FGA-0238 - Testes de Software

Turma	01	**Semestre**	2024.1

Equipe	NeverMind

Nome	Matrícula
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:https://gitlab.com/lappis-unb/projetos-energia/mec-energia/mec-energia-api

2. **Descrição**

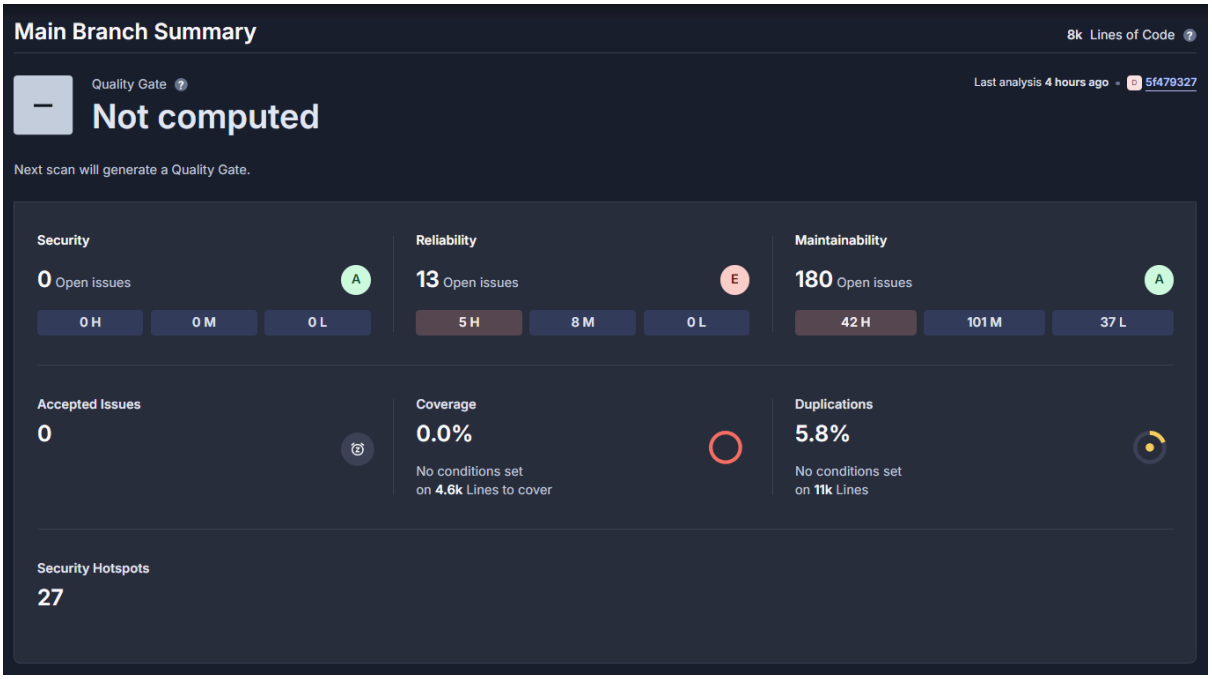
   O projeto **MEC Energia API** é uma aplicação desenvolvida pelo Laboratório Avançado de Produção, Pesquisa e Inovação em Software (LAPPIS) da Universidade de Brasília. Esta API faz parte de uma iniciativa para gerenciar e otimizar o consumo de energia elétrica em instituições educacionais. A API oferece 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.

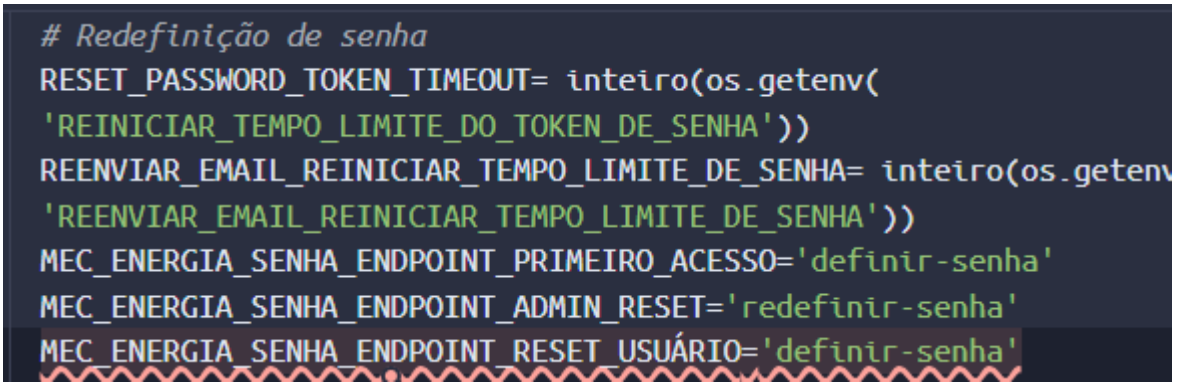
Vulnerabilidades: Embora os hotspots indiquem possíveis problemas, é necessário investigar se essas questões representam vulnerabilidades reais ou falsos positivos. No caso da função mencionada, o risco está na previsibilidade do PRNG, o que poderia comprometer a segurança dos usuários. Para garantir 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

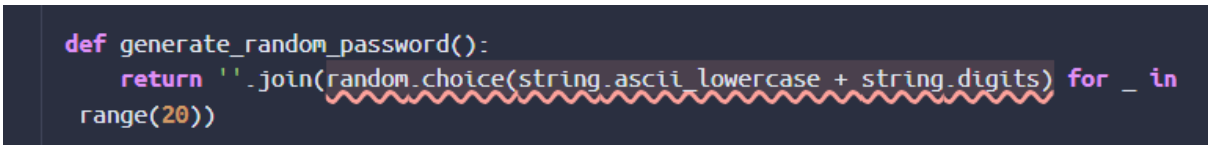
Acessando o Sonar Cloud, foi possível verificar as seguintes vulnerabilidades no projeto:

- 1. 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, as 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'
```

- 2. 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.

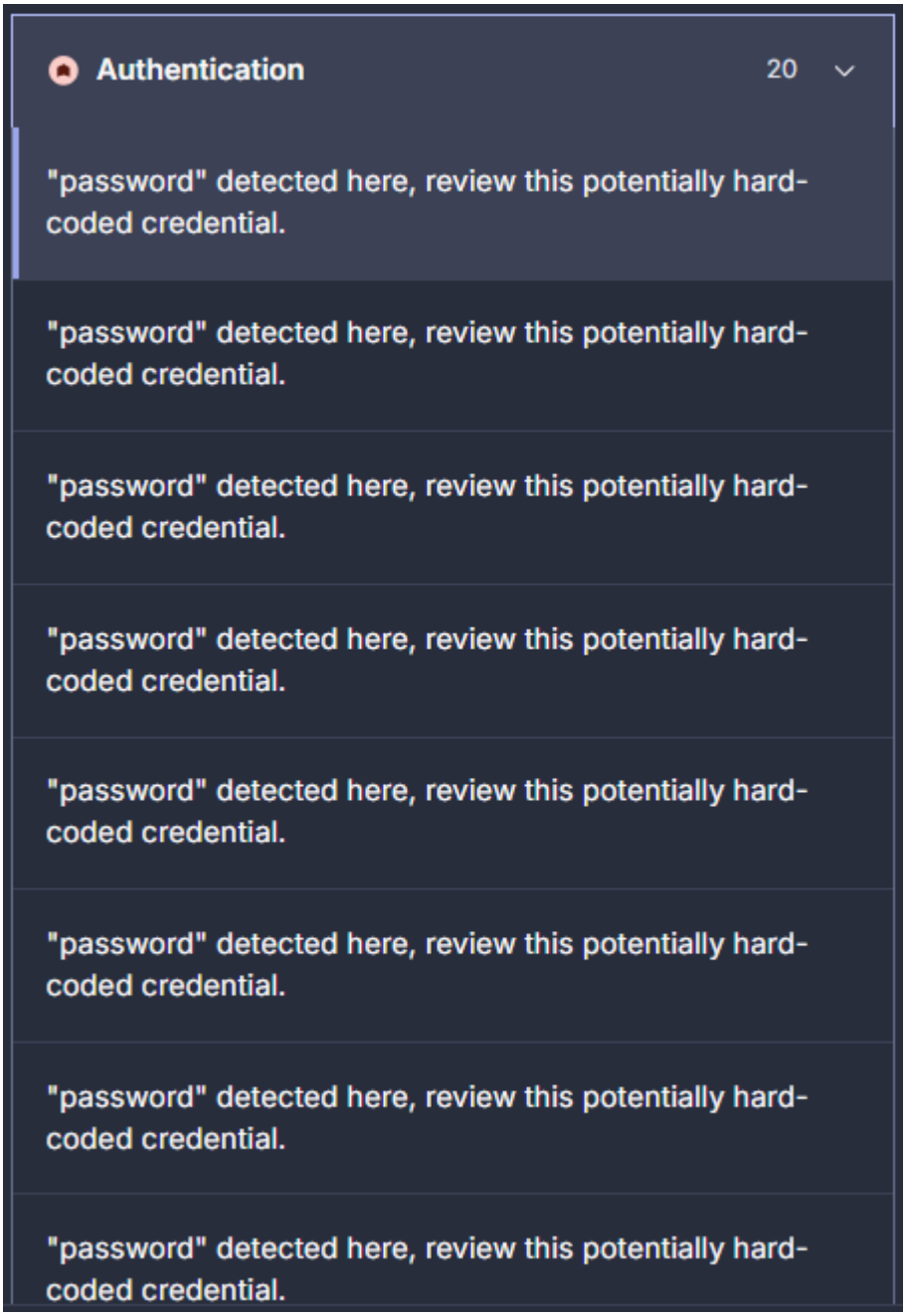


```
def generate_random_password():
    return ''.join(random.choice(string.ascii_lowercase + string.digits) for _ in
 range(20))
```

4. Hot Spots

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

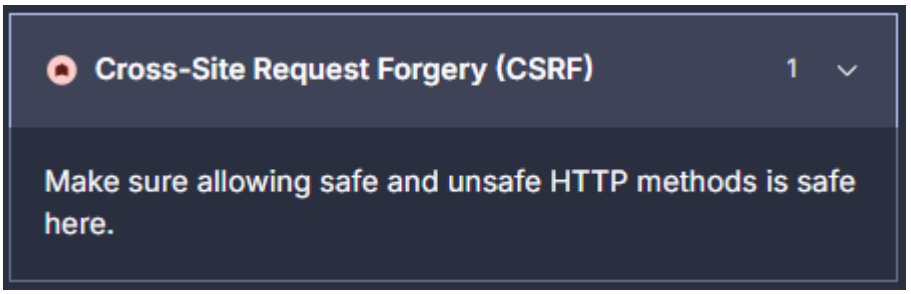
- Alta Prioridade
    a. Foram encontrados 20 Hot Spots referentes a Autenticação, a maioria sendo de campos “password” que são inseridos de maneira crua no código, contudo, os citados, são apenas para testes e seeds para importação de campos para testes.



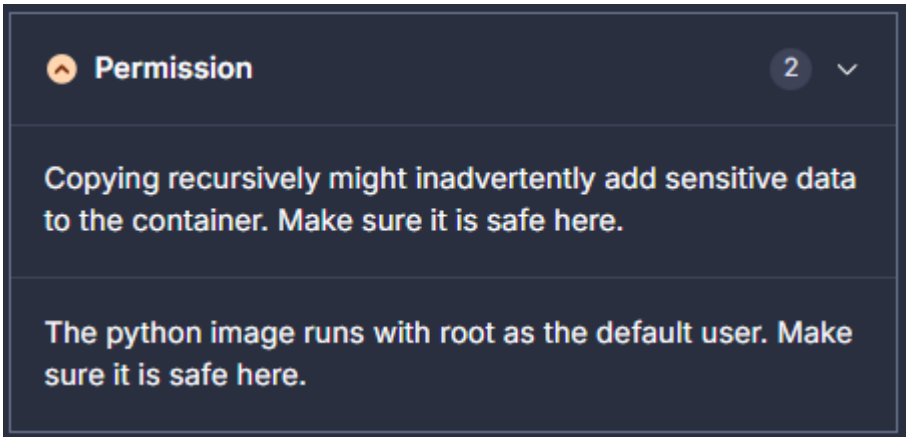
b. Este outro unico "Hot Spot" de testes alerta sobre o risco de incluir credenciais diretamente no código-fonte de uma aplicação. Como é fácil extrair strings do código ou de um binário, credenciais não devem ser hard-coded (inseridas diretamente no código). Isso é especialmente importante para aplicações 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.

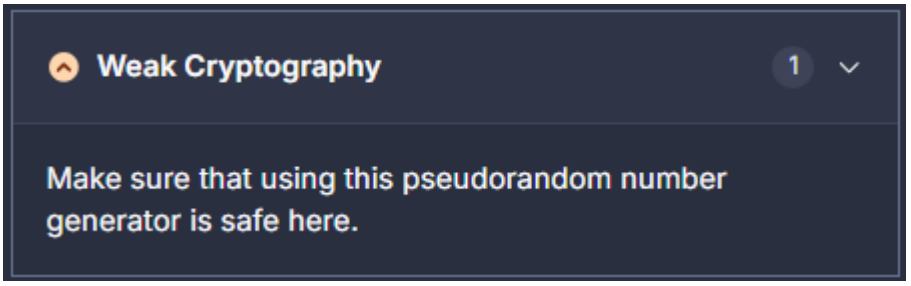
   Recomenda-se personalizar a configuração dessa regra, adicionando palavras-chave relacionadas a credenciais, como "oauthToken" e "secret", para aumentar a detecção.



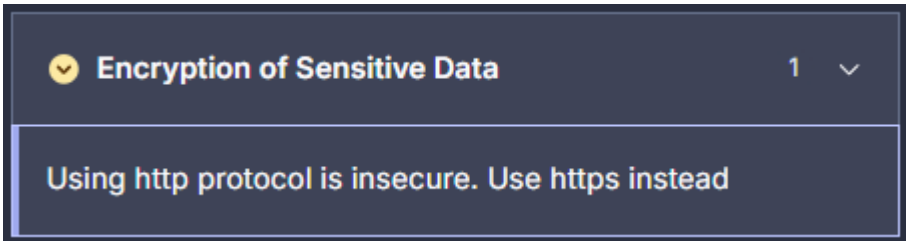
- Média Prioridade
    a. Este “Hot Spot” se refere a dois erros de permissão identificados no código:
        - **Cópia de Arquivos Inesperados em Imagens Docker:** Durante a construção de uma imagem Docker, o uso das diretivas `COPY` ou `ADD` pode copiar acidentalmente arquivos inesperados do diretório 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.



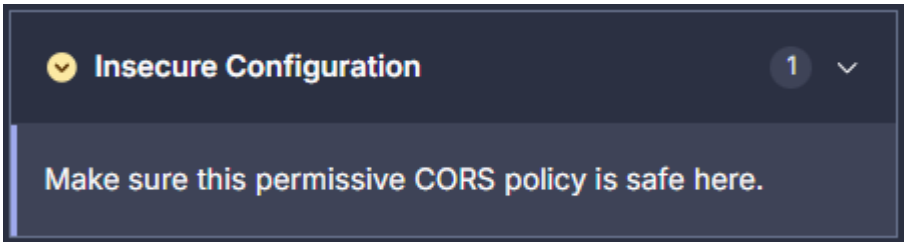
b. O uso de geradores de números pseudoaleatórios (PRNGs) é sensível à segurança. No passado, isso causou vulnerabilidades como CVE-2013-6386, CVE-2006-3419 e CVE-2008-4102. Quando o software gera valores previsíveis em contextos que requerem imprevisibilidade, um atacante pode adivinhar o próximo valor 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.



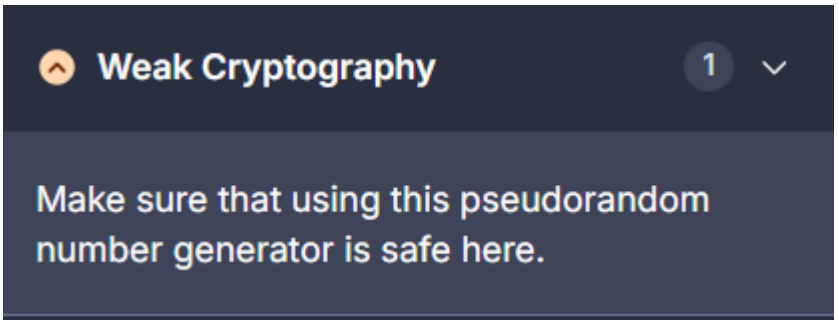
b. Ter uma política de Cross-Origin Resource Sharing (CORS) permissiva é sensível à segurança. Isso pode levar a vulnerabilidades como CVE-2018-0269 e CVE-2017-14460. A política de mesma origem dos navegadores impede, por padrão, que um frontend JavaScript faça requisições HTTP para recursos com origens 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

5.1. < Criptografia Fraca /Hotspot 4>




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

5.1.1. 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.

5.1.2. 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.