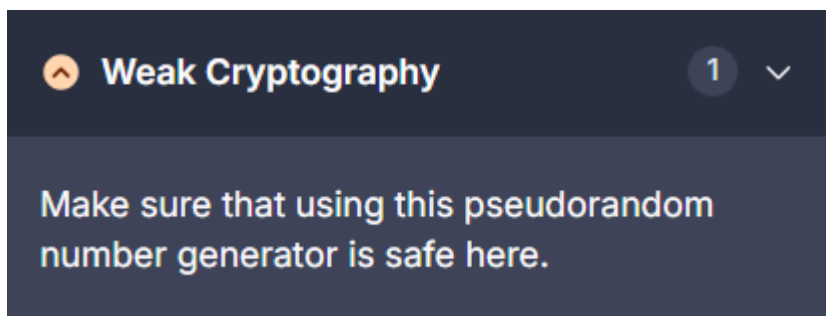


- c. Instalar pacotes recomendados automaticamente pode criar vulnerabilidades na imagem Docker. Pacotes desnecessários, instalados por gerenciadores de pacotes como o Debian, aumentam a superfície de ataque do container, pois podem conter vulnerabilidades ou código malicioso. Isso pode facilitar ataques à cadeia de suprimentos e permitir escalonamento de privilégios por atacantes. Remover pacotes não utilizados também reduz o tamanho da imagem Docker. Para segurança, remova pacotes desnecessários e faça varreduras de vulnerabilidades regularmente.



## 5. Análise das Vulnerabilidades ou Hot Spots

### a. Criptografia Fraca /Hotspot 4



```
def generate_random_password():
    return ''.join(random.choice(string.ascii_lowercase + string.digits) for _ in
range(20))
```

### b. Descrição

Como evidenciado na imagem acima, foi identificada na seção de pontos críticos uma função responsável pela geração de senhas aleatórias com um limite de 20 caracteres. Essa senha é criada no momento do cadastro de um usuário, que deve, em seguida, realizar o login e alterar a senha inicialmente fornecida. No entanto, em alguns casos, usuários podem postergar a alteração da senha por um período prolongado, o que constitui uma vulnerabilidade crítica. Isso representa um risco significativo de invasão, uma vez que a senha gerada com 20 caracteres pode não ser suficientemente robusta para proteger contra acessos indesejados.

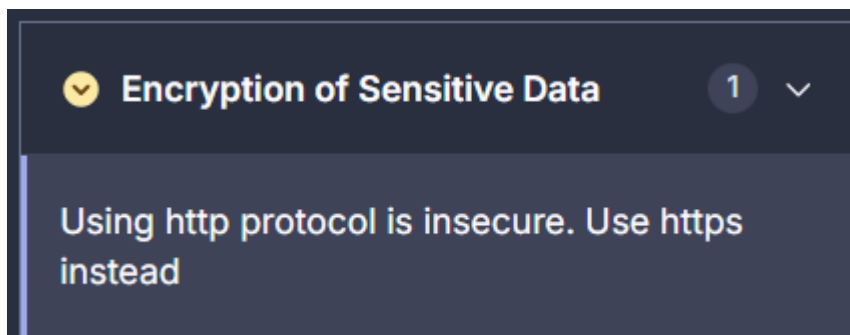
### c. Solução

Uma alternativa à biblioteca random utilizada na função em questão seriam bibliotecas como secrets e cryptography, que são mais recomendadas para a geração de senhas criptograficamente seguras. Dado que a senha é criada para o usuário no momento do cadastro, uma abordagem para garantir maior segurança seria restringir o acesso com a senha temporária a um curto período de tempo. Após esse período, o usuário deve solicitar ao sistema o envio de uma nova senha para o e-mail cadastrado, possibilitando o acesso e a subsequente alteração da senha. Essa medida tornaria o acesso ao login significativamente mais difícil para um potencial invasor.

## 6. Análise de Hot Spots individuais (PTOSS - 5)

Elias Faria de Oliveira - 221007706

**Hotspot:** Criptografia de dados



**Figura xx:** Hotspot de Criptografia de dados

### Descrição:

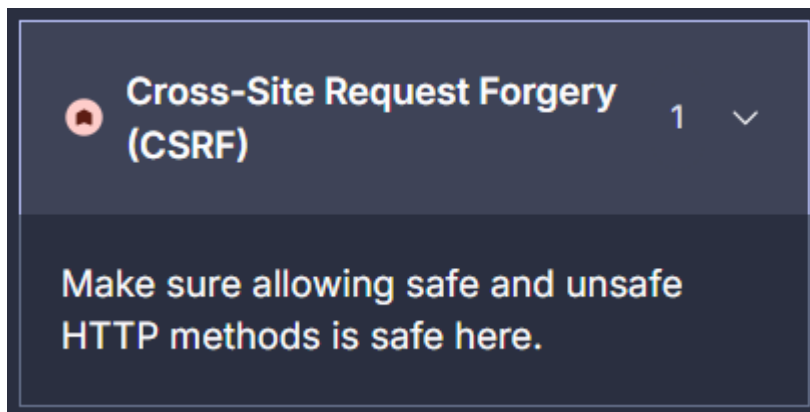
O protocolo HTTP é utilizado para comunicação na web através de texto, este não criptografa os dados que são apresentados na nuvem, expondo dados e gerando inseguranças e vulnerabilidades no seu uso. Existem diversos meios que usuários maliciosos podem utilizar para manipular, expor e prejudicar algum usuário por meio de falta de segurança na utilização do devido protocolo. Alguns dos possíveis problemas que podem ocorrer são: A exposição de dados, falsificação de links, redirecionamento a servidores maliciosos, corrupção de informações críticas. Tendo em vista o hotspot encontrado, vemos que o protocolo é utilizado em um servidor local, que mesmo tendo menores possibilidades de invasão e sofrer algum ataque existem alguns tipos de ataque que podem ocorrer, como: Comprometer componentes da rede, contornar mecanismos de isolamento, obter credenciais, entre outras.

### Solução:

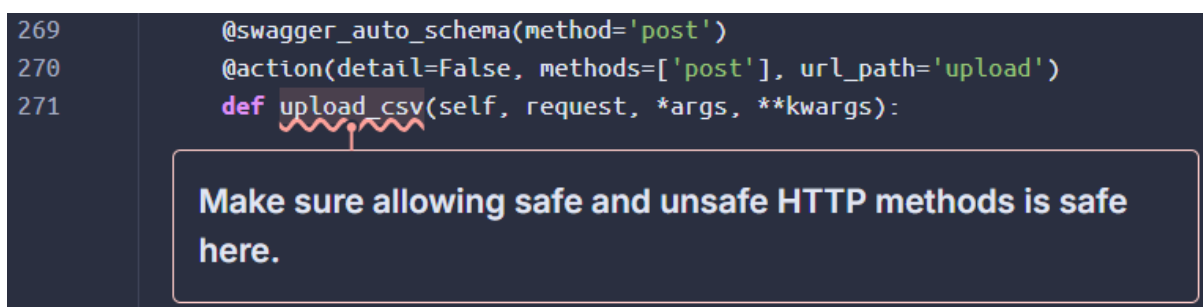
Para mitigar e praticamente reduzir a zero as inseguranças e fraquezas relacionadas a comunicação com texto web é: utilizar um protocolo web que criptografa as informações e dados utilizados, como o protocolo 'HTTPS', que realiza as mesmas funções do protocolo utilizado, mas utiliza a criptografia TLS( Transport Layer Security ) para transmitir os dados.

João Eduardo Pereira Rabelo - 180053299

**HotSpot:** Métodos HTTP inseguros na chamada de uma função



**Figura xx:** Hotspot de métodos HTTP inseguros. (Fonte: João Eduardo)



**Figura xx:** Código do Hotspot de métodos HTTP inseguros. (Fonte: João Eduardo)

### Descrição:

A ferramenta Sonar Cloud identificou um potencial problema de segurança relacionado ao uso de métodos HTTP em uma função chamada “upload\_csv”. Especificamente, o alerta indica que tanto métodos HTTP seguros (como GET, HEAD, OPTIONS) quanto inseguros (como POST, PUT, DELETE) podem estar sendo utilizados para realizar a mesma operação. Métodos seguros são usados para operações de leitura, enquanto métodos inseguros alteram o estado da aplicação. Misturar esses métodos em uma única operação pode expor a aplicação a vulnerabilidades, como ataques CSRF (Cross-Site Request Forgery), onde operações não autorizadas podem ser realizadas em nome de um usuário autenticado.

### **Problema Real ou Falso Positivo:**

Esse é um problema real. A preocupação levantada pela ferramenta Sonar Cloud é válida porque permitir que métodos HTTP inseguros sejam usados de maneira imprópria pode comprometer a segurança da aplicação. Se a função “`upload_csv`” permite que métodos como POST sejam usados sem as devidas proteções, há um risco significativo de que um atacante possa explorar essa vulnerabilidade, especialmente se a aplicação não implementar mecanismos robustos de proteção contra CSRF.

### **Sugestão de Solução:**

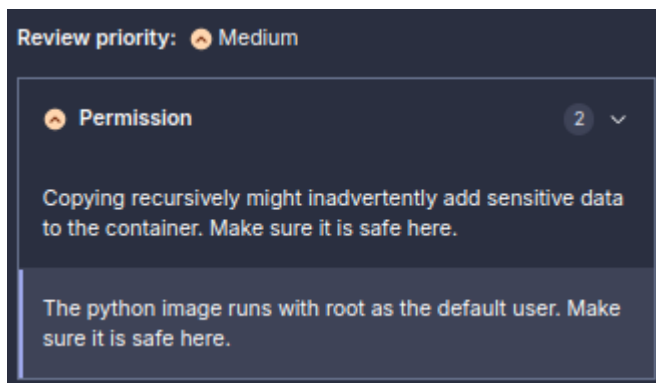
A solução recomendada é garantir que apenas métodos HTTP apropriados sejam usados na função “`upload_csv`”. Se a função deve alterar o estado da aplicação, por exemplo, realizando o upload de um arquivo CSV, deve-se restringir o uso ao método POST, que é o padrão para operações que alteram o estado. Além disso, deve-se implementar medidas de segurança adicionais, como verificações CSRF, para garantir que apenas solicitações legítimas possam executar operações potencialmente perigosas.

### **Conclusões sobre o Teste de Segurança Realizado:**

O teste de segurança realizado pela ferramenta Sonar Cloud foi eficaz em identificar um ponto potencialmente vulnerável na aplicação. Ao alertar sobre o uso inadequado de métodos HTTP, o teste destaca a importância de seguir as melhores práticas de segurança na implementação de APIs e funções que manipulam dados sensíveis ou alteram o estado da aplicação. A resolução do problema não apenas melhora a segurança da aplicação, mas também aumenta a sua resiliência contra ataques comuns na web.

**Daniel dos Santos Barros de Sousa - 211030980**

**Hotspot** : Permissão do Docker



**Figura xx** - Hotspot de Permissão. (Fonte: Daniel)

### **Descrição:**

Existe um arquivo docker que sobre um container executando alguns comandos, sendo que alguns deles só podem ser executados com um usuário com privilégios elevados, e é isso que ele faz do começo ao fim da sua execução.

### **Análise - Problema Real:**

Uma vez que do começo ao fim o usuário tem privilégios elevados, qualquer comando dentro deste contêiner vai ser feito com privilégios elevados, ou seja, se alguém de fora conseguir acesso ao container através de alguma vulnerabilidade, ele terá acesso a privilégios elevados.

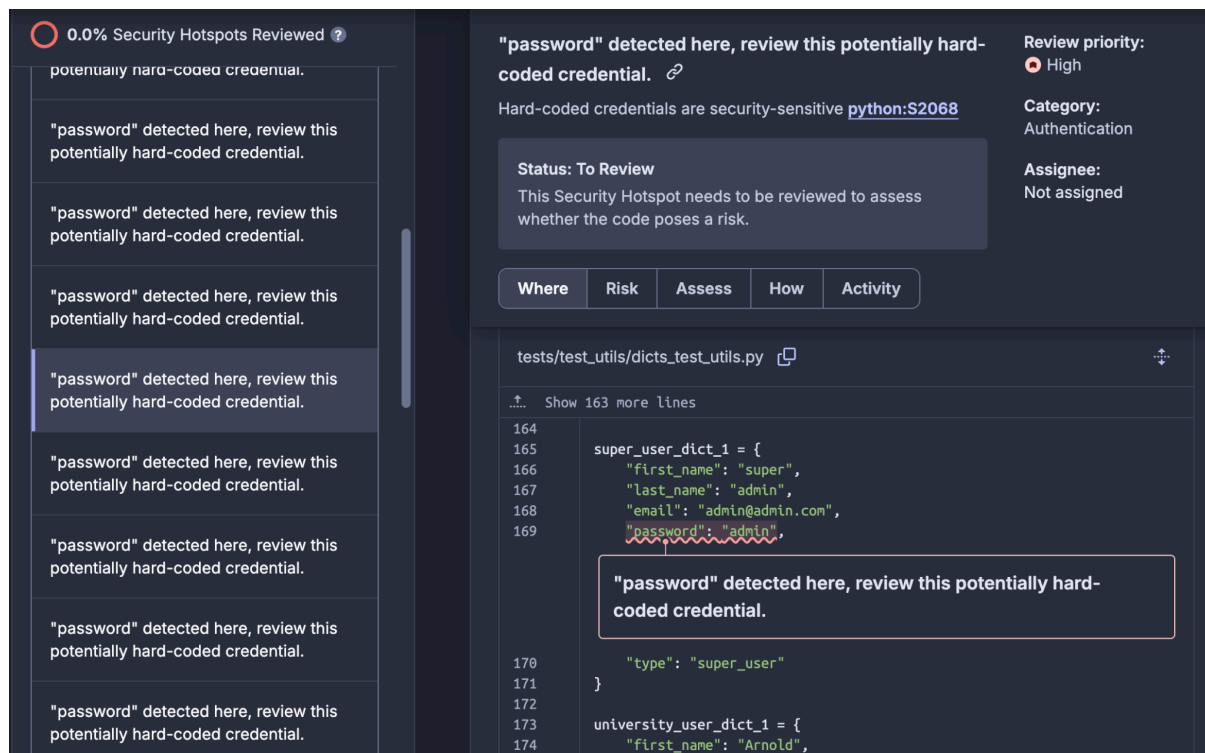
Fato esse que fica mais grave se o container não estiver reforçado para impedir o uso de um Shell, interpretador ou recursos do SO, isso faz com que o invasor do sistema consiga acesso de leitura e modificação de qualquer arquivo presente, como também manipular as conexões do docker e instalações, e em casos muito graves, sair do isolamento do container e explorar o resto da arquitetura.

### **Solução:**

Uma solução possível, é que no arquivo docker se crie um novo usuário, assim, se execute todos os comandos que se deseje com o root e no fim cria-se um novo usuário, que vai ser o usuário enquanto o container estiver rodando, assim, em questões de exploração de vulnerabilidade relacionado ao container, o estrago será mais contido.

Yan Lucas Souza Guimarães - 222006220

## Hotspot: Detecção de credenciais codificadas



The screenshot displays a security tool interface with a dark theme. On the left, a sidebar shows a list of detected issues, all stating: "potentially hard-coded credential." The main panel on the right shows a detailed view of one such issue. It highlights a line of Python code in a file named `tests/test_utils/dicts_test_utils.py` at line 169: `"password": "admin",`. A red squiggly line underlines the value `"admin"`. A tooltip box points to this line with the message: "password" detected here, review this potentially hard-coded credential. To the right of the code view, a summary box indicates the issue is "To Review", has a "High" priority, and is in the "Authentication" category. Below the code, there are tabs for "Where", "Risk", "Assess", "How", and "Activity".

**Figura xx:** Detecção de credenciais codificadas. (Fonte: Yan Lucas)

### Descrição:

A presença de credenciais embutidas. O revisor enfatiza que incluir informações sensíveis, como credenciais de banco de dados ou LDAP, diretamente no código-fonte é uma prática perigosa, especialmente em aplicações distribuídas ou de código aberto. Foram citadas vulnerabilidades específicas (CVE-2019-13466 e CVE-2018-15389) que surgiram por conta de problemas semelhantes no passado.

### Análise - Falso positivo:

Por conta do código que contém as credenciais embutidas ser de um teste e não do código de produção, a afirmação é falsa em relação a ser uma preocupação crítica de segurança. Embora não seja uma prática recomendada incluir credenciais diretamente no código de teste, o risco não é o mesmo que no código de produção.

### Solução:

No entanto, ainda é aconselhável evitar a hard-coding de informações sensíveis em código de teste, utilizando variáveis de ambiente, dados mock, ou arquivos de configuração de teste.