

PTOSS 5
Turma: 01
Equipe: TLE
Alunos:

Marcos Castilhos - 221008300
Carlos Henrique de S - 211061529
Guilherme Coelho - 202016364

Eric Akio Lages Nishimura - 190105895
João Pedro Veras Gomes - 211061968
Breno Queiroz Lima - 211063069

Insecure Configuration - CORS (Carlos)



```
backend/src/app.ts
... Show 11 more lines
12 function buildApp(): express.Application {
13   const app = express();
14
15   app.use(urlencoded({ extended: true }));
16   app.use(json());
17   app.use(cors());
18
19   app.use(helmet());
20   app.set("trust proxy", 1);
21
22   app.use(contextMiddleware);
... Show 13 more lines
```

Make sure that enabling CORS is safe here.

Entendendo o Problema

A ferramenta de análise estática de código identificou um potencial problema relacionado à configuração do CORS (Cross-Origin Resource Sharing) na linha 17 do arquivo backend/src/app.ts. A mensagem de alerta "Make sure that enabling CORS is safe here" indica que a ferramenta considera que permitir o CORS em todas as origens pode apresentar riscos de segurança.

É um Falso Positivo?

Não, não é necessariamente um falso positivo. Embora o CORS seja uma ferramenta essencial para permitir que aplicações front-end em diferentes domínios se comuniquem com um back-end, ele também pode ser explorado por atacantes.

Por que a preocupação com o CORS?

Ataques CSRF (Cross-Site Request Forgery): Um atacante pode criar um link malicioso que, quando clicado por um usuário autenticado, executa uma ação indesejada no seu servidor.

Ataques de injeção: Se não configurado corretamente, o CORS pode permitir que um atacante injete dados maliciosos em suas requisições.

Solução Proposta

Configurar o CORS de forma restrita: Em vez de permitir o CORS para todas as origens, é altamente recomendável restringir o acesso a apenas as origens confiáveis.

```
77 app.use(cors({
78   origin: ['http://seu-front-end.com', 'https://seu-outro-front-end.com'],
79   methods: ['GET', 'POST', 'PUT', 'DELETE'],
80   allowedHeaders: ['Content-Type', 'Authorization']
81 }));
82
```

Explicação da configuração

Origin: Lista as URLs das aplicações front-end autorizadas a fazer requisições.

Methods: Especifica os métodos HTTP permitidos (GET, POST, PUT, DELETE, etc.).

AllowedHeaders: Define os cabeçalhos HTTP permitidos (Content-Type, Authorization, etc.).

Análise Geral da Segurança da Aplicação

Com base apenas neste trecho de código, é difícil fazer uma análise completa da segurança da aplicação. No entanto, a configuração do CORS é um ponto crucial e a sua resolução já é um passo importante.

Outras considerações para uma análise mais profunda

Validação de dados: Certifique-se de validar todos os dados de entrada para evitar injeções de SQL, XSS e outras vulnerabilidades.

Gerenciamento de senhas: Utilize hashes seguros para armazenar senhas e implemente políticas de autenticação fortes.

Proteção contra ataques DDoS: Considere utilizar soluções para mitigar ataques de negação de serviço.

Atualizações de dependências: Mantenha todas as bibliotecas e frameworks utilizados na aplicação atualizados para corrigir vulnerabilidades conhecidas.

Recomendação

Recomenda-se realizar uma análise de segurança mais abrangente da aplicação, utilizando ferramentas de varredura de vulnerabilidades e testes de penetração. Além disso, seguir as melhores práticas de desenvolvimento seguro pode ajudar a prevenir a maioria das vulnerabilidades comuns.

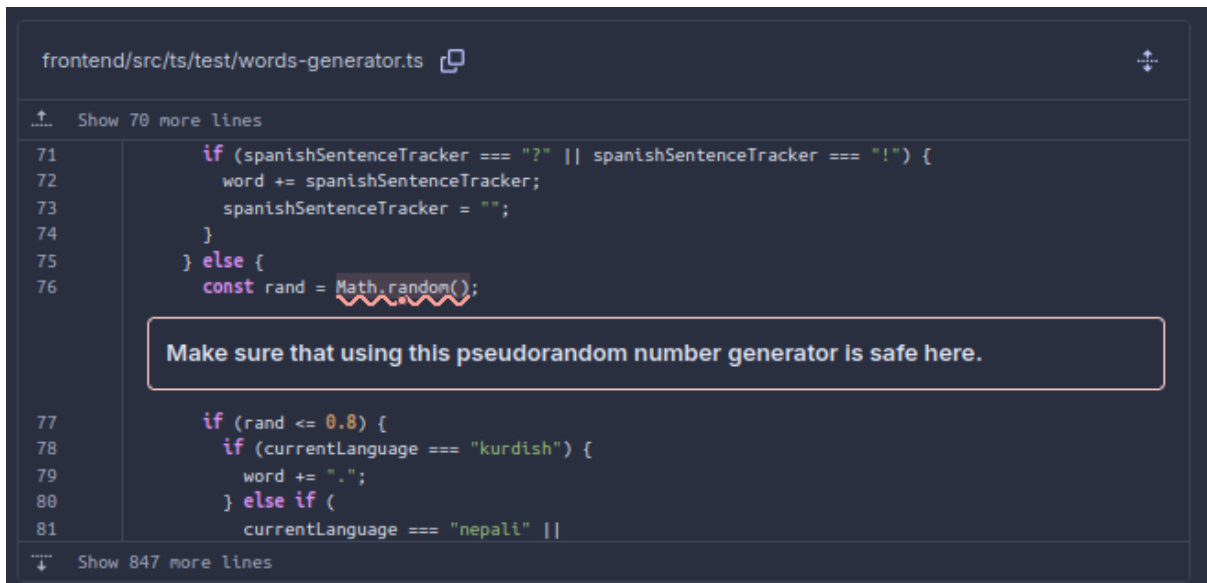
Observações

Ferramentas de análise: Existem diversas ferramentas disponíveis para realizar análises de segurança de código, como SonarQube, Snyk e OWASP ZAP.

Linguagem de programação: A configuração do CORS pode variar ligeiramente dependendo da linguagem de programação e do framework utilizado.

Contexto da aplicação: A configuração ideal do CORS pode variar dependendo da arquitetura da aplicação e dos requisitos específicos do projeto.

Weak Cryptography - Eric



The screenshot shows a code editor with a file named `frontend/src/ts/test/words-generator.ts`. The code is in TypeScript and includes a function that builds a word based on a `spanishSentenceTracker`. A warning box is overlaid on the code, pointing to the `Math.random()` function call on line 76. The warning text reads: "Make sure that using this pseudorandom number generator is safe here." The code snippet is as follows:

```
71     if (spanishSentenceTracker === "?" || spanishSentenceTracker === "!") {
72         word += spanishSentenceTracker;
73         spanishSentenceTracker = "";
74     }
75     } else {
76         const rand = Math.random();
77
78         if (rand <= 0.8) {
79             if (currentLanguage === "kurdish") {
80                 word += ".";
81             } else if (
82                 currentLanguage === "nepali" ||
```


Entendendo o problema:

Foi apresentado pela ferramenta a mensagem de erro “Make sure that using this pseudorandom number generator is safe here”, apontando como possível problema de criptografia um gerador de número aleatório.

É um falso positivo?

Sim, uma vez que, nesse contexto, pelo número aleatório gerado ser obrigatoriamente entre 0 e 1, tem nuances o suficiente para definir onde e como é pontuado aleatoriamente a frase do teste de digitação.

Denial of Service (DoS) - (João Pedro Veras)



```
packages/release/src/buildChangelog.js

248 // });
249
250 //split message using regex based on fix(language): spelling mistakes in Nepali wordlist and quotes
(sapradhan) (#4528)
//scope is optional, username is optional, pr number is optional
251 const [, type, scope, message, message2, message3] = title.split(
252 //^(\\w+)(?:\\((\\[\\^\\]\\\\)\\)?;\\s+(.+)?\\s*(?:\\((\\[\\^\\]\\\\)\\)?(?:\\s+\\((\\[\\^\\]\\\\)\\)?(?:\\s+\\((\\[\\^\\]\\\\)\\)?\\s/
253
254 );
255
256 const usernames = message2 && message3 ? message2.split(", ") : [];
257
258 let pr;
```

Make sure the regex used here, which is vulnerable to super-linear runtime due to backtracking, cannot lead to denial of service.

Entendendo o Problema

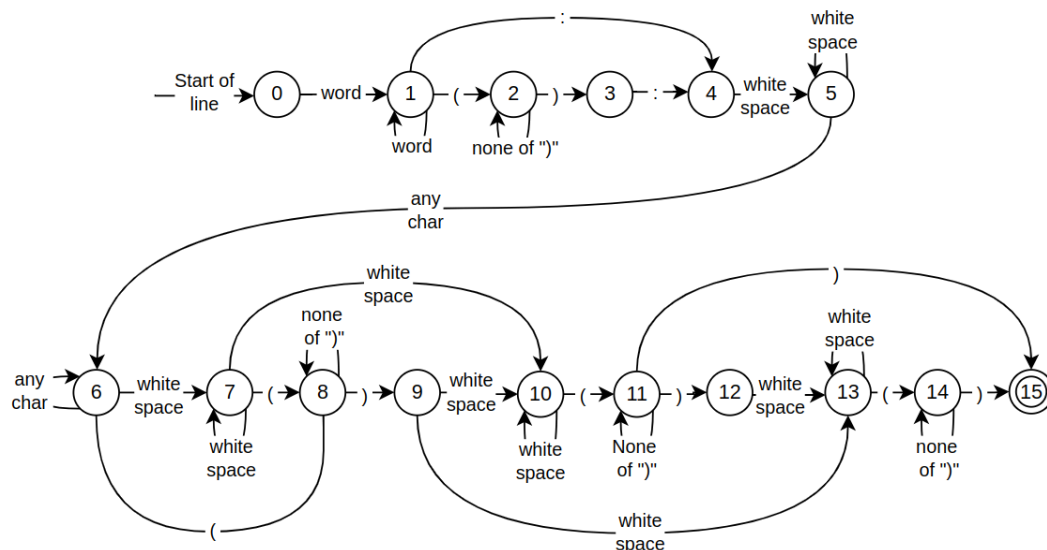
Foi apontado um risco de ataque DoS (Denial of Service), ou ataque de negação de serviço, que consiste em invalidar um servidor através de sobrecarga de recursos, como memória ou processamento, impedindo que o servidor proveja seus serviços normalmente aos clientes.

No caso apontado, em particular, o potencial ataque DoS ocorreria através de uma Regex, ou expressão regular. Segundo [este artigo da OWASP](#), certos tipos de Regex são problemáticos se gerarem AFNs (Autômatos Finitos Não-determinísticos).

As AFNs têm pelo menos um estado com transições ambíguas (i.e., ao menos duas de suas transições possuem o mesmo rótulo/símbolo de transição). Por isso, o processamento desse tipo de Regex está sujeito, para certas entradas, a arrendimentos/regressões na análise. Isso pode ser danoso caso entradas maliciosas sejam introduzidas, pois a complexidade da análise, que a princípio deveria ser linear, pode ser exponencial, o que consoma o ataque DoS.

É um Falso Positivo?

Para verificar se o caso observado é uma vulnerabilidade real, deve-se identificar se a Regex em questão corresponde a um AFN. Abaixo está o diagrama de estados do autômato correspondente à Regex em questão:



Percebe-se que o autômato acima é não-determinístico, dada a ambiguidade de transições nos estados 7, 9 e 11. Mesmo que as entradas sejam verificadas antes da análise da Regex, sua má forma ainda pode apresentar risco, principalmente para versões futuras do código.

Solução Proposta

Recomenda-se a conversão da Regex para uma forma determinística.

Permission

Entendendo o Problema

O trecho do arquivo Dockerfile da imagem abaixo utiliza uma imagem com permissão de usuário root. Isso significa que caso um atacante consiga acesso ao container ele poderá executar qualquer comando sem a necessidade de senha.

```
docker/frontend/Dockerfile
22 #build
23 RUN npm i --frozen-lockfile
24 RUN npm run build
25
26 # COPY to target
27 FROM nginx:mainline-alpine
28
29 COPY --from=builder /app/frontend/dist /usr/share/nginx/html
30 COPY docker/frontend/updateConfig.sh /docker-entrypoint.d/updateConfig.sh
31 RUN chmod +x /docker-entrypoint.d/updateConfig.sh
32
33 # entry
```

The nginx image runs with root as the default user. Make sure it is safe here.

É falso positivo?

Não, caso um atacante consiga acesso ao container, ele poderá executar qualquer comando, inclusive derrubar o serviço.

Solução proposta

Para lidar com esse problema as imagens dockers possuem o Rootless mode, que permite a execução do daemon Docker e seus containers como usuários no-root, para mitigar possíveis vulnerabilidades.

O problema pode ser solucionado adicionando algumas linhas extras do arquivo Dockerfile, como mostra a imagem abaixo

```
# COPY to target

FROM nginx:mainline-alpine
COPY --from=builder /app/frontend/dist /usr/share/nginx/html
COPY docker/frontend/updateConfig.sh /docker-entrypoint.d/updateConfig.sh
RUN chmod +x /docker-entrypoint.d/updateConfig.sh

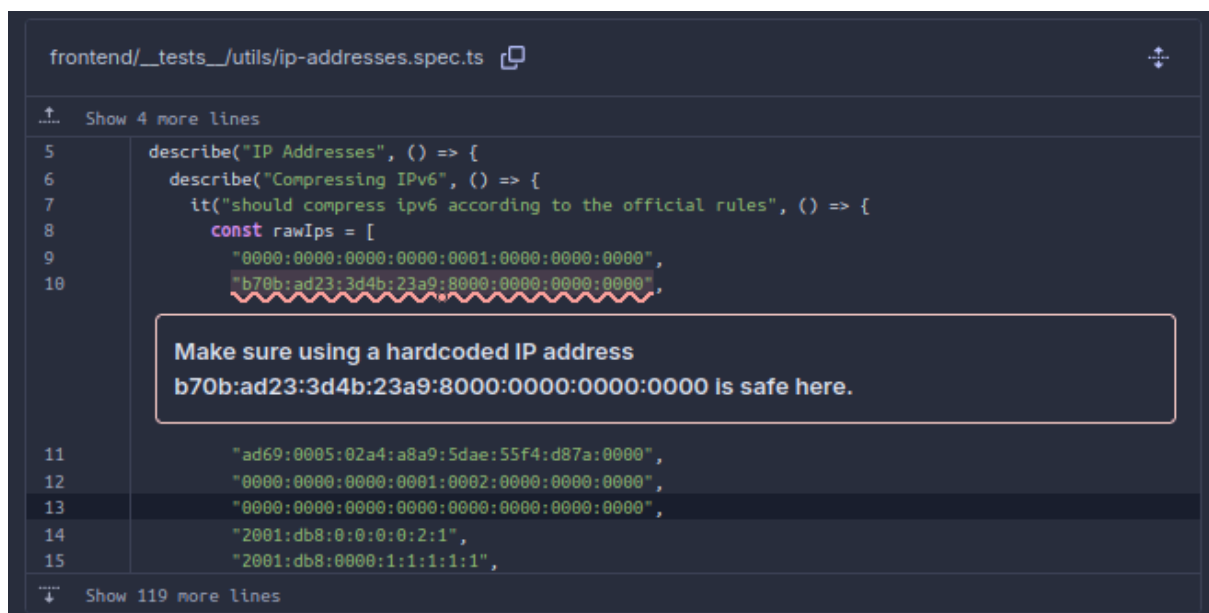
RUN addgroup -S nginxgroup && adduser -S nginxuser -G nginxgroup

RUN chown -R nginxuser:nginxgroup /var/cache/nginx /var/run /var/log/nginx

USER nginxuser
```

As linhas extras de código criam um novo usuário em um grupo de permissão diferente, e define este como usuário.

Hardcoded IP Address (Guilherme Coelho)



```
frontend/__tests__/utils/ip-addresses.spec.ts

5 describe("IP Addresses", () => {
6   describe("Compressing IPv6", () => {
7     it("should compress ipv6 according to the official rules", () => {
8       const rawIps = [
9         "0000:0000:0000:0000:0001:0000:0000:0000",
10        "b70b:ad23:3d4b:23a9:8000:0000:0000:0000",
11
12        "ad69:0005:02a4:a8a9:5dae:55f4:d87a:0000",
13        "0000:0000:0000:0001:0002:0000:0000:0000",
14        "0000:0000:0000:0000:0000:0000:0000:0000",
15        "2001:db8:0:0:0:2:1",
16        "2001:db8:0000:1:1:1:1:1",
17      ];
18    });
19  });
20 }
```

Compreendendo o Problema

A análise estática de código detectou um potencial risco na linha 10 do arquivo `frontend/__tests__/utils/ip-addresses.spec.ts` devido ao uso de um endereço IP embutido diretamente no código. O aviso "Certifique-se de que o uso

de um endereço IP fixo b70b:ad23:3d4b:23a9:8000:0000:0000:0000 seja seguro aqui" sugere que essa prática pode acarretar problemas de segurança.

É um Falso Positivo?

Não necessariamente. Embora possa haver situações em que o uso de endereços IP fixos seja justificado, especialmente em cenários de teste, essa prática ainda pode causar problemas de segurança e dificultar a manutenção do código.

Por que a Preocupação com IPs Hardcoded?

Segurança: Incluir um endereço IP diretamente no código pode revelar informações confidenciais sobre a infraestrutura da aplicação, como a localização de serviços internos ou servidores de produção. Isso pode ser explorado por invasores para obter acesso não autorizado ou realizar ataques específicos.

Manutenção: Deixar endereços IP fixos no código torna as mudanças e atualizações mais complicadas. Se o endereço IP precisar ser alterado, todas as ocorrências no código terão que ser modificadas manualmente, aumentando o risco de erros e inconsistências.

Ambientes Diferentes: Endereços IP fixos podem gerar problemas quando o código é transferido entre diferentes ambientes (como desenvolvimento, teste e produção). Um endereço IP que funciona em um ambiente pode não ser adequado ou válido em outro.

Solução Proposta

Trocar o Endereço IP Hardcoded por uma Variável de Ambiente ou Configuração Externa

Em vez de definir o endereço IP diretamente no código, recomenda-se utilizar uma variável de ambiente ou um arquivo de configuração externo para armazená-lo. Isso proporciona mais flexibilidade e aumenta a segurança, permitindo ajustes mais fáceis e seguros conforme necessário.

Ex:

```
test.py > ...
1  const TEST_IPv6 = process.env.TEST_IPv6 || 'default:ipv6:address:here';
2
3  describe('IP Addresses', () => {
4    describe('Compressing IPv6', () => {
5      it('should compress IPv6 according to the official rules', () => {
6        const rawIPs = [
7          TEST_IPv6,
8          "0000:0000:0000:0000:0001:0000:0000:0000",
9          // Outros endereços IP de teste
10       ];
11
12       // Implementação do teste
13       rawIPs.forEach(ip => {
14         const compressedIP = compressIPv6(ip);
15
16       });
17     });
18   });
```

Explicação da Configuração

Variável de Ambiente (`process.env.TEST_IPv6`): Permite que o endereço IP seja configurado fora do código-fonte, facilitando mudanças sem a necessidade de modificar o código.

Valor Padrão: Um valor padrão pode ser fornecido para ser usado se a variável de ambiente não estiver definida. Isso é útil em ambientes de teste onde um IP padrão é aceitável.

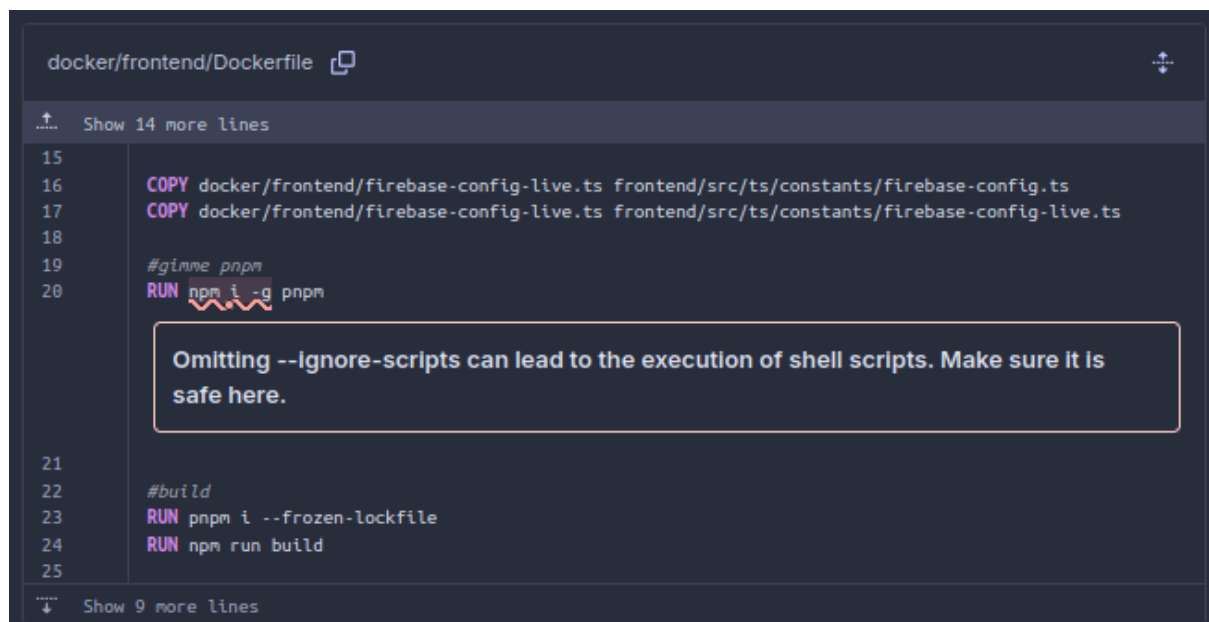
Segurança: Lidar adequadamente com endereços IP hardcoded é um passo importante para garantir que o código seja seguro e fácil de manter.

Recomendação:

Utilizar ferramentas de varredura de vulnerabilidades, testes de penetração e seguir as melhores práticas de desenvolvimento seguro para prevenir a maioria das vulnerabilidades comuns. Existem diversas ferramentas disponíveis para realizar

análises de segurança de código, como SonarQube, Snyk e OWASP ZAP. Podendo ser adaptada para outras linguagens e frameworks conforme necessário, sabendo que a solução ideal pode variar dependendo do contexto da aplicação e dos requisitos específicos do projeto.

Omitting --ignore-scripts (Marcos Castilhos)



The screenshot shows a code editor window titled 'docker/frontend/Dockerfile'. The code is as follows:

```
15
16 COPY docker/frontend/firebase-config-live.ts frontend/src/ts/constants/firebase-config.ts
17 COPY docker/frontend/firebase-config-live.ts frontend/src/ts/constants/firebase-config-live.ts
18
19 #give me pnpm
20 RUN npm i -g pnpm

21
22 #build
23 RUN pnpm i --frozen-lockfile
24 RUN npm run build
25
```

A warning box is overlaid on the code, stating: 'Omitting --ignore-scripts can lead to the execution of shell scripts. Make sure it is safe here.'

Entendendo o Problema

O código acima é parte de um arquivo Dockerfile utilizado para construir uma imagem Docker. O SonarCloud está alertando sobre um possível problema de segurança relacionado à execução de scripts durante a instalação de pacotes usando pnpm. Especificamente, o alerta menciona que a omissão da flag `--ignore-scripts` ao executar `pnpm i` pode permitir a execução de scripts potencialmente maliciosos que fazem parte do processo de instalação dos pacotes.

É um Falso Positivo?

Nem sempre. Em alguns contextos, a execução de scripts durante a instalação de pacotes é intencional e necessária. Por exemplo, scripts de pós-instalação podem configurar adequadamente os pacotes ou realizar tarefas necessárias para o

funcionamento da aplicação. No entanto, há cenários em que isso pode representar um risco de segurança, especialmente se o código for executado em ambientes controlados ou se os pacotes incluírem scripts maliciosos.

No entanto, o alerta pode ser um falso positivo se:

1. **Ambiente Controlado:** O ambiente onde a instalação ocorre é controlado, por exemplo, um ambiente de desenvolvimento ou CI/CD (que é o caso do Monkeytype) com dependências bem definidas, o risco de execução de scripts maliciosos é muito baixo. Nesse caso, o alerta pode ser desconsiderado.

Verificar Scripts Necessários: Analisar os pacotes que estão sendo instalados para entender se há scripts que precisam ser executados.

O comando `RUN npm i -g pnpm` é utilizado em um Dockerfile para instalar o gerenciador de pacotes pnpm globalmente no ambiente de construção da imagem Docker.



Esses actions indicam que a aplicação está integrada a um pipeline de CI/CD (Continuous Integration/Continuous Deployment) que inclui a construção e publicação de imagens Docker e a verificação de falhas em pull requests.

Dado o contexto, o alerta do SonarCloud sobre a ausência da flag `--ignore-scripts` pode ser considerado um falso positivo. Isso porque:

- O ambiente de CI/CD é controlado e bem configurado.
- As dependências e scripts são auditados e provenientes de fontes confiáveis.
- A execução desses scripts durante o processo de build é necessária para a construção e publicação da imagem Docker.

Portanto, a omissão da flag `--ignore-scripts` é segura no contexto da aplicação, e o alerta pode ser desconsiderado como um falso positivo.

Solução Proposta

No entanto, uma possível solução poderia ser:

Adicionar a Flag `--ignore-scripts`:

Se for determinado que nenhum script precisa ser executado (o que não é o caso), a solução mais segura seria modificar o comando de instalação para incluir a flag **`--ignore-scripts`**:

RUN `pnpm i --frozen-lockfile --ignore-scripts`