MINISTÉRIO DA EDUCAÇÃO SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA GOIANO - CÂMPUS CERES

Eric Ferreira Gomes

Segurança em Docker: Ações que Comprometem Imagens Seladas e Avaliação de Ferramentas de Detecção

Ceres - Goiás

2025

Eric Ferreira Gomes

Segurança em Docker: Ações que Comprometem Imagens Seladas e Avaliação de Ferramentas de Detecção

Trabalho de curso apresentado ao curso de Sistemas de Informação do Instituto Federal Goiano – Campus Ceres, como requisito parcial para a obtenção do título de Bacharel em Sistemas de Informação, sob orientação do Prof. Me. Roitier Campos Gonçalves e co-orientação do Prof. Me. Rangel Rigo

Ceres - Goiás
2025

Ficha de identificação da obra elaborada pelo autor, através do Programa de Geração Automática do Sistema Integrado de Bibliotecas do IF Goiano - SIBi

Ferreira Gomes, Eric
G633 Segurança em Docker: Ações que Comprometem Imagens Seladas e Avaliação de Ferramentas de Detecção / Eric Ferreira Gomes. Ceres 2025.

39f. il.

Orientador: Prof. Me. Roitier Campos Gonçalves.
Coorientador: Prof. Me. Rangel Rigo.
Tcc (Bacharel) - Instituto Federal Goiano, curso de 0320203 - Bacharelado em Sistemas de Informação - Ceres (Campus Ceres).
I. Título.

INSTITUTO FEDERAL
Goiano

Repositório Institucional do IF Goiano - RIIF Goiano
Sistema Integrado de Bibliotecas

TERMO DE CIÊNCIA E DE AUTORIZAÇÃO

PARA DISPONIBILIZAR PRODUÇÕES TÉCNICO-CIENTÍFICAS NO REPOSITÓRIO INSTITUCIONAL DO IF GOIANO

Com base no disposto na Lei Federal nº 9.610, de 19 de fevereiro de 1998, AUTORIZO o Instituto Federal de Educação, Ciência e Tecnologia Goiano a disponibilizar gratuitamente o documento em formato digital no Repositório Institucional do IF Goiano (RIIF Goiano), sem ressarcimento de direitos autorais, conforme permissão assinada abaixo, para fins de leitura, download e impressão, a título de divulgação da produção técnico-científica no IF Goiano.

IDENTIFICAÇÃO DA PRODUÇÃO TÉCNICO-CIENTÍFICA

- ☐ Tese (doutorado)
- ☐ Dissertação (mestrado)
- ☐ Monografia (especialização)
- ☑ TCC (graduação)
- ☐ Artigo científico
- ☐ Capítulo de livro
- ☐ Livro
- ☐ Trabalho apresentado em evento
- ☐ Produto técnico e educacional - Tipo:

Nome completo do autor:
Eric Ferreira Gomes

Matrícula:
2022103202030187

Título do trabalho:

Segurança em Docker: Ações que Comprometem Imagens Seladas e Avaliação de Ferramentas de Detecção

RESTRIÇÕES DE ACESSO AO DOCUMENTO

Documento confidencial: ☑ Não ☐ Sim, justifique:

Informe a data que poderá ser disponibilizado no RIIF Goiano: 12 /12 /2025

O documento está sujeito a registro de patente? ☐ Sim ☑ Não

O documento pode vir a ser publicado como livro? ☐ Sim ☑ Não

DECLARAÇÃO DE DISTRIBUIÇÃO NÃO-EXCLUSIVA

O(a) referido(a) autor(a) declara:

• Que o documento é seu trabalho original, detém os direitos autorais da produção técnico-científica e não infringe os direitos de qualquer outra pessoa ou entidade;

• Que obteve autorização de quaisquer materiais inclusos no documento do qual não detém os direitos de autoria, para conceder ao Instituto Federal de Educação, Ciência e Tecnologia Goiano os direitos requeridos e que este material cujos direitos autorais são de terceiros, estão claramente identificados e reconhecidos no texto ou conteúdo do documento entregue;

• Que cumpriu quaisquer obrigações exigidas por contrato ou acordo, caso o documento entregue seja baseado em trabalho financiado ou apoiado por outra instituição que não o Instituto Federal de Educação, Ciência e Tecnologia Goiano.

Ceres — Local
10 /12 /2025 — Data

gov.br Documento assinado digitalmente
ERIC FERREIRA GOMES
Data: 10/12/2025 16:17:48-0300
Verifique em https://validar.iti.gov.br

Assinatura do autor e/ou detentor dos direitos autor

Ciente e de acordo:

gov.br Documento assinado digitalmente
ROITIER CAMPOS GONCALVES
Data: 11/12/2025 10:43:09-0300
Verifique em https://validar.iti.gov.br

Assinatura do(a) orientador(a)

SERVIÇO PÚBLICO FEDERAL
MINISTÉRIO DA EDUCAÇÃO
SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA GOIANO

ATA DE DEFESA DE TRABALHO DE CURSO

Aos seis dias do mês de junho do ano de dois mil e vinte e cinco, reuniu-se a banca examinadora de defesa de Trabalho de Curso da acadêmica **Eric Ferreira Gomes**, do Curso de BACHARELADO EM SISTEMAS DE INFORMAÇÃO, matrícula **2022103202030187**, cujo título é "**Segurança em Docker: Ações que Comprometem Imagens Seladas e Avaliação de Ferramentas de Detecção**". A defesa iniciou-se às 15 horas e 36 minutos, finalizando-se às 20 horas e 15 minutos. A banca examinadora considerou o trabalho APROVADO com média 9,0 no trabalho escrito, média 9,5 no trabalho oral, apresentando assim média aritmética final de 9,25 pontos, estando o estudante APTO para fins de conclusão do Trabalho de Curso. Após atender às considerações da banca e respeitando o prazo disposto em calendário acadêmico, a estudante deverá fazer a submissão da versão corrigida em formato digital (.pdf) no Repositório Institucional do IF Goiano – RIIF, acompanhado do Termo Ciência e Autorização Eletrônico (TCAE), devidamente assinado pelo autor e orientador.

Os integrantes da banca examinadora assinam a presente.

.

(Assinado Eletronicamente)
Roitier Campos Gonçalves
Professor Orientador

(Assinado Eletronicamente)
Vilson Soares de Siqueira
Membro Interno

(Assinado Eletronicamente)
Andréia Mendes Parreira Maciel
Membro Externo

gov.br Documento assinado digitalmente
ANDREIA MENDES PARREIRA MACIEL
Data: 10/12/2025 16:15:29-0300
Verifique em https://validar.iti.gov.br

Documento assinado eletronicamente por:

- **Roitier Campos Goncalves**, **PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 09/12/2025 10:15:31.
- **Vilson Soares de Siqueira**, **PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 09/12/2025 16:27:53.

Este documento foi emitido pelo SUAP em 01/12/2025. Para comprovar sua autenticidade, faça a leitura do QRCode ao lado ou acesse https://suap.ifgoiano.edu.br/autenticar-documento/ e forneça os dados abaixo:

Código Verificador: 768911
Código de Autenticação: 95664d4c30

INSTITUTO FEDERAL GOIANO
Campus Ceres
Rodovia GO-154, Km 03, SN, Zona Rural, CERES / GO, CEP 76300-000
(62) 3307-7100

Agradecimentos

Primeiramente, agradeço a Deus por me conceder saúde, sabedoria e força para superar cada desafio encontrado ao longo desta caminhada. Agradeço à minha mãe e ao meu pai pelo apoio e pelas orientações dadas durante todo o processo de estudo e desenvolvimento deste trabalho. À minha namorada, pelo apoio constante nos momentos de dificuldade, por acreditar em mim e me ajudar a retomar as forças quando mais precisei.

Agradeço aos meus amigos, pelos momentos de descontração que tornaram a trajetória acadêmica mais leve e agradável, mesmo em períodos de tensão. Aos meus orientadores, pelo voto de confiança, pela paciência e pelas valiosas orientações que contribuíram significativamente para a realização deste trabalho.

Agradeço a instituição por proporcionar um ambiente que favoreceu para o desenvolvimento de meus conhecimentos, que foram imprescindíveis para construção desse projeto.

Agradeço também a todos os demais professores, que, com paciência e dedicação, contribuíram para a minha formação e me capacitaram ao máximo para concluir mais uma jornada, adquirindo os conhecimentos necessários para prosseguir em minha carreira profissional.

Resumo

Docker é uma ferramenta de conteinerização que se popularizou entre os desenvolvedores devido sua portabilidade e segurança. A ferramenta utiliza imagens que estão disponíveis no Docker Hub, que podem possuir selos que garantem a segurança e identificam de qual projeto ela pertence. Mesmo que essas imagens possuem segurança base inicial garantida, os usuários utilizam elas ignoram suas limitações de segurança. Essas vulnerabilidades são criadas a partir de má utilização da ferramenta, dessa maneira, tornando elas inseguras e inapropriadas para se adequar às leis como a LGPD e conformidade com a ISO 27001. Sendo assim, existem algumas ferramentas *open source* como Syft e Grype, Trivy junto com a ferramenta Docker Scout, uma ferramenta própria do Docker, que podem escanear os containers em busca dessas vulnerabilidades. Essas ferramentas realizam varreduras nos containers e comparações com bases de dados de vulnerabilidades comuns, em busca das falhas mencionadas, expondo os resultados na tela. Sendo assim, é possível realizar uma comparação de resultados e analisar qual ferramenta seria mais eficiente e eficaz para identificação dessas falhas. Entretanto, é possível observar que os resultados demonstram alguns cenários que criam um conclusão que a utilização das ferramentas em conjunto se demonstra mais eficiente para identificar vulnerabilidades em pacotes e para falhas de boas práticas demonstra a necessidade de atenção para a temática e reforça a necessidade de melhorias que abrangem essas falhas nas ferramentas de escaneamento. Visto que, os scanners podem auxiliar na conformidade com a ISO e comprimento da LGPD. Este trabalho ainda abre a possibilidade de um futuro estudo para identificar qual ferramenta dessas mencionadas é a mais eficiente, já que, este trabalho não possui este objetivo nessa temática.

Palavra-chave: Segurança em Containers. Escaneamento de Vulnerabilidades. Varredura de Segurança. Falhas de Configuração.

Abstract

Docker is a containerization tool that has become popular among developers due to its portability and security. The tool uses images available on Docker Hub, which may have security seals that guarantee security and identify the project to which they belong. Even though these images have guaranteed initial security, users often ignore their security limitations. These vulnerabilities are created from misuse of the tool, making them insecure and unsuitable for compliance with laws such as the LGPD (Brazilian General Data Protection Law) and ISO 27001. Therefore, there are some open-source tools like Syft and Grype, Trivy, along with Docker Scout, a Docker-specific tool, that can scan containers for these vulnerabilities. These tools scan containers and compare them with databases of common vulnerabilities, searching for the mentioned flaws and displaying the results on the screen. Thus, it is possible to compare results and analyze which tool would be most efficient and effective in identifying these flaws. However, it is possible to observe that the results demonstrate some scenarios that lead to the conclusion that using the tools together is more efficient for identifying vulnerabilities in packages and for addressing failures in best practices. This highlights the need for attention to this issue and reinforces the need for improvements that address these flaws in scanning tools. Given that scanners can assist in compliance with ISO standards and the LGPD (Brazilian General Data Protection Law), this work also opens the possibility for a future study to identify which of these mentioned tools is the most efficient, as this work does not have this objective within this context.

Keywords: Container Security. Vulnerability Scanning. Security Scanning. Configuration Flaws.

Listas de Tabela

Listas de Ilustrações

Sumário

1. Introdução

Docker é uma ferramenta de conteinerização que permite a criação, distribuição e execução de aplicações isoladas do seu sistema operacional principal, proporcionando um ambiente de desenvolvimento mais seguro. A tecnologia tem se tornado cada vez mais popular entre os desenvolvedores por oferecer implantações portáteis, estáveis e rápidas, conforme destaca Balabanian et al. (2019). A ferramenta funciona a partir de imagens pré-existentes ou personalizadas pelo usuário, que contêm instruções que possibilitam configurar e personalizar os contêineres e suas aplicações.

Segundo Monteiro (2019), imagens utilizadas para a criação de container estão disponíveis no Docker Hub, um repositório público com diversas imagens de serviços e aplicações que podem ser obtidas por *download*. O autor ainda menciona a possibilidade de *uploads* de imagens criadas pelos usuários, permitindo controle de acesso com repositórios públicos e privados.

Os repositórios podem estar bem documentados, descrevendo as ferramentas configuradas que contém, enquanto outros possuem pouca ou nenhuma documentação. Dentre várias imagens, é possível encontrar as imagens “seladas”, que possuem selos que garantem a segurança base inicial, suas características e projeto de origem. Atualmente, existem três tipos de selo.

Um deles e apresentando como *Docker Official Images*, no qual, a documentação descreve o processo para reforçar a segurança dessa versão: “O sistema de *build* do Docker Official Images é mantido atualizado por meio de *builds* automatizados, auditorias de segurança regulares, colaboração com projetos upstream, testes contínuos e patches de segurança”. (DOCKER INC, 2024).

Além disso, destacam-se as versões *Sponsored OSS* e *Verified Publisher* que a documentação cita uma outra vertente para criar a segurança base inicial. Ela menciona o uso da ferramenta Docker Scout para garantir a segurança base das imagens. A documentação oficial (Docker, 2025) diz que projetos *Sponsored OSS*, conseguem utilizar de forma gratuita o Docker Scout em até 100 repositórios, onde a ferramenta verifica vulnerabilidades em tempo real e automaticamente.

Já projetos com o selo *Verified Publisher*, é mencionado na documentação Docker (Docker, 2025) a utilização do Docker Scout para varreduras automáticas em busca de vulnerabilidades. Esse processo fornece evidências para os

desenvolvedores sobre a confiabilidade, segurança e integridade das imagens antes da publicação.

Sendo assim, para garantir a segurança a documentação (Docker, 2025) aborda de forma breve como a ferramenta funciona, nesse sentido, o Docker Scout realiza escaneamentos em containers em busca de vulnerabilidades, ao identificar é gerado um inventário dos componentes do software (SBOM), que lista todas as dependências que aquela aplicação utiliza. Em sequência, a ferramenta compara o resultado com bancos de dados de vulnerabilidade (CVEs), e expõe as falhas identificadas.

No entanto, de acordo com Balabanian et al, (2019), alguns usuários utilizam as imagens ignorando suas limitações e cuidados necessários, implicando em problemas de segurança que, se seguidas as boas práticas, seriam mitigados. Essas práticas além de afetar a integridade do ambiente Docker também infringe leis como a Lei de Proteção de Dados (LGPD) e normas como a ISO 27001, que exige requisitos para que um ambiente esteja devidamente seguro.

Neste cenário, existem ferramentas além do Docker Scout capazes de escanear containers em busca de vulnerabilidades, na tentativa de manter a integridade dos containers sem comprometimentos. Assim, essas ferramentas poderiam auxiliar para uma prática mais preventiva na execução de um novo ambiente Docker, aprovação dos requisitos da ISO e garantindo conformidade com a LGPD.

2. Revisão de literatura

A segurança em containers Docker possui grande relevância em ambientes de desenvolvimentos, demonstrando importância na discussão deste tema. Os containers são criados a partir de imagens, que podem ser baixadas no repositório oficial do Docker, chamados de Docker Hub, que disponibiliza as imagens em repositórios públicos e privados Monteiro (2019).

Nestes repositórios é possível encontrar imagens com selos de validação de segurança e identificação da origem do projeto. A documentação demonstra 3 (três) selos classificados como: *Docker official image, Verified Publisher*, *Sponsored OSS*.

Para garantir a segurança das imagens com selo *Verified Publisher e Sponsored OSS*, o Docker (Docker, 2025) menciona o uso da ferramenta Docker

Scout, que realiza buscas por vulnerabilidades através de análise de dependências de arquivos e cruzamento dessas informações com bases de dados, assim retornam as vulnerabilidades encontradas.

A documentação ainda menciona que, para imagens classificadas com o selo *Docker official image,* são utilizados *builds* automatizados (construção da imagem), auditorias de segurança regulares, testes contínuos, dentre outros. Todos esses recursos são utilizados para garantir a segurança base inicial dessas imagens. Porém, BALABANIAN et al. (2019), apresenta que os usuários que não utilizam boas práticas acabam quebrando essa camada de segurança inicial criada.

A falta de boas práticas podem estar relacionadas com a má utilização de Dockerfiles no processo de *builds* utilizando as imagens com selos. BALABANIAN et al. (2019) abordam em seu trabalho a ferramenta *Tocker*, um framework capaz de auxiliar no processo de configuração de *firewalls* para segurança em ambientes Docker. A ferramenta é utilizada para evitar configurações inadequadas de firewalls, que permitiriam abertura para intrusos invadirem o sistema e ter acesso à rede, conseguindo navegar em buscas de outros containers conectados nela.

Os autores Lee et al. (2023), apontam para o mesmo problema, ao mencionar que, um container infectado poderá acessar a rede Docker, infectar pacotes que são enviados via tráfego de rede e ainda viabiliza um ataque de movimento lateral.

Tanto Lee et al. quanto BALABANIAN et al. claramente mencionam o ataque de movimento lateral, que busca dispositivos na mesma rede para identificar novas vulnerabilidades para, por exemplo, realizar outros ataques como extração de dados. Sendo assim, ULLAH demonstra a gravidade do ataque quando analisaram o Relatório de Quebras da ITRC no ano de 2017.

Lee et al. ainda menciona sobre a possibilidade do movimento lateral proporcionar um ambiente propício para realização de ataques DDoS, que busca esgotar recursos de hardware, que afetaria a operação normal do host.

Outra falha que aparece expressamente na literatura, consiste no uso de usuários com privilégios elevados (root), dentro de containers. Os autores Ghadi et al. (2009) e Chen et al. (2002) descrevem que o usuário root, identificado pelo UID 0, possui privilégios absolutos no sistema. Nesse sentido, Manu et al (2016) menciona preocupação em caso de invasão de um ambiente Docker que possuem um usuário com esses privilégios. Sendo assim, os autores Eng et al. (2021)

demonstraram que a prática é recorrente, o que fundamenta mais ainda a preocupação de Manu et al.

Outra ocorrência provida de falta de boas práticas, refere-se ao uso indevido de cache de camada. A prática de *layer caching* é descrita pela documentação oficial do Docker (2025) como um recurso de otimização de builds, permitindo o reaproveitamento de camadas. Entretanto, autores como Kaur et al. (2021) destacam que, quando utilizada sem atualização constante dos pacotes, essa técnica pode gerar vulnerabilidades decorrentes de dependências desatualizadas.

Diante dessa necessidade de mitigação de riscos, a utilização de ferramentas de varredura de vulnerabilidades é apresentada pela documentação oficial do Docker e por pesquisas acadêmicas como uma medida preventiva. Contudo, o que a literatura não aborda em profundidade é uma análise comparativa da eficácia desses *scanners* sejam eles *open source* (a exemplo de Syft/Grype e Trivy) ou a solução oficial da Docker (Docker Scout) na identificação direta das vulnerabilidades causadas pelas más práticas de implementação em imagens seladas. Portanto, o presente trabalho se propõe a investigar e comparar a capacidade destas ferramentas de detectar as falhas de configuração e de dependências desatualizadas, decorrentes dos erros de implementação, validando a eficácia dos *scanners* em um cenário experimental. O planejamento e a execução deste experimento serão detalhados na Seção 8 (Metodologia).

3. Selos

O Docker possui várias imagens em seu repositório, algumas delas são de confiança da equipe Docker devido sua origem, seja uma imagem oficial do Docker ou de empresas que disponibilizam suas próprias imagens no Docker Hub. Cada uma delas possui um selo que é usado para identificar e diferenciar as imagens.

No total, existem três selos no Docker Hub, sendo eles, Docker Official Images, Verified Publisher e Sponsored OSS, cada um deles possui sua classificação para uma determinada imagem.

Conforme descrito no site oficial do Docker (Docker, 2025) o selo Docker Official Images representa imagens construídas pelos mantenedores de softwares *(upstream)*, especialistas em segurança e comunidade Docker mais ampla. Embora o Docker prefira que os titulares dos softwares mantenham suas imagens, isso não

é uma restrição. Outros usuários podem contribuir com feedback, código, sugerir mudanças de processo ou até mesmo propor uma nova Imagem Oficial.

Ainda no site oficial (Docker, 2025) o selo Verified Publisher é caracterizado por imagens de editores comerciais que expõem suas aplicações no repositório, onde esses editores são verificados pelo Docker. Imagens com esse selo são diretamente mantidas por essas entidades comerciais.

Imagens com selo Sponsored OSS, identificado no site oficial (Docker, 2025), ajudam os usuários a identificarem projetos *open source* (Código aberto) considerados seguros, confiáveis e ativos. Esses projetos são patrocinados e incentivados pelo Docker.

4. Justificativa

Com o crescimento constante do Docker devido sua portabilidade, flexibilidade e acessibilidade, trouxe vários benefícios para o mundo da tecnologia atraindo diversos desenvolvedores a utilizá-lo. Com diversas imagens expostas no seu repositório (Docker Hub), a experiência do usuário ficou cada vez mais simples na execução de um container já configurado. Algumas imagens são devidamente testadas pela equipe Docker e são identificadas através de selos. Sendo assim, severos testes de segurança são realizados nas imagens para manter um nível de qualidade e confiabilidade.

Ademais, alguns usuários ignoram as recomendações de segurança, o que leva a quebra da arquitetura de segurança que o contêiner possui, isso pode ocorrer devido a falta de conhecimento ou experiência com a ferramenta.

Tais consequências podem ocasionar em vazamento de dados e informações de empresas, colocando em risco vários usuários. Dessa maneira, infringindo a Lei Geral de Proteção de Dados (LGPD) e normas como a ISO 27001, violando os princípios da segurança como confidencialidade e integridade dos dados.

Diante disso, se faz necessário a tratativa do tema, com o objetivo de corrigir falhas por meio de ferramentas capazes de analisar e identificar contêineres que possuem vulnerabilidades. Logo, serão mitigados erros de imagens mal configuradas e erros graves ou sensíveis de usuários menos experientes ou até mesmo usuários mais experientes que sem perceber podem ignorar as recomendações de segurança.

Além disso, o estudo contribui para a área acadêmica ao apresentar uma análise comparativa entre ferramentas de segurança voltadas à detecção de vulnerabilidades em containers, fornecendo resultados para futuras pesquisas e boas práticas na utilização do Docker.

5. Objetivos

5.1 Objetivos Gerais

Analisar quais maneiras os usuários utilizam para quebrarem as camadas de segurança e a arquitetura das imagens 'seladas' do Docker Hub, transformando um container que possui uma camada extra de segurança, em uma ferramenta insegura. Apresentando conceitos que demonstram a relevância e os riscos envolvidos nessas ações.

5.2 Objetivos Específicos

- expor ações dos usuários que compromete a arquitetura de segurança de uma imagem Docker, com foco nas imagens com selo disponibilizada no Docker Hub;
- apresentar ferramentas para identificar possíveis vulnerabilidades em containers Docker;
- Comparar eficiência das ferramentas na identificação de vulnerabilidades e falhas de usuários

6. Erros graves ou sensível de usuários

Ao personalizar uma imagem Docker, se faz necessário cuidados especiais para que os arquivos Docker (Dockerfile e docker-compose) estejam codificados de forma que sua integridade não seja comprometida, no entanto, existe algumas falhas que usuários comentem ao *buildar* uma imagem Docker que foi adaptada para um projeto real ou para estudos.

Este trabalho apresenta o termo “grave ou sensível” com alguns significados:

- expressar um problema que gera impacto grave na sociedade e segurança, como o exemplo apresentado na Seção 6.3;

- expressar ausência de uma instrução em Dockerfiles, que deveria ter uma presença maior;
- uma ação deveria ser mais ausente, porém, foi apresentada como algo recorrente e comum no Dockerfile

6.1 Execução de contêineres com usuários privilegiados

Ghadi et al. (2009), descreve o usuário root como um usuário todo poderoso, pois é capaz de executar tarefas como escrita, leitura e execução de arquivos, sem restrições ou bloqueios. A pesquisa de Chen et al. (2002), complementa mencionando que o superusuário root, carrega consigo o id de identificação (UID) 0, que é reservado para usuário root, dessa maneira, o sistema entende que aquele usuário possui privilégios elevados.

Segundo o site oficial Docker (DOCKER, 2024) o gerenciamento de usuários é um processo crucial para garantir a integridade de seus aplicativos. Se não especificado qual usuário irá manejar o container mencionando a instrução “USER”, por padrão o Docker utiliza um usuário que possui privilégios root. Nesse cenário, MANU et al destacam que “isso levanta a maior preocupação sobre até que ponto um programa malicioso pode usá-lo/explorá-lo” (MANU et al., 2016, p. 7).

Essa preocupação fica mais evidente quando os autores Eng et al. (2021), demonstram em seus estudos com análise de instruções presentes em mais de 9 milhões de Dockerfiles encontrados no WoC (World of Code), entre os anos de 2013 a 2020, observou que o comando “USER” foi utilizado como instrução em apenas 1,13% dos Dockerfiles em suas inúmeras versões que foram analisadas, permanecendo a maioria dos arquivos sem a instrução de usuários, assim, sendo executados com usuário padrão.

Logo, ao não inserir a instrução “USER”, contêineres são executados com usuários com privilégios, o que abre margens para invasores em caso de quebra de segurança

6.2 Riscos do uso indevido do cache de camadas (layer caching)

Na documentação oficial Docker (DOCKER, 2025) diz que o uso de *layer caching* é uma forma mais eficiente de criar contêineres mais rapidamente, visto que, ao executar um Dockerfile, cada comando é compreendido em formas de camadas (*layers*) na execução do container. Sempre que existir uma alteração em uma camada, exemplo no comando COPY é necessário executar um novo *build* para que as novas mudanças sejam visualizadas no container, assim, ignorando o cache. Caso essa camada não seja modificada é validado e aproveitado o cache para construir a nova imagem, dessa maneira, aproveitando o cache da camada antiga usufruindo dos arquivos no estado antigo, assim construindo o container de forma mais rápida.

No entanto, essa prática se torna negativa quando os usuários não atualizam os arquivos que são enviados para dentro de seus contêineres. Nesse contexto, executando um container com pendências ou recursos desatualizados que foram reaproveitados em cache.

Os autores Kaur et al. (2021), mencionam em seus estudos com imagens oficiais encontradas no Docker Hub, mais de 30% delas possuem vulnerabilidades em que comumente ocorre devido ao uso de pacotes desatualizados nas construções das imagens. Isso reforça a importância de manter imagens devidamente atualizadas mesmo que elas tenham sido originalmente publicadas com selos de verificação ou construídas com ferramentas de segurança como o Docker Scout.

Portanto, é essencial que os usuários mantenham seus arquivos, dependências e comandos (RUN, COPY é FROM) atualizados, tanto no Dockerfile quanto no docker compose, com isso, evitando o uso de cache que poderá mascará uma atualização crítica é prejudicar segurança do container.

6.3 Vulnerabilidades na rede bridge padrão do Docker

Containers Docker, descritos na documentação oficial (Docker, 2025), possuem redes de ponte (*bridge*) criados por software no host. Essa rede é utilizada por padrão quando o usuário não especifica uma rede para aquele container, assim, por meio da bridge, permite que contêineres conectados a essa rede possam se comunicar entre si e isolados de outras aplicações que não estão conectadas a essa ponte.

Essa prática pode corroborar para que contêineres infectados conectados a essa mesma rede consigam invadir e infectar novos contêineres. De acordo com estudos de Lee et al. (2023), se um container malicioso estiver dentro de uma ponte, este container poderá falsificar pacotes enviados por outros contêineres e assim permitir que eles possam navegar na rede (*brigde*), sem que sejam detectados por inspeção convencionais. Além disso, o estudo aponta que, caso o container possui autoridade de rede sobre o *host*, como o modo *host,* ele pode comprometer a disponibilidade do sistema, executando um ataque DoS e monitorando o tráfego de todos os contêineres infectados.

Nos estudos de BALABANIAN et al. (2019), apontam para o mesmo problema, até mesmo com o uso de *firewalls* bloqueando o acesso a rede livre, os contêineres possuem comunicação entre si pelo uso da bridge, o que permitiria que um container infectado pudesse alcançar qualquer outro que esteja conectado na mesma rede.

Além disso, ataque de movimento lateral consiste em um atacante invadir um sistema e a partir dele se espalhar pela rede em busca de novos alvos. Uma vez invadido, o atacante consegue acessar qualquer serviço ou aplicação, posteriormente realizar uma extração de dados, roubando informações valiosas de seus clientes. ULLAH et al. (2017) demonstra o Relatório de Quebras da ITRC no ano de 2017 que relatou mais de 1056 incidentes de violação de dados. O mais popular foi a violação de mais de 200 milhões de dados de eleitores americanos. Os dados consistem em endereço, número de telefone, data de nascimento, nome pessoal é detalhes de registro de eleitores. O caso ganhou destaque pela dimensão do vazamento, que ocorreu devido a falhas na configuração do banco de dados.

Nesse contexto, é necessário dar atenção ao tema, visto que a extração de dados viola a lei de proteção de dados (LGPD), pois quebra a integridade e confiabilidade da segurança das imagens.

7. Ferramentas para análise de contêineres

Imagens com Selos encontradas no Docker Hub, podem apresentar segurança extra, desenvolvidas por equipes especialistas ou asseguradas por uma ferramenta nativa (Docker Scout). No entanto, usuários em processo de *build,* com algumas ações podem quebrar as camadas de segurança, tornando então, essas imagens inseguras ou facilitando o acesso para invasores.

Visando seguir um estilo de análise semelhante de análise comparativa e experimental realizado por Kaur et al. (2021), este trabalho utilizará ferramentas de escaneamento de vulnerabilidades, como Syft e Grype, Trivy. Tais ferramentas são apresentadas como soluções *open source* (código aberto) o que justifica a escolha neste trabalho, por apresentar baixo custo sem a necessidade de uso de licenças, transparência na reprodução, facilidade para reaplicação do trabalho e contribuir com sugestões para possíveis melhorias das ferramentas. Assim, os softwares serão utilizados para analisar contêineres com erros.

Outra ferramenta que será utilizada para verificação será o Docker Scout, no qual, será utilizada para fins comparativos e teste de força do escanear no encontro de vulnerabilidades, visto que, essa ferramenta é oficial do Docker. Dessa maneira, buscando verificar a confiabilidade da ferramenta, sendo que, é utilizada para garantir segurança de projetos como Verified Publisher e Sponsored OSS, como mencionado na seção introdutória.

7.1 Syft e Grype

Syft é uma ferramenta *open source* e CLI (*Command-line interface*) mantida pela Anchore. Ou seja, é uma ferramenta de código aberto que funciona em linha de comando e não por interface gráfica. Em seu repositório oficial Github (Syft, 2024) menciona que a ferramenta tem como objetivo gerar listas de materiais de softwares (SBOM). Essas listas são utilizadas para ter maior detalhamento e controle dos pacotes e dependências, quanto a vulnerabilidades e conformidades com licenças de softwares. A ferramenta faz uso da biblioteca da linguagem Golang que realiza análises em imagens de contêineres e sistemas de arquivos para gerar os SBOM.

Grype é uma ferramenta *Open Source* e CLI mantida pela Anchore que segundo sua documentação oficial no Github (Grype, 2024), menciona que a

ferramenta é capaz de escanear imagens de contêineres e sistemas de arquivos. Esse escaneamento funciona a partir de um inventário SBOM previamente gerado por ferramentas como o Syft, podendo estar em formatos padronizados como SPDX ou CycloneDX. Essas vulnerabilidades são identificadas através de bases de dados públicas mantidas localmente como, CVE (Common Vulnerabilities and Exposures), Avisos de segurança do GitHub (GHSAs) dentre outras.

Ambas as duas ferramentas são usadas em conjuntos, no qual, o inventário SBOM é gerado pela ferramenta Syft salvando o arquivo em formatos como json, xml, text e padrões como SPDX ou CycloneDX, e em seguida, o Grype é utilizado para escanear e apontar quais vulnerabilidades foram encontradas. A partir dessa lista de vulnerabilidade de software é gerado um relatório com nome do pacote, versão instalada, nível crítico da vulnerabilidade dentre outros.

7.1.1 Executando o Syft

Para gerar os SBOMs, a documentação (Syft, 2024), recomenda a utilização de comandos como `syft <image> -o <format>`. Esse comando analisa a imagem e com o paramento -o de output, permite a inserção de formatos para retornar o SBOM gerado no formato dos arquivos mencionados na Seção 7.1. O retorno dele será impresso na tela.

Ademais, o uso do comando `syft <image> -o <format>`, pode não ser interessante, se considerar a utilização do grype para verificar o inventário gerado, pois, o comando exibe o retorna no terminal e não salvamento em um arquivo em sua máquina. Para corrigir isso é necessário adicionar o comando `> nome_do_arquivo.sua_extensão`. Esse comando permite salvar o inventário em um arquivo conforme a extensão mencionada, dessa maneira, o exemplo do comando completo ficaria assim: `syft <image> -o <format> > nome_do_arquivo.json`. Esse comando disponibiliza um arquivo em sua máquina pronto para ser escaneado pelo Grype.

7.1.2 Executando o Grype

Para realizar o escaneamento dos SBOMs, a documentação (Grype, 2024) menciona o uso do comando `grype sbom:./sbom.json`. Essa instrução realiza busca no arquivo mencionado para encontrar vulnerabilidades, assim que encontradas, ele cruza com os bancos de dados como CVE, Avisos de segurança do GitHub (GHSAs) e retorna um relatório na tela. O comando utiliza o alvo `sbom:`, o que significa que a ferramenta irá buscar um SBOM válido no arquivo citado.

A ferramenta ainda consegue analisar imagens sem a necessidade de uso de SBOM, a documentação menciona o uso da instrução `grype sua_imagem`, que 5fanalisa a imagem por completo. No entanto, a documentação (Grype, 2024) recomenda a utilização de SBOMs criados pelo Syft para uma varredura mais rápida.

7.2 Trivy

Trivy é descrito em sua documentação no Github (Trivy, 2024), como uma ferramenta poderosa de escaneamento que procura quebras de segurança em contêineres, sistema de arquivos, problemas e configurações incorretas do IaC entre outros. Ele é capaz de identificar vulnerabilidades a partir de imagens de contêineres, repositórios do Git e clusters Kubernetes.

A ferramenta conta com um site que também possui uma documentação. Com ênfase no escaneamento de imagens de contêineres, a documentação (Trivy, s. d) menciona que a ferramenta pode verificar dois cenários:

1. Arquivos dentro de imagens de contêiner
2. Metadados de imagem de contêiner

No primeiro cenário a documentação (Trivy, 2025) diz que a ferramenta examinará os arquivos dentro do contêiner em busca de vulnerabilidades, configurações incorretas, segredos e licenças. Para cada cenário é necessário passar uma instrução diferente na linha de comando.

Conforme descrito na documentação (Trivy, 2025) a busca por vulnerabilidades é feita por padrão com o comando `trivy image [YOUR_IMAGE_NAME]`. Para obter apenas as vulnerabilidades sem o escaneamento completo é necessário adicionar a flag `--scanners vuln`, que realiza um filtro

para buscar vulnerabilidades existentes. Assim o comando ficaria da seguinte forma: `trivy image --scanners vuln [YOUR_IMAGE_NAME].`

De acordo com a documentação (Trivy, 2025), para escanear uma imagem de contêineres em busca de problemas de configuração de infraestrutura usa-se o comando: `trivy image --scanners misconfig [YOUR_IMAGE_NAME]`. Esse comando realiza buscas em arquivos IaC que possuem erros de configuração. Ainda no site oficial (Trivy, s. d), menciona o uso do comando `trivy config [YOUR_IaC_DIRECTORY]`, para realizar varredura em arquivos específicos em buscas de falhas de configurações, o que configura-se como IaC.

A ferramenta ainda é capaz de analisar licenças e segredos (secrets) de softwares. Para encontrar segredos em seu contêiner, a documentação (Trivy, 2025) menciona que a varredura de segredos já vem incluída por padrão no comando `trivy image [YOUR_IMAGE_NAME]`. Já a varredura por licenças não possuem relevância para o presente trabalho, pois não se tratam de falhas de boas práticas e nem de vulnerabilidades que afetam o container.

Para o segundo caso, conforme a documentação oficial (Trivy, 2025), demonstra que a ferramenta é capaz de verificar configurações incorretas e licenças, mas assim como no primeiro caso as licenças foram desconsideradas. Nesse caso, a análise é feita da própria imagem, verificando as camadas de comando do Dockerfile diferentemente do primeiro caso.

Para realizar essa varredura, a documentação (Trivy, 2025) orienta a utilização do comando `trivy image --image-config-scanners misconfig [YOUR_IMAGE_NAME].` O comando irá percorrer cada camada do contêiner em busca de falhas de segurança ocasionadas por uma instrução Docker indevidamente configurada ou ausente.

Após a execução dos comandos a ferramenta cria um SBOM interno e a partir dele o Trivy compara com o banco de dados CVEs encontrando as vulnerabilidades, falhas e erros de execução.

7.3 Docker Scout

Docker Scout é uma ferramenta desenvolvida pelo Docker. De acordo com a documentação oficial (Docker, 2025), destaca o uso do Docker Scout para aprimorar a segurança de imagens Docker. A ferramenta realiza uma varredura em uma imagem criando um SBOM, em seguida, realiza comparações com banco de dados CVEs. Dessa maneira, a ferramenta gera um relatório com as vulnerabilidades encontradas com: nível crítico, código CVE, link para acesso a banco de dados cves, dentre outros dados relevantes.

Conforme descrito pelo Docker (2025) em sua documentação, faz menção da integração com vários ambientes Docker, como: Docker Desktop, Docker Hub, CLI do Docker e o Painel do Docker Scout. Dentre essas, este trabalho executará varredura em imagens hospedadas localmente, onde a execução da ferramenta é feita via linha de comando. Entretanto, a análise também poderá ser feita no Docker Hub, onde a diferença entre as duas maneiras é a forma de configuração, indicando que as imagens hospedadas precisam que seus repositórios estejam com escaneamento via Docker Scout habilitado.

Docker (Docker, 2025) em sua documentação oficial destaca que ao habilitar a ferramenta para um repositório no Docker hub, serão gerados metadados instantâneos das imagens analisadas. Sendo assim, de modo que novas falhas forem adicionadas aos bancos de dados de vulnerabilidades comuns (CVE), serão automaticamente verificadas nas imagens do repositório. Dessa maneira, o Docker Scout consegue atualizar relatórios de segurança das imagens em tempo real, sem a necessidade de reanálise.

Em relação à utilização da ferramenta, o Docker (Docker, 2025) oferece planos de assinatura mensal/anual, que incluem a utilização do Docker Scout em mais repositórios, contudo, na versão gratuita, é possível utilizar a ferramenta através de um único repositório privado. Outra forma de utilizar o Docker Scout é por meio de imagens locais, em que a análise ocorre sem a necessidade de integração com repositórios ou planos de mensalidade.

7.3.1 Como executar

Para executar uma imagem que está hospedada, inicialmente é necessário habilitar o uso da ferramenta em um repositório privado. Essa ativação poderá ser feita através do repositório em que a imagem será hospedada.

Para escanear uma imagem, a documentação (Docker, 2025) menciona o uso do comando: `docker scout cves --only-package express`, no entanto, esse comando está configurado com um filtro para buscar vulnerabilidades relacionado a pacotes express, sendo assim, o comando executado deverá ser: `docker scout cves`, sem filtro e sem a necessidade de especificar uma imagem, pois o Docker Scout executará a última imagem hospedada.

Outro comando que a documentação (Docker, 2025) apresenta é a utilização do comando: `docker scout quickview`. Este comando é utilizado para verificar status de conformidades com políticas padrões, em outras palavras, será feita varreduras analisando boas práticas na construção de imagens.

Para executar varredura em imagens locais, segue a mesma premissa das instruções anteriores, no entanto, é necessário adicionar o nome da imagem local que deseja escanear. Logo, os comandos seriam formatos da seguinte forma: `docker scout cves nome_da_imagem/tag` é para verificação de políticas padrões: `docker scout quickview nome_da_imagem/tag`. As imagens e suas tags podem ser obtidas através do comando `docker ps`.

Com a execução dos comandos, o docker scout retorna na linha de comando um relatório completo com as vulnerabilidades encontradas é na página do repositório também é possível analisar as falhas encontradas pelo painel de vulnerabilidade presente no repositório com Docker Scout habilitado.

8. Metodologia

Este trabalho analisa e identifica o uso imagens Docker seladas que sofreram quebras nas camadas de segurança, devido a implementações incorretas dos contêineres. Essas falhas acontecem por ações de usuários e estão descritas na Seção 6.

Diante do exposto, a execução do presente trabalho será realizada através de experimento feito em imagens Docker. Para isso, serão utilizados padrões de usuários que quebram a camada de seguranças dessas imagens. Nesse cenário, a

execução dos erros de implementação será feita conforme descrito na Seção 6. A cada erro detalhado, será executado na imagem Nginx na versão *Latest* digest sha256:8adbdcb969e2676478ee2c7ad333956f0c8e0e4c5a7463f4611d7a2e7a7ff5dc.

Sua escolha foi feita através de uma análise no Docker Hub. Foram identificadas pouco mais de 20 imagens com mais de 1 Bilhão de *pulls*. Assim, foi necessário um critério para decidir qual a mais popular. No presente momento deste trabalho, o critério adotado foi a quantidade de estrelas recebidas. A imagem do servidor Nginx recebeu mais de 20 mil estrelas de usuários no Docker Hub. Isso mostra que ela tem alta demanda, seja para novas implementações ou simplesmente para ser favoritada e facilitar o acesso em usos futuros.

Para replicar os erros, inicialmente serão criados três Dockerfiles. Cada arquivo será incluído de propósito apenas um dos erros mencionados, assegurando que as demais falhas estejam configurados corretamente.

Assim, ao executar os containers, será realizado um escaneamento usando as ferramentas. Com o intuito de verificar a força dos *scanners* em encontrar essas vulnerabilidades. Os scanners estão mencionadas na Seção 7. Cada Dockerfile copiará dois arquivos para o container (exceto o primeiro Dockerfile que contém apenas o segundo arquivo). Dentre os arquivos, foi utilizado um para configuração do Nginx e outro arquivo html com um site qualquer para realização dos testes. O arquivo de configuração foi realizado a remoção do comando que indica um usuário mencionado por padrão no Nginx. A configuração pode ser analisada na Figura 1.

Figura 1. Arquivo de configuração do Nginx sem menção de usuário.

```
worker_processes auto;
pid /tmp/nginx.pid;
error_log /var/log/nginx/error.log;
include /etc/nginx/modules-enabled/*.conf;

events {
    worker_connections 768;
}

http {

    sendfile on;
    tcp_nopush on;
    types_hash_max_size 2048;


    include /etc/nginx/mime.types;
    default_type application/octet-stream;

    ssl_protocols TLSv1 TLSv1.1 TLSv1.2 TLSv1.3; # Dropping SSLv3, ref: POODLE
    ssl_prefer_server_ciphers on;

    access_log /var/log/nginx/access.log;

    gzip on;

    include /etc/nginx/conf.d/*.conf;
    include /etc/nginx/sites-enabled/*;
}
```

Fonte: Própria

Para executar o Dockerfile com os erros presentes na Seção 6.1 é necessário garantir que não esteja utilizando o cache de camada. Para isso, é necessário utilizar instruções antes da execução do *build,* para criar uma imagem a partir das novas configurações. Nesse cenário, a documentação oficial (Docker, 2025) menciona a utilização do comando `docker image prune -a`. Este comando é utilizado para excluir todas as imagens do host. Caso necessário é possível consultar as imagens locais para exclusão de apenas uma imagem específica.

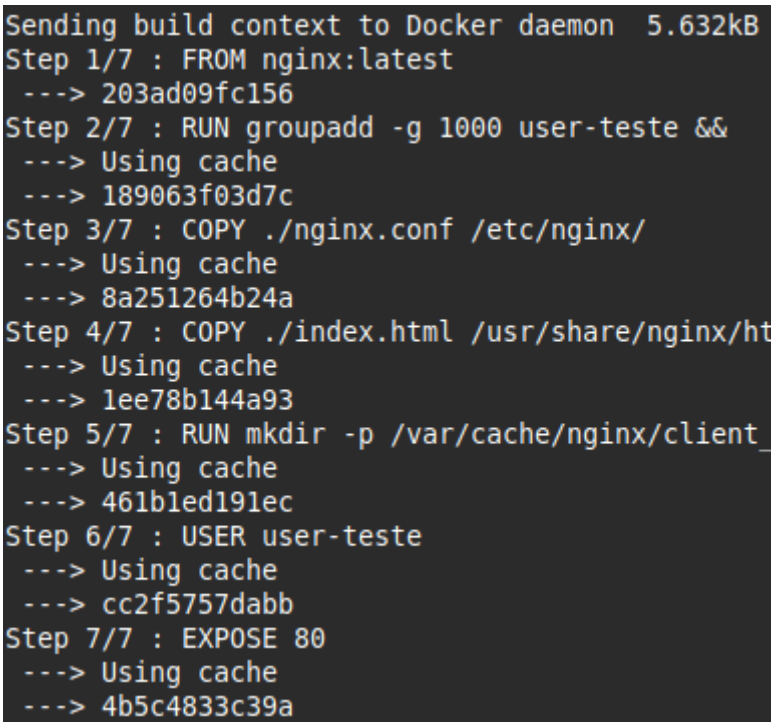
Conforme descrito pelo Docker (Docker, 2025), utiliza-se o comando `docker images`, para observar as imagens presentes na máquina. Neste comando o guia oficial (Docker, 2025) indica que as imagens alocadas no host podem ser utilizadas pelo Docker como cache de camada para acelerar futuros *builds*.

Após identificar a imagem o Docker recomenda (Docker, 2025) a utilização do comando `docker rmi [nome_da_imagem]`, para excluir uma imagem específica. Assim, é possível garantir que o cache será invalidado.

Além disso, antes da execução, deve-se ainda, criar uma rede Docker para o container ser inserido. Para isso, a documentação (Docker, 2025) descreve o uso do comando `docker network create nome_da_rede`, para criar uma rede isolada da rede padrão. Assim, para criar o container com a nova rede a documentação (Docker, 2025) indica a utilização da flag `--network`, usada para especificar uma rede ao container, sendo utilizado no comando run, isolando o ambiente da rede padrão.

Para o erro apresentado na Seção 6.2, é necessário a utilização do cache em camada antes da realização da varredura. Para isso, a imagem precisa ser construída uma vez a partir do Dockerfile, em sequência deve-se reconstruir outra imagem usando o mesmo arquivo. Dessa maneira, nos logs no terminal será apresentando uma mensagem de "*Using Cache*", demonstrando a validação do cache. A Figura 2, demonstra o resultado no terminal com o cache sendo validado e utilizado.

Figura 2. Demonstração do log com o Docker executando uma nova imagem utilizando cache.



Fonte: Própria

Ademais, é indispensável a indicação de um usuário para manusear o container, porém, este passo está codificado como instrução dentro do Dockerfile que será apresentado na Seção 9.

No caso do erro da Seção 6.3, segue o mesmo processo de invalidação do cache, descrito na execução do erro da Seção 6.1 e também é necessário ser criado o container com o usuário não root especificado, conforme mencionado anteriormente. Dessa maneira, os três Dockerfiles estarão devidamente configurados para executar um erro por vez isoladamente.

Após a criação do Dockerfiles e execução do mesmo, será realizada uma varredura, conforme os comandos mencionados na Seção 7. Os resultados dos escaneamentos são apresentados no próprio terminal, porém para facilitar a análise, cada resultado foi transferido para um arquivo.txt. Os testes foram realizados utilizando o Docker Scout, na versão v1.18.3 o GoLang, na versão go1.24.6 - linux/amd64. Com a ferramenta Trivy foi utilizado na versão 0.66.0 e os softwares da Anchore, foram utilizados na versão 0.99.1 para Grype e a versão 1.32.0 para Syft e por fim o Docker na versão 27.5.1.

Logo após a realização dos testes, os resultados serão apresentados através de tabelas na seção de resultados. Será possível analisar quais códigos CVEs foram detectados por cada ferramenta e verificar se os erros causados pelos usuários foram encontrados. Também será apresentando comparações entre as ferramentas, considerando a análise de severidades como críticas e altas relacionados a pacotes encontrados simultaneamente entre as ferramentas e a quantidade de vulnerabilidades de cada nível de severidade. Outro ponto que poderá ser analisado é a possibilidade de uma ferramenta classificar um pacote como de severidade mais alta, enquanto outra classifica um nível mais baixo para o mesmo pacote.

Dessa maneira, essa metodologia permite simular falhas em containers e realizar uma análise comparativa entre as ferramentas de escaneamento. Dessa forma, é possível entender como cada scanner identifica vulnerabilidades e avaliar se são capazes de detectar os erros de implementação. Os resultados desse processo serão mostrados e discutidos na próxima seção.

9. Resultados

Os resultados desta pesquisa permitem compreender a importância da segurança em ambientes Docker e analisar como as ferramentas de escaneamento auxiliam nesse processo de fortalecimento de containers.

Nesse cenário, foi possível analisar resultados que apontam uma diferenciação de classificação entre as ferramentas na quantidade de severidade encontrada. Isso ocorre devido a diversidade de bases de dados CVEs que cada ferramenta utiliza. Os resultados estão disponibilizados na Tabela 1.

Também foi possível observar o retorno da mesma quantidade de severidade encontrada pelas ferramentas em todas as imagens escaneadas. O que já era esperado, visto que, ambos os Dockerfiles utilizaram a mesma imagem base. A tabela apresenta duas colunas denominadas como “desconhecido” e “insignificante”, que caracterizam termos usados pelas ferramentas para classificar uma severidade que não seja alta, média ou baixa. Dessa maneira, o uso do símbolo “-” representa que, as ferramentas não utilizam o termo para referência esse tipo de severidade não identificada.

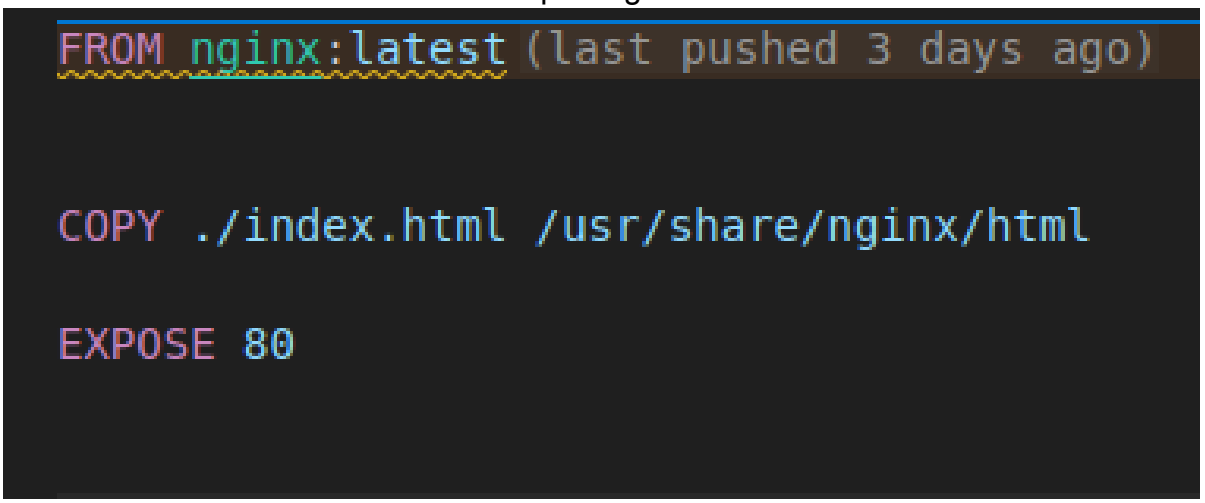
Tabela 1. Quantidade de severidade encontradas por cada scanner em todos os containers - Brasil, 2025

Ferramentas	Crítico	Alto	Médio	Baixo	Desconhecido	Insignificante	Total
Syft & Grype	0	5	7	7	-	72	91
Trivy	0	2	10	84	0	-	96
Docker Scout	0	2	1	46	0	-	49

Fonte: Própria

Para execução dos testes, pode-se visualizar a configuração adotada para o primeiro Dockerfile, demonstrado na Figura 3. A Figura apresenta configurações básicas para execução do container, contendo exposição da porta 80 para acessá-lo é cópia de um arquivo html com um site para ilustração.

Figura 3. Dockerfile com configurações para execução de erro com usuário padrão com privilégio root



Fonte: Própria

Para execução do erro da Seção 6.2 e 6.3, os Dockerfile utilizavam a mesma configuração do anterior, no entanto, houve um acréscimo de um usuário para manipular o container. No arquivo possui comandos para criar o usuário, instruções para dar as devidas permissões para o usuário é utilização da instrução USER para indicar que este usuário irá comandar o container. Estas configurações estão visíveis na Figura 4.

Figura 4. Configuração Dockerfile para execução do erro da Seção 6.2 e 6.3.

```
FROM nginx:latest

RUN groupadd -g 1000 user-teste && \
    useradd -u 1000 -g user-teste user-teste

COPY ./nginx.conf /etc/nginx/
COPY ./index.html /usr/share/nginx/html

RUN mkdir -p /var/cache/nginx/client_temp \
    && chown -R user-teste:user-teste /var/cache/nginx \
    && chown -R user-teste:user-teste /usr/share/nginx/html \
    && chown -R user-teste:user-teste /etc/nginx
USER user-teste

EXPOSE 80
```

Fonte: Própria

Nesse cenário, ao executar o container, se inicia a fase de verificação para tentativa de encontrar os erros. Para isso, foram executados comandos para encontrar falhas explicitamente de configuração em arquivos Dockerfile e containers. Estes encontrados na Seção 7. Vale ressaltar que a ferramenta Grype não possui instruções capazes de realizar esse tipo de varredura, sendo assim, não computado seus resultados. A Tabela 2 demonstra o resultado encontrado, onde apresentam os valores como “sim” para erros que as ferramentas encontraram e “não” para não encontrados.

Tabela 2. Resultados da varredura em busca de erros de configuração em arquivos Docker e containers - Brasil, 2025.

Ferramentas	Privilégio root	Uso de Layer Cache	Uso da rede padrão
Syft & Grype	-	-	-
Trivy	Sim	Não	Não
Docker Scout	Sim	Não	Não

Fonte: Própria

No entanto, ao considerar algumas vulnerabilidades encontradas, é possível classificar o uso de Layer Cache como “sim”, na Tabela 2, visto que, algumas vulnerabilidades podem ser julgadas como pacotes desatualizados, onde a ferramenta apresenta a versão do pacote que resolve a falha encontrada. Essas versões são apresentadas na coluna *Fixed Version* e *FIXED IN,* encontradas no Trivy e no Grype, respectivamente ou na menção ao termo *Fixed version*, encontrado no Docker Scout.

Nesse sentido, o erro de uso de Cache indevido, poderá ser identificado de forma indireta pela ferramenta, visto que, a análise dos softwares não menciona o erro diretamente causado por uso de *Layer Cache* é sim pela identificação dos pacotes desatualizados.

Outro resultado que foi possível analisar em relação a comparação entre as ferramentas, são os resultados de severidade como crítica e alta. Como não foram obtidas severidades de nível crítico, foram consideradas apenas as de nível Alto. Nesse sentido, foi possível analisar a capacidade das ferramentas de classificar os mesmos pacotes com a mesma severidade. Assim, a Tabela 3 demonstra pacotes que foram encontrados pela ferramenta Syft/Grype.

Tabela 3. Análise de resultado da ferramenta Syft e Grype na detecção das vulnerabilidades com severidade alta - Brasil, 2025.

Pacotes	CVEs	Identificação	Severidade
ibexpat1 2.7.1-2	CVE-2025-59375	Sim	Alta
libtiff6 4.7.0-3	CVE-2025-9900	Sim	Alta
curl 8.14.1-2	CVE-2025-9086	Sim	Alta
libcurl4t64 8.14.1-2	CVE-2025-9086	Sim	Alta
libxslt1.1 1.1.35-1.2+deb13u2	CVE-2025-7425	Sim	Alta

Fonte: Própria

Os resultados encontrados pela ferramenta Syft/Grype demonstraram a identificação de todas as vulnerabilidades classificadas como alta entre os pacotes vulneráveis que foram identificados simultaneamente pelas ferramentas. Na Tabela 4 é possível verificar resultados da ferramenta Trivy.

Tabela 4. Análise de resultado da ferramenta Trivy na detecção das vulnerabilidades com severidade alta - Brasil, 2025.

Pacotes	CVEs	Identificação	Severidade
ibexpat1 2.7.1-2	CVE-2025-59375	Sim	Médio
libtiff6 4.7.0-3	CVE-2025-9900	Sim	Alta
curl 8.14.1-2	CVE-2025-9086	Sim	Médio
libcurl4t64 8.14.1-2	CVE-2025-9086	Sim	Médio
libxslt1.1 1.1.35-1.2+deb13u2	CVE-2025-7425	Sim	Alta

Fonte: Própria

Nesta Tabela demonstra que a ferramenta encontrou todas as vulnerabilidades que foram identificadas e classificadas com severidade Alta. No entanto, três delas receberam classificação média pela ferramenta, demonstrando o impacto da diversidade entre as bases de dados utilizadas pelas ferramentas. Por fim, a Tabela 5 demonstra os resultados da ferramenta Docker Scout.

Tabela 5. Análise de resultado da ferramenta Docker Scout na detecção das vulnerabilidades com severidade alta - Brasil, 2025.

Pacotes	CVEs	Identificação	Severidade
ibexpat1 2.7.1-2	CVE-2025-59375	Sim	Alta
libtiff6 4.7.0-3	CVE-2025-9900	Sim	Alta
curl 8.14.1-2	CVE-2025-9086	Não	-
libcurl4t64 8.14.1-2	CVE-2025-9086	Não	-
libxslt1.1 1.1.35-1.2+deb13u2	CVE-2025-7425	Não	-

Fonte: Própria

Os Resultados da Tabela 5 demonstraram que a ferramenta Docker Scout foi capaz de identificar duas vulnerabilidade classificadas como alto em comum pelas ferramentas, o restante marcado pelo símbolo “-” significa a ausência da identificação desses pacotes.

10. Análise e Discussões

A metodologia aplicada neste trabalho expõe resultados relevantes para garantir segurança de containers, demonstrando eficiência de cada ferramenta para identificar possíveis vulnerabilidades que comprometam a integridade de ambientes Docker. Sendo assim, é possível analisar dois cenários

No primeiro cenário os resultados demonstraram que as ferramentas Trivy e Docker Scout classificaram todas as vulnerabilidades que eles identificam, no qual, a ferramenta Trivy foi a que mais identificou pacotes vulneráveis. Já as ferramentas Syft e Grype, classificaram 72 vulnerabilidades como “Insignificante”, o que pode-se considerar um resultado que permite lacunas na análise para resolução das falhas, visto que, não demonstra de forma clara um ponto de partida para a resolução.

Além disso, foi realizada uma análise comparativa dos pacotes classificados como de alta severidade, verificando quais deles foram identificados simultaneamente por todas as ferramentas. Nesse cenário, as ferramentas Syft e Grype, identificaram todos os pacotes vulneráveis classificados com alta severidade. Já a ferramenta Trivy apresentou 2 (dois) pacotes com severidade alta e o restante classificado com severidade média. Já a ferramenta Docker Scout apresentou uma

classificação diferente, em que, foram identificados dois pacotes com severidade alta é os demais não foram apresentados. Essa reação do Docker Scout é justificado pois a ferramenta é projetada para ignorar certas vulnerabilidades, descartando falsos positivos. Nesse sentido, a documentação Docker, menciona que a presença de uma vulnerabilidade não significa que essa falha seja explorada no ambiente Docker. Logo, a ferramenta descarta vulnerabilidades que não serão exploradas nos containers.

Isso é evidenciado na documentação oficial (Docker, 2025) que menciona que, ao identificar um CVE marcado como não aplicável ou falso positivo, o Docker Scout o remove automaticamente dos resultados finais da análise. O que poderia acusar as outras ferramentas de considerarem falsos positivos. O que justifica também, a ferramenta ter a menor taxa de identificação de vulnerabilidades em relação às outras.

Já no segundo cenário, na análise de falhas de boas práticas em Dockerfiles, é possível observar mais dois cenários distintos. No primeiro caso de maneira direta, os resultados obtidos das ferramentas como Trivy e Docker Scout demonstram a identificação de ao menos uma falha das três aplicadas neste estudo. Já o Syft e Grype, não realizam verificação de boas práticas em Dockerfiles, portanto não identificam falhas de configuração de containers. Este resultado expõe pontos de melhorias ou de inclusão de vulnerabilidades ocorridas por más práticas, justificando a gravidade que estas falhas podem causar em ambientes Docker.

O segundo caso demonstra a identificação de duas falhas. As ferramentas são capazes de identificar pacotes desatualizados, isto demonstra uma forma indireta de detecção do erro de uso indevido de cache de camada. Sendo assim,a detecção resultou em duas falhas encontradas pelo Docker Scout e Trivy é uma falha nas ferramentas Syft e Grype.

Apesar do segundo caso apresentar resultados “satisfatórios”, ainda é necessário dar importância para erros formulados por más práticas em ambientes Docker. Visto que as ferramentas não identificaram de forma direta as falhas, o que permite que erros de usuários possam passar despercebidos pelos resultados da varredura. Sendo assim, é necessário a abordagem dessas identificação de falhas em um processo mais assertivas é exploratório pelo mantenedores de softwares, para contribuição de ambientes de desenvolvimento com Docker mais seguros.

11. Conclusão

Este trabalho teve como objetivo analisar o comportamento de diferentes ferramentas de varredura de vulnerabilidades e cenários de más práticas aplicadas a ambientes Docker e ainda propor uma comparação entre as ferramentas.

Nesse cenário, foram realizados testes utilizando ferramentas como Syft/Grype, Trivy e Docker Scout, nas quais cada uma apresenta comportamentos diferentes. Syft e Grype apresentaram resultados que em verificação de severidade foram classificadas 72 como “insignificantes”, o que gera incertezas na tomada de decisão sobre a prioridade de correção. Já a ferramenta Trivy e Docker Scout demonstraram maior profundidade na detecção de falhas, em que, todas vulnerabilidades identificadas foram classificadas. Entretanto, o Docker Scout retornou um número menor de falhas, pois a ferramenta ignora vulnerabilidades que não apresentam riscos para o container, ou seja, o software ignora falsos positivos.

Sendo assim, apesar do Trivy apresentar resultados mais completos, na classificação de severidade alta em pacotes que foram identificados simultaneamente pelas ferramentas, o *scanner* apresentou três pacotes como severidade média, enquanto o Syft e Grype, classificou os mesmos como severidade alta. O Docker Scout considerou apenas dois com severidade alta e os demais acabaram ignorando e julgando como falso positivo.

Essas comparações também levam a uma incerteza de qual ferramenta possui resultados mais assertivos nas varreduras, visto que geram vários cenários divergentes o que dificulta uma avaliação mais assertiva.

Já no cenário de falha de configuração na estrutura de Dockerfiles, é possível verificar que, duas ferramentas, Trivy e Docker Scout identificaram containers com uso de usuários com privilégios root. As ferramentas Syft e Grype não realizaram a busca dessas vulnerabilidades que ocorrem devido a más práticas. Já no uso de *Layer Caching*, pode-se observar a identificação de forma indireta, visto que todas as ferramentas identificam pacotes desatualizados e a identificação de uso da mesma rede ou da rede padrão não foi identificado por nenhuma ferramenta.

Estes resultados demonstram que até mesmo ferramentas dedicadas a identificar vulnerabilidades de pacotes, não conseguem abranger todos os cenários vulneráveis, deixando lacunas na avaliação. Isso leva à conclusão que a utilização de todas elas demonstra ser um cenário mais eficiente para executar um ambiente

Docker mais seguro, na tentativa de se adequar à ISO e estar em conformidade com LGPD.

Já no cenário de varredura de erros de configuração no ambiente Docker indicam que é necessária uma maior atenção para o tema, pois nenhuma ferramenta foi capaz de identificar todos os casos, visto que, este trabalho comprovou seu impacto eficiente em containers mal configurados.

Além disso, não foi possível identificar qual ferramenta possui maior eficiência, sendo assim, abre espaço para pesquisas mais completas e dedicadas a concluir qual ferramenta possui maior assertividade na busca de vulnerabilidades em container Docker.

Referências bibliográficas

BALABANIAN, Felipe; HENRIQUES, Marco. **Tocker: framework para a segurança de containers Docker**. In: WORKSHOP DE TRABALHOS DE INICIAÇÃO CIENTÍFICA E DE GRADUAÇÃO - SIMPÓSIO BRASILEIRO DE SEGURANÇA DA INFORMAÇÃO E DE SISTEMAS COMPUTACIONAIS (SBSEG), 19. , 2019, São Paulo. Anais [...]. Porto Alegre: Sociedade Brasileira de Computação, 2019. p. 145-154. DOI: https://doi.org/10.5753/sbseg_estendido.2019.14016.Acesso em: 25 Mar. 2025.

Bhupinder Kaur, Mathieu Dugré, Aiman Hanna, Tristan Glatard, **An analysis of security vulnerabilities in container images for scientific data analysis**, GigaScience, Volume 10, Issue 6, June 2021, giab025, https://doi.org/10.1f093/gigascience/giab025. Acesso em: 17 Jul. 2025

CHEN, Hao; WAGNER, David; DEAN, Drew. *Setuid Demystified.* In: **USENIX SECURITY SYMPOSIUM (11th: 2002: San Francisco, CA, USA).** *Proceedings*. Berkeley, CA: USENIX Association, 2002. Disponível em: https://www.usenix.org/conference/11th-usenix-security-symposium/setuid-demystified. Acesso em: 27 out. 2025.

DOCKER INC. **Bridge network driver**,2025. Disponível em: https://docs.docker.com/engine/network/drivers/bridge/. Acesso em: 23 Jul. 2025.

DOCKER INC. **Docker build cache**, 2025. Disponível em: https://docs.docker.com/build/cache/. Acesso em: 17 Jul. 2025

DOCKER INC. **Docker image prune**.2025. Disponível em: https://docs.docker.com/reference/cli/docker/image/prune/. Acesso em 3 de Out. 2025.

DOCKER INC. **docker network create**. 2025.. Disponível em: https://docs.docker.com/reference/cli/docker/network/create/. Acesso em 3 de Out. 2025.

DOCKER INC. **Docker Official Images**. 2025. Disponível em: https://docs.docker.com/docker-hub/repos/manage/trusted-content/official-images/. Acesso em: 2 Abr. 2025.

DOCKER INC. **Docker Scout**. 2025. Disponível em: https://docs.docker.com/scout/. Acesso em: 14 abr. 2025.

DOCKER INC. **Docker Scout image analysis.** 2025**.** Disponível em: https://docs.docker.com/scout/explore/analysis/. Acesso em 26 Set. 2025.

DOCKER INC. **Docker Scout quickstart.** 2025. Disponível em: https://docs.docker.com/scout/quickstart/#step-6-improve-compliance. Acesso em 26 Set. 2025.

DOCKER INC. **Docker-Sponsored Open Source Program**. 2025. Disponível em: https://docs.docker.com/docker-hub/repos/manage/trusted-content/dsos-program/. Acesso em: 3 Abr. 2025.

DOCKER INC. **Docker Verified Publisher Program**. 2025. Disponível em: https://docs.docker.com/docker-hub/repos/manage/trusted-content/dvp-program/. Acesso em: 3 Abr. 2025.

DOCKER INC. **Find Your Perfect Docker Plan**. 2025. Disponível em: https://www.docker.com/pricing/. Acesso em 26 Set. 2025.

DOCKER INC. **From misconceptions to mastery: enhancing security and transparency with Docker Official Images**. 2024. Disponível em: https://www.docker.com/blog/enhancing-security-and-transparency-with-docker-official-images/. Acesso em: 26 mar. 2025.

DOCKER INC. **Gerenciar exceções de vulnerabilidade**. 2025.. Disponível em: https://docs.docker.com/scout/explore/exceptions/. Acesso em 17 de Out. 2025.

DOCKER INC. **Imagem docker ls**. 2025.. Disponível em: https://docs.docker.com/reference/cli/docker/image/ls/. Acesso em 3 de Out. 2025.

DOCKER INC. **Imagem docker rm**. 2025. Disponível em: https://docs.docker.com/reference/cli/docker/image/rm/. Acesso em 3 de Out. 2025.

DOCKER INC. **Understanding the Docker USER Instruction**, 2024. Disponível em: https://www.docker.com/blog/understanding-the-docker-user-instruction/. Acesso em 14 Mai. 2025.

Ghadi, Abderrahim; Mammass, D.; Mignotte, Maurice; Sartout, Alain. (2009). **Hierarchical Role Graph Model for UNIX Access Control.** ***International Journal of Future Generation Communication and Networking***, 2, p. 3. Disponível em: https://www.researchgate.net/publication/228827984_Hierarchical_Role_Graph_Model_for_UNIX_Access_Control. Acesso em: 27 out. 2025.

Github. **anchore-engine**, 2023. Disponível em: https://github.com/anchore/anchore-engine?tab=readme-ov-file. Acesso em 18 Set. 2025.

Github. **grype**, 2024. Disponível em: https://github.com/anchore/syft. Acesso em 16 Set. 2025.

Github. **syft**, 2024. Disponível em: https://github.com/anchore/syft. Acesso em 16 Set. 2025.

K. Eng and A. Hindle, **"Revisiting Dockerfiles in Open Source Software Over Time**," 2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR), Madrid, Spain, 2021, pp. 449-459, doi: 10.1109/MSR52588.2021.00057. Acesso em 22 Ago. 2025.

LEE, H.; KWON, S.; LEE, J.-H. ***Experimental Analysis of Security Attacks for Docker Container Communications*. Electronics**, v. 12, n. 4, p. 1–19, 2023. Disponível em: https://doi.org/10.3390/electronics12040940. Acesso em: 23 jul. 2025.

MANU, A. R.; PATEL, J. K.; AKHTAR, S.; AGRAWAL, V. K.; MURTHY, K. N. B. S. **Docker container security via heuristics-based multilateral security – conceptual and pragmatic study.** In: 2016 International Conference on Circuit, Power and Computing Technologies (ICCPCT). IEEE, 2016. p. 1–14. DOI: https://doi.org/10.1109/ICCPCT.2016.7530217. Acesso em: 20 Mai. 2025.

MONTEIRO, Luciano Aguiar; ALMEIDA, Washington Henrique Carvalho. **Cluster de Alta Disponibilidade com Docker Swarm**. In: ESCOLA REGIONAL DE INFORMÁTICA DO PIAUÍ, 3., 2017, Teresina. *Anais...* Teresina: UFPI, 2017. p. 279-295. Disponível em: https://www.researchgate.net/publication/332510704. Acesso em: 8 abr. 2025.

Trivy. **Container Image**. 2025. Disponível em: https://trivy.dev/docs/v0.66/target/container_image/. Acesso em 18 Set. 2025.

Trivy. **Misconfiguration Scanning**. 2025. Disponível em: https://trivy.dev/docs/v0.66/scanner/misconfiguration/. Acesso em 18 Set. 2025.

Ullah, Faheem & Edwards, Matthew & Ramdhany, Rajiv & Chitchyan, Ruzanna & Ali Babar, Muhammad & Rashid, Awais. (2017). **Data Exfiltration: A Review of External Attack Vectors and Countermeasures**. Journal of Network and Computer Applications. 101. https://doi.org/10.1016/j.jnca.2017.10.016. Acesso em 31 Ago. 2025.