Segurança em Docker Multiusuário: Avaliação de Riscos de Privilege Escalation e Data Tampering

Lucas Da Costa Miranda[1], **Charles Tim Batista Garrocho**[1]

[1] Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais (IFMG)
Ouro Branco, Minas Gerais, Brasil

lucasdacostamiranda812@gmail.com, charles.garrocho@ifmg.edu.br

Abstract. *The use of Docker in multi-user environments has grown significantly due to its efficiency and portability. However, insecure configurations, such as allowing access to the* `docker` *group, introduce critical vulnerabilities that allow Privilege Escalation and Data Tampering attacks. This work investigates these risks through a case study at IFMG, where the inappropriate use of* `docker.sock` *on shared workstations was identified. To quantify this exposure, the Integrity Breach Exposure Index (IEVI) is proposed, a metric based on the DREAD framework that consolidates risk dimensions into a quantitative score. Additionally, the IntegrityGuard tool is presented, responsible for automating vulnerability detection and the execution of controlled tests. The experimental evaluation considers vulnerable and mitigated scenarios, analyzing the impact of different configurations. The results demonstrate that adopting the Docker Rootless mode drastically reduces the attack surface, decreasing the IEVI from 100 to 32 and limiting the exploitation of critical attacks. These findings reinforce the importance of the principle of least privilege and highlight the effectiveness of approaches aligned with the Zero Trust model in shared infrastructures.*

Resumo. *O uso do Docker em ambientes multiusuário cresceu significativamente devido à sua eficiência e portabilidade. Contudo, configurações inseguras, como a permissão de acesso ao grupo* `docker`*, introduzem vulnerabilidades críticas que permitem ataques de Escalonamento de Privilégios e Violação de Integridade (Data Tampering). Este trabalho investiga esses riscos por meio de um estudo de caso no IFMG, onde foi identificado o uso inadequado do* `docker.sock` *em estações compartilhadas. Para quantificar essa exposição, propõe-se o Índice de Exposição à Violação de Integridade (IEVI), uma métrica baseada no framework DREAD que consolida dimensões de risco em um escore quantitativo. Complementarmente, apresenta-se a ferramenta IntegrityGuard, responsável por automatizar a detecção de vulnerabilidades e a execução de testes controlados. A avaliação experimental considera cenários vulneráveis e mitigados, analisando o impacto de diferentes configurações. Os resultados demonstram que a adoção do modo Docker Rootless reduz drasticamente a superfície de ataque, diminuindo o IEVI de 100 para 32 e limitando a exploração de ataques críticos. Esses achados reforçam a importância do princípio do menor privilégio e evidenciam a efetividade de abordagens alinhadas ao modelo Zero Trust em infraestruturas compartilhadas.*

1. Introdução

A conteinerização de aplicações, notadamente através do *Docker*, revolucionou o desenvolvimento e a implantação de software, oferecendo vantagens significativas sobre máquinas virtuais tradicionais, como maior eficiência no uso de recursos e portabilidade (Sultan et al. 2019). A ampla adoção do *Docker* em ambientes de desenvolvimento e produção é uma tendência consolidada; contudo, sua utilização em hosts multiusuário, como laboratórios computacionais, centros de pesquisa e máquinas compartilhadas em equipes corporativas, impõe desafios de segurança específicos. Isso ocorre porque *containers* compartilham o mesmo *kernel* do sistema hospedeiro, ampliando a superfície de ataque e expondo o sistema a diferentes vetores de exploração (Wong et al. 2023).

A motivação para este estudo surge de uma análise prática realizada nos laboratórios do Instituto Federal de Minas Gerais Campi Ouro Branco (IFMG), onde foi identificada uma vulnerabilidade sistêmica em mais de 120 máquinas de uso compartilhado por alunos e professores. Essas estações de trabalho apresentavam uma falha crítica de governança: a inclusão de usuários comuns no grupo secundário `docker`. Essa configuração permite o acesso direto ao *socket* `/var/run/docker.sock`, o que equivale, na prática, à concessão de privilégios de *root* ao usuário local (Rajyashree et al. 2024). Em hosts compartilhados, a exploração dessa configuração insegura pode levar a duas classes de impacto diretamente relacionadas ao *framework* STRIDE (Wong et al. 2023).

Um cenário prático que ilustra ambas as ameaças é a montagem do sistema de arquivos do *host* dentro de um *container* e a modificação do arquivo `/etc/passwd`. Por exemplo:

Listagem 1. Comando para criação de usuário administrativo via vulnerabilidade *docker*.

```
docker run -v /etc:/mnt/host/etc -it --rm alpine \
  sh -c "echo 'hacker:x:0:0:Hacker:/root:/bin/sh' >> /mnt/host/etc/
     passwd"
```

Esse simples comando demonstra que, além de corromper dados (*Data Tampering*), um atacante pode criar um usuário com *User ID* (UID) 0, obtendo assim controle administrativo do *host*. O tempo de exploração *Time-to-Exploit* (TTE) para essa classe de ataque é baixo (segundos), o que a torna de alto risco em ambientes multiusuário.

A problemática central reside no fato de que, embora a conteinerização ofereça isolamento lógico, ela não garante o isolamento físico robusto característico das máquinas virtuais, uma vez que todos os *containers* compartilham o mesmo *kernel* do sistema hospedeiro (Sultan et al. 2019; Wong et al. 2023). Em cenários de uso compartilhado, como laboratórios acadêmicos e centros de pesquisa, a segurança é frequentemente negligenciada em favor da conveniência operacional. A inclusão de usuários no grupo `docker` para simplificar a execução de tarefas acadêmicas acaba por anular as barreiras de proteção do sistema operacional, transformando uma ferramenta de produtividade em um vetor crítico de ataque. Essa configuração insegura permite que um usuário comum interaja diretamente com o *socket* do *Docker*, o que equivale, na prática, à concessão de privilégios de *root* ao usuário local (Rajyashree et al. 2024). A gravidade dessa falha é acentuada pelo baixo tempo de exploração (*Time-to-Exploit*), permitindo que um atacante comprometa a

integridade de arquivos sensíveis do *host* em poucos segundos, justificando a necessidade urgente de ferramentas de diagnóstico automatizado e métricas de risco quantitativas.

Diante desse cenário, o objetivo deste trabalho é analisar, quantificar e mitigar os riscos associados tanto à Violação de Integridade quanto à Escalada de Privilégios decorrentes do acesso indevido ao `docker.sock`. As principais contribuições deste artigo são:

1. A formulação do Índice de Exposição à Violação de Integridade (IEVI), uma métrica quantitativa derivada do DREAD, estendida para refletir impactos de *tampering* e de escalada de privilégio;
2. O desenvolvimento da ferramenta *IntegrityGuard*, que automatiza a detecção da exposição, executa testes controlados e recomenda contramedidas para reduzir o IEVI;
3. A validação empírica dos riscos e das mitigações propostas, demonstrando a eficácia do modo *Docker Rootless* na redução da superfície de ataque em ambientes reais.

O restante deste trabalho está organizado da seguinte forma: a Seção 2 aborda o referencial teórico sobre gestão de privilégios e a plataforma *Docker*, além de discutir os trabalhos relacionados. A Seção 3 descreve a metodologia experimental, os procedimentos de teste, a fundamentação da métrica IEVI, e detalha a arquitetura e funcionalidade da ferramenta *IntegrityGuard*. Na Seção 4, são apresentados e discutidos os resultados quantitativos e a convergência com o modelo *Zero Trust*. As limitações encontradas e as propostas de trabalhos futuros são exploradas nas Seções 5 e 6, respectivamente. Finalmente, a Seção 7 apresenta as conclusões deste estudo.

2. Referencial Teórico

Esta seção fundamenta os pilares de segurança em sistemas operacionais e conteinerização necessários para a compreensão das vulnerabilidades analisadas neste estudo. São discutidos a gestão de privilégios no ecossistema Linux, a arquitetura cliente-servidor da plataforma Docker, as metodologias de modelagem de ameaças STRIDE e DREAD, além de uma revisão crítica dos trabalhos relacionados no estado da arte.

2.1. Gestão de Privilégios e o Usuário Root no Linux

O modelo de segurança do sistema operacional *Linux* é fundamentado no controle de acesso discricionário, onde cada processo e arquivo é associado a um proprietário e a um grupo. No topo dessa hierarquia reside o usuário *root*, também conhecido como *superuser*, que detém controle absoluto sobre o sistema. Identificado tecnicamente pelo *User ID* (*UID*) 0, este usuário possui permissões para modificar qualquer arquivo, interromper processos e alterar configurações críticas do *kernel* (Rajyashree et al. 2024).

A arquitetura tradicional do *Docker* depende fortemente desses privilégios elevados, uma vez que o *Docker daemon* (*dockerd*) geralmente é executado com permissões de *root* para gerenciar recursos de baixo nível, como *namespaces* e *control groups* (*cgroups*) (Merkel et al. 2014). O compartilhamento do *kernel* entre o *host* e os *containers* significa que a barreira de isolamento é lógica, e não física como em máquinas virtuais (Sultan et al. 2019; Wong et al. 2023), como exemplificado na Figura 1.

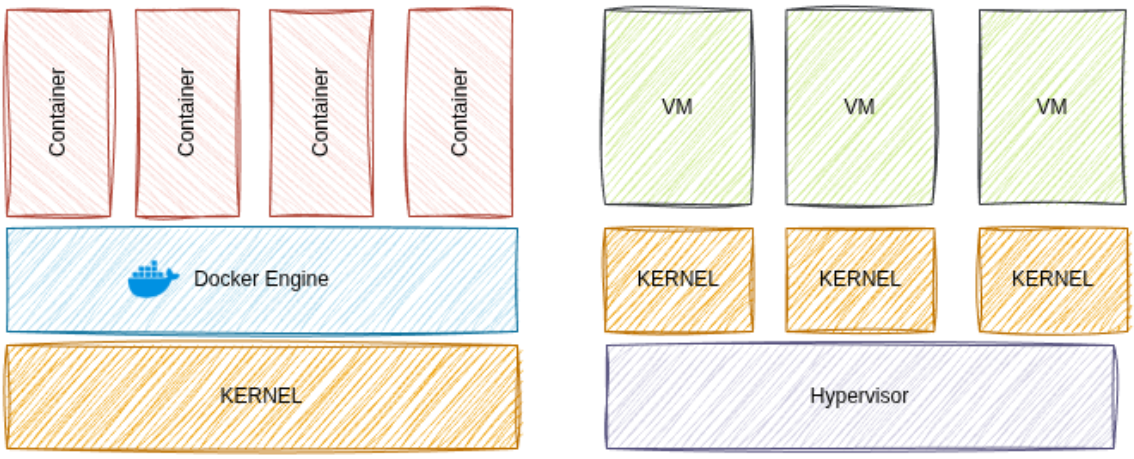

Figura 1. Docker vs VM.

Em ambientes multiusuário, a concessão inadvertida de acesso ao *socket* do *Docker* permite que um usuário comum instrua o *daemon* a realizar tarefas em seu nome. Como o *daemon* opera com o *UID* 0, qualquer operação solicitada é executada com a autoridade máxima do sistema. Esse cenário é a base para ataques de *Privilege Escalation*, onde um ator mal-intencionado utiliza o *runtime* de *containers* para contornar as restrições impostas a usuários convencionais e obter controle total sobre a infraestrutura hospedeira (Rajyashree et al. 2024; Wong et al. 2023).

2.2. Plataforma e Arquitetura Docker

O *Docker* é uma plataforma aberta voltada para o desenvolvimento, envio e execução de aplicações, permitindo a separação entre o software e a infraestrutura subjacente. Ele utiliza uma arquitetura cliente-servidor, na qual o cliente *Docker* (`docker`) interage com o *Docker daemon* (`dockerd`), responsável por gerenciar objetos como imagens, *containers*, redes e volumes. A comunicação entre esses componentes ocorre por meio de uma *API REST*, podendo ser estabelecida via *sockets* UNIX ou interfaces de rede (Docker 2026).

Diferente das máquinas virtuais tradicionais, os *containers* são instâncias executáveis de imagens que operam de forma leve e isolada, compartilhando o mesmo *kernel* do sistema hospedeiro (Wong et al. 2023). Essa eficiência é alcançada através de tecnologias do *kernel* Linux, como `namespaces`, que criam espaços de trabalho isolados, e `control groups` (cgroups), que gerenciam a alocação de recursos de hardware (Docker 2026).

2.2.1. *Docker Rootless* e o Princípio do Privilégio Mínimo

O modo *rootless* surge como uma evolução crítica na arquitetura da plataforma, permitindo que tanto o *Docker daemon* quanto os *containers* sejam executados sem a necessidade de privilégios de *root* no sistema hospedeiro. Enquanto a arquitetura tradicional depende de um *daemon* privilegiado que opera com *User ID* (*UID*) 0, o modo *rootless* utiliza a tecnologia de *User Namespaces* do *kernel Linux* para mapear um

usuário comum para o *root* interno do *container*, mantendo-o sem privilégios no *host* (Sultan et al. 2019; Raftopoulos 2025).

A importância desta abordagem é fundamentada na redução drástica da superfície de ataque. Ao eliminar a necessidade de inclusão de usuários no grupo *docker* e o acesso direto ao *socket* privilegiado, o sistema mitiga a principal vulnerabilidade explorada para *Privilege Escalation* (Rajyashree et al. 2024). Caso ocorra uma falha de segurança ou um *container escape*, o atacante fica restrito às permissões do usuário comum que iniciou o processo, impedindo o comprometimento total da infraestrutura e protegendo a integridade dos arquivos críticos do sistema operacional (VS et al. 2023; Wong et al. 2023). Em ambientes de uso compartilhado, como laboratórios acadêmicos, essa configuração é essencial para equilibrar a flexibilidade do desenvolvedor com a governança de segurança necessária (Docker 2026).

2.3. Modelagem de Ameaças: STRIDE e DREAD

A modelagem de ameaças é um processo estruturado essencial para identificar e mitigar riscos de segurança desde as fases iniciais do ciclo de vida de um sistema (Shostack 2014). No ecossistema de containers, essa prática permite mapear superfícies de ataque que abrangem desde o repositório de código até o runtime de execução no host (Wong et al. 2023). Este trabalho adota as metodologias STRIDE e DREAD para a categorização e priorização dos riscos identificados.

O modelo STRIDE, desenvolvido pela Microsoft, é amplamente reconhecido por sua maturidade e aplicabilidade em sistemas distribuídos. Ele classifica as ameaças em seis categorias distintas, cada uma associada à violação de uma propriedade de segurança fundamental (Wong et al. 2023):

1. *Spoofing* (Falsificação): Ocorre quando um atacante assume a identidade de um usuário ou serviço legítimo, violando a propriedade de autenticidade. Em ambientes Docker, isso pode incluir a falsificação de credenciais de acesso ao registro de imagens.
2. *Tampering* (Adulteração): Refere-se à modificação não autorizada de dados ou processos, comprometendo a integridade. A injeção de comandos maliciosos em um `Dockerfile` é um exemplo crítico nesta categoria.
3. *Repudiation* (Repúdio): Envolve a capacidade de um ator negar a realização de uma ação devido à falta de evidências auditáveis. O comprometimento de logs do daemon Docker pode facilitar ataques desta natureza.
4. *Information Disclosure* (Vazamento de Informação): Consiste no acesso não autorizado a dados sensíveis, violando a confidencialidade. A exposição do `docker.sock` pode permitir que um usuário comum visualize informações privilegiadas do host.
5. *Denial of Service* (Negação de Serviço): Visa tornar um serviço ou recurso indisponível, afetando a disponibilidade. O consumo excessivo de CPU ou memória por um container mal configurado exemplifica esta ameaça.
6. *Elevation of Privilege* (Elevação de Privilégio): Ocorre quando um usuário obtém permissões superiores às autorizadas originalmente. Esta é uma das ameaças mais críticas em hosts multiusuário, frequentemente explorada via sockets mal configurados.

Enquanto o STRIDE foca na identificação qualitativa, o modelo DREAD fornece uma abordagem quantitativa para a priorização de riscos. Ele permite calcular um índice de risco (de 1 a 10) com base na média de cinco critérios:

1. *Damage* (Dano): Avalia a extensão do impacto negativo caso a ameaça seja concretizada.
2. *Reproducibility* (Reprodutibilidade): Mede a facilidade com que o ataque pode ser replicado com sucesso.
3. *Exploitability* (Explorabilidade): Analisa o esforço e o conhecimento técnico necessários para executar o ataque.
4. *Affected Users* (Usuários Afetados): Estima a proporção de usuários ou sistemas impactados pela ocorrência da ameaça.
5. *Discoverability* (Descoberta): Avalia o quão fácil é para um atacante identificar a vulnerabilidade no sistema.

A integração desses dois modelos permite que administradores de ambientes acadêmicos e corporativos não apenas identifiquem onde estão as falhas no Docker, mas também quais devem ser mitigadas com maior urgência para garantir a resiliência operacional (VS et al. 2023; Sultan et al. 2019).

2.4. Trabalhos Relacionados e Lacunas na Literatura

Jarkas et al. (2025) (Jarkas et al. 2025) publicaram recentemente o levantamento mais abrangente sobre a segurança de contêineres até o momento, analisando mais de 200 vulnerabilidades e propondo uma taxonomia de 47 tipos de exploração em 11 vetores de ataque. O estudo ressalta que a pesquisa acadêmica ainda é predominantemente teórica e carece de dados empíricos robustos sobre a natureza das vulnerabilidades e explorações reais. Embora os autores forneçam um framework acionável para mapear mitigações de software e hardware, o trabalho mantém o foco em uma visão panorâmica e taxonômica do ecossistema de nuvem. É nesse hiato que este estudo se posiciona, ao prover não apenas a análise teórica, mas uma ferramenta prática de diagnóstico e uma métrica operacional (IEVI) voltada especificamente para o controle de integridade em *hosts* locais de uso compartilhado.

A literatura acadêmica recente apresenta diversas abordagens sobre a segurança de *containers*, com foco predominante em infraestruturas de nuvem e microsserviços (Sultan et al. 2019; Wong et al. 2023). O estudo de rajyashree2024 realiza uma investigação empírica sobre o *Docker socket* como vetor para *Privilege Escalation*, sugerindo defesas baseadas em mapeamento de *namespaces*. Contudo, o trabalho limita-se a cenários de usuário único e não explora os riscos de *Data Tampering* em larga escala ou a necessidade de automação do diagnóstico em ambientes com alta rotatividade de usuários, como laboratórios acadêmicos.

Em uma perspectiva de modelagem sistêmica, wong2023 utiliza o *framework* STRIDE para analisar a superfície de ataque de ecossistemas de *containers*. Embora forneça uma taxonomia abrangente de ataques, o estudo mantém-se no campo teórico-qualitativo, sem propor uma ferramenta de detecção ativa ou métricas que permitam a administradores priorizarem riscos de forma numérica. Complementarmente, vs2023 discute a aplicação do modelo DREAD em microsserviços, mas foca em vulnerabilidades de rede e API, negligenciando a governança de permissões locais no sistema hospedeiro.

A principal lacuna identificada reside na ausência de soluções que integrem o diagnóstico preventivo à prova de conceito (PoC) ativa em hosts multiusuário. Diferente dos trabalhos citados, esta pesquisa foca especificamente na governança de segurança em máquinas compartilhadas, integrando a detecção via *IntegrityGuard* à quantificação de risco pelo *IEVI*. Essa abordagem permite que o administrador visualize o impacto imediato na integridade do sistema operacional, algo que as modelagens puramente teóricas ou os escâneres estáticos de conformidade não provêm de forma consolidada (Tabela 1).

Tabela 1. Análise comparativa entre trabalhos relacionados e a abordagem proposta.

Trabalho	*Escalation*	*Tampering*	Multiusuário	Métrica	Ferramenta
(Rajyashree et al. 2024)	Sim	Não	Não	Não	Não
(Wong et al. 2023)	Sim	Sim	Não	Não	Não
(VS et al. 2023)	Sim	Sim	Não	Sim	Não
(Sultan et al. 2019)	Sim	Sim	Não	Não	Não
(Jarkas et al. 2025)	Sim	Sim	Parcial	Não	Não
Este Trabalho	**Sim**	**Sim**	**Sim**	**Sim**	**Sim**

3. Metodologia e Desenvolvimento da IntegrityGuard

A metodologia adotada combina análise teórica de riscos em conteinerização com experimentação controlada. O estudo integra os frameworks STRIDE e DREAD para identificar e priorizar vulnerabilidades associadas ao uso do `docker.sock` em ambientes multiusuário. Com base nesse embasamento, foi definido um arcabouço experimental capaz de reproduzir cenários reais de exposição, permitindo a análise sistemática de riscos de Violação de Integridade e Escalonamento de Privilégios.

Como contribuição prática, propõe-se a ferramenta *IntegrityGuard*, que automatiza o diagnóstico de segurança por meio de testes baseados em vetores de ataque conhecidos. A abordagem utiliza validação comparativa entre cenários vulneráveis (modo *rootful*) e mitigados (modo *rootless*), permitindo mensurar a redução da superfície de ataque com base no IEVI. Esta seção apresenta o ambiente experimental, a arquitetura da solução e os procedimentos de avaliação.

3.1. Ambiente Experimental

A definição do cenário de testes baseou-se em uma auditoria de segurança prática realizada nos laboratórios do IFMG, onde identificou-se que mais de 120 máquinas de uso compartilhado apresentavam vulnerabilidades críticas de governança, permitindo o acesso irrestrito ao *socket* UNIX (Rajyashree et al. 2024). Para validar as funcionalidades da ferramenta *IntegrityGuard* sem comprometer a infraestrutura institucional, foi desenvolvida uma *Virtual Machine* (VM) com configuração técnica idêntica às máquinas vulneráveis encontradas no *campus*. O ambiente controlado operou com Ubuntu 22.04 LTS e Docker Engine na versão 24.0.7 (Merkel et al. 2014). Reproduziu-se a falha de configuração original, garantindo que o *daemon* operasse em modo *rootful*, permitindo que usuários não privilegiados interagissem diretamente com o *kernel* hospedeiro através da API do Docker (Wong et al. 2023). Não foram implementadas camadas adicionais de *hardening*, como *AppArmor* ou *SELinux*, a fim de quantificar a eficácia isolada da mitigação via modo *rootless* (VS et al. 2023).

3.2. Arquitetura e Módulos da IntegrityGuard

A ferramenta *IntegrityGuard* foi desenvolvida em *Python* como uma *command-line interface* (CLI), visando auxiliar administradores de *hosts* na identificação da exposição do *docker.sock*. A ferramenta foi publicada no repositório oficial de pacotes Python, PyPI, sob o nome *integrityguard-cli*[1]. O software adota uma arquitetura modular composta por quatro componentes principais:

1. **Main CLI**: Orquestra a interação com o usuário e provê os argumentos necessários para a escolha dos formatos de saída estruturados (*JSON* (Figura 2) e *HTML* (Figura 4));
2. **IntegrityGuardTool**: Responsável por gerir a execução sequencial das rotinas de diagnóstico e coordenar a geração dos relatórios finais;
3. **IEVICalculator**: Implementa a lógica de mapeamento de severidade para os critérios DREAD e calcula a média final do índice IEVI;

```
-- Starting IntegrityGuard Analysis ---
*] Check User Docker Group Membership: VULNERABLE
*] Check Create File on Host via Container: VULNERABLE
*] Check Read /etc/shadow via Docker: VULNERABLE
*] Check Elevate Privilege via Container: VULNERABLE

-- JSON Output (Ready for HTML Report) ---

   "summary": {
       "ievi_score": 100.0,
       "risk_level": "CRITICAL",
       "dread_breakdown": {
           "D": 10,
           "R": 10,
           "E": 10,
           "A": 10,
           "Di": 10
       }
   },
   "findings": [
       {
           "check_id": "DX-001",
           "check_name": "User Docker Group Membership",
           "is_vulnerable": true,
           "severity": "Severity.CRITICAL",
           "description": "Checks if the user has root-equivalent access via docker group.",
           "evidence": "User 'labuser' found in 'docker' group.",
           "recommendation": "Remove the user from the 'docker' group to prevent root privilege escalation.",
           "trace": [
               "Identified current user: labuser",
               "Retrieving all groups and their members...",
               "User belongs to groups: sudo, docker",
               "Found 'docker' in user's group list."
           ]
       },
```

Figura 2. Interface de saída em formato *JSON* estruturado.

3.3. Procedimentos de Teste e Fluxo de Operação

O fluxo operacional inicia-se com a conexão ao *Docker engine* através da biblioteca *docker-py*. A metodologia de avaliação foi estruturada para validar quatro rotinas de diagnóstico fundamentadas em vetores de ataque conhecidos na literatura (Sultan et al. 2019; Wong et al. 2023):

1. **Verificação de Grupo Docker**: Identifica se o usuário pertence ao grupo secundário *docker*, configuração recorrentemente apontada como o principal facilitador para o acesso sem restrições ao *daemon*;

[1]Disponível em: `https://pypi.org/project/integrityguard-cli/`

2. **Verificação de Leitura Sensível**: Simula um ataque de *information disclosure* ao tentar instanciar um *container* que realiza a montagem do diretório *etc*, visando o arquivo *shadow*;
3. **Verificação de Escrita no Hospedeiro**: Avalia o risco de *data tampering* ao tentar criar arquivos de teste no diretório *tmp* do hospedeiro a partir de um processo isolado;
4. **Verificação de Escalonamento de Privilégios**: Representa o cenário de maior risco, onde a ferramenta tenta injetar um novo usuário administrativo no arquivo *passwd* do *host* com *User ID* (UID) 0.

O fluxo de exploração sistemático (Figura 3) demonstra a progressão lógica desde a validação de acesso inicial até o compromisso total do hospedeiro (Wong et al. 2023). Após as verificações no cenário vulnerável, aplica-se a mitigação através do modo *rootless*, que utiliza *user namespaces* para isolar as capacidades do usuário, e os testes são repetidos para quantificar a redução da superfície de ataque (VS et al. 2023; Docker 2026).

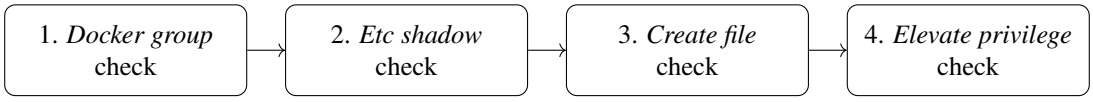

Figura 3. Fluxo de exploração fundamentado em vetores de ataque conhecidos.

3.4. Fundamentação da Métrica IEVI

O Índice de Exposição à Violação de Integridade (IEVI) é a métrica proposta para quantificar o risco operacional decorrente da exposição do *socket*. O IEVI utiliza os valores atribuídos às categorias do *framework* DREAD para consolidar um *score* centesimal (VS et al. 2023). A formulação é dada pela média aritmética simples dos critérios pontuados:

$$IEVI = \frac{D + R + E + A + Di}{5} \times 10 \tag{1}$$

Nesta equação, as variáveis representam o Dano Potencial (D), a Reprodutibilidade (R), a Explorabilidade (E), os Usuários Afetados (A) e a Descoberta (Di). A opção pela média simples fundamenta-se na necessidade de um diagnóstico equilibrado, onde a facilidade de exploração em ambientes acadêmicos possui relevância equivalente ao dano potencial. Esta abordagem mantém a objetividade proposta por (Shostack 2014), evitando vieses subjetivos na calibração de pesos específicos e permitindo que administradores estabeleçam prioridades de intervenção claras (Sultan et al. 2019).

4. Resultados e Discussão

A avaliação da ferramenta *IntegrityGuard* permitiu mensurar de forma quantitativa e qualitativa o risco prático associado à exposição do *socket* do Docker em ambientes multiusuário. No cenário inicial (*rootful*), caracterizado pela configuração padrão observada nos laboratórios, a ferramenta diagnosticou um *IEVI* de 100, atingindo o nível de severidade máxima (*CRITICAL*), conforme ilustrado no relatório da Figura 4.

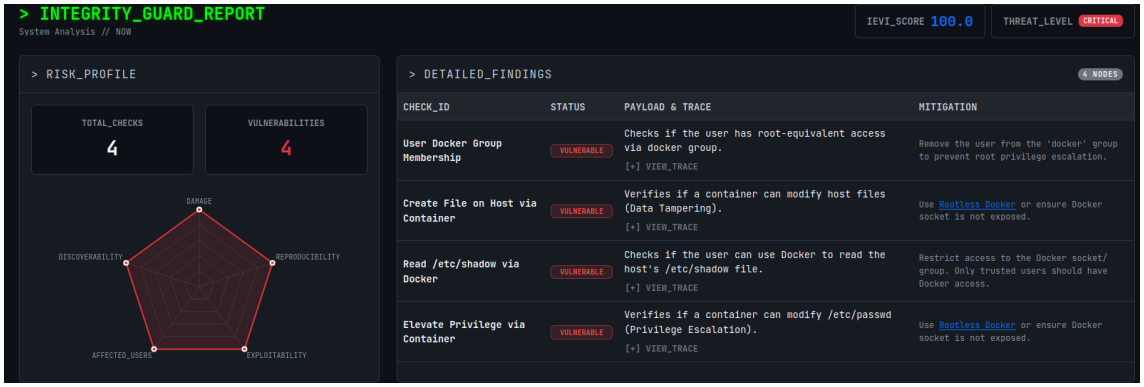

Figura 4. Interface do relatório visual da *IntegrityGuard* em cenário vulnerável.

Após a implementação da mitigação via modo *rootless*, o índice foi drasticamente reduzido para 32, o que enquadra o sistema em um nível de risco baixo, como apresentado na Figura 5. Esta redução de 68% na exposição valida a eficácia da transição para uma arquitetura fundamentada no princípio do privilégio mínimo e no isolamento de *user namespaces*.

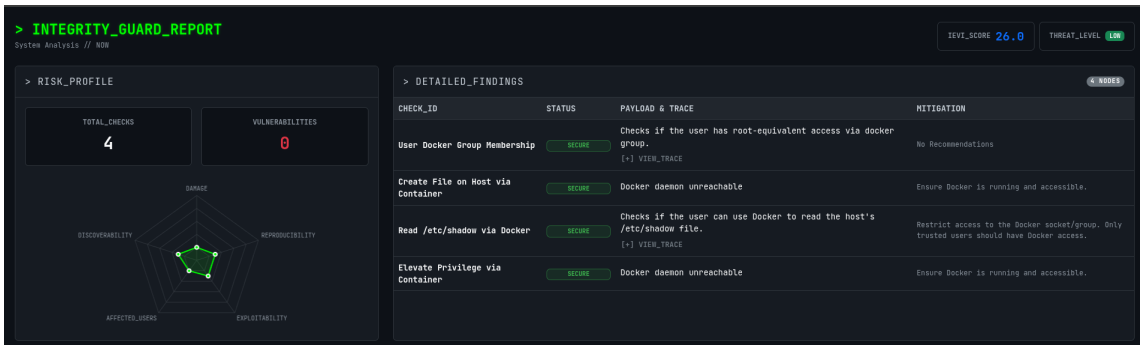

Figura 5. Interface do relatório em modo mitigado (Rootless).

Durante a execução dos testes no cenário vulnerável, o módulo *elevate privilege check* obteve sucesso na modificação do arquivo *etc/passwd* do hospedeiro em um tempo inferior a 20 s. Este resultado comprova empiricamente que o isolamento lógico provido pelo Docker em modo privilegiado é insuficiente para conter um usuário local mal-intencionado que possua acesso ao grupo *docker*. O diagnóstico automatizado confirmou que a falha de integridade (*data tampering*) permitiu a criação de um novo usuário administrativo, resultando na escalada imediata de privilégios para o nível *root* (Rajyashree et al. 2024).

A viabilidade técnica da solução foi avaliada através de indicadores de desempenho e consumo de recursos, consolidados na Tabela 2. Observou-se que a *IntegrityGuard* opera com um *overhead* computacional desprezível, apresentando um consumo de CPU inferior a 1% e pico de memória RAM de 58 MB. O tempo total de execução do diagnóstico completo foi inferior a 5 s, o que demonstra a eficiência da ferramenta para integração em rotinas de monitoramento contínua sem degradar a performance das estações de trabalho.

Tabela 2. Consolidação da pontuação IEVI e indicadores de desempenho computacional.

Métrica de Avaliação	Cenário Vulnerável	Cenário Rootless
Índice IEVI (Risco)	**100 (Crítico)**	**32 (Baixo)**
Tempo de Execução do Diagnóstico	4,5 s	3,9 s
Consumo de Memória RAM (Pico)	58 MB	42 MB
Uso de CPU do Hospedeiro	¡ 1%	¡ 1%

4.1. Análise Comparativa e Baseline

A necessidade da *IntegrityGuard* torna-se evidente ao contrastar seus resultados com ferramentas consolidadas como o *Docker Bench for Security*. Em testes práticos, o *Docker Bench* realizou 86 verificações baseadas no padrão *CIS Docker Benchmark*, resultando em um *Score: 0*. Embora útil para conformidade, a ferramenta exibiu limitações críticas para o diagnóstico de risco em ambientes multiusuário: diversas verificações falharam com a mensagem *"You must be root to run this program"*, evidenciando que a ferramenta de auditoria exige privilégios elevados para identificar falhas de segurança. Além disso, a recomendação de execução do *daemon* como não-root foi classificada apenas como uma nota de verificação manual, sem testar ativamente se um usuário comum poderia explorar a falha.

Diferente da abordagem de *compliance* estática, a *IntegrityGuard* diferencia-se pela execução de uma Prova de Conceito (PoC) ativa que valida a viabilidade real da ameaça. Enquanto ferramentas como o *Trivy* especializam-se na detecção de vulnerabilidades conhecidas (CVEs) em binários e imagens, a solução proposta foca no *runtime* e no escape de privilégios para o hospedeiro. A Tabela 3 detalha como a ferramenta preenche lacunas que as soluções de mercado tratam de forma genérica ou manual.

Tabela 3. Comparativo qualitativo de recursos entre ferramentas de auditoria de segurança.

Recurso de Segurança	*Docker Bench*	*Trivy*	***IntegrityGuard***
Escaneamento de CVEs (Imagens)	Não	Sim	Não
Conformidade CIS (Audit)	Sim	Não	Parcial
Execução sem privilégios de Root	Não	Sim	**Sim**
Cálculo de Risco de Integridade (IEVI)	Não	Não	**Sim**
Prova de Conceito de Exploração Ativa	Não	Não	**Sim**
Diagnóstico de Escalonamento Local	Manual	Não	**Sim**

Os resultados reforçam a convergência com o modelo *Zero Trust*. A mitigação sugerida pela *IntegrityGuard* remove a confiança implícita concedida pelo acesso direto ao *socket* privilegiado. Ao adotar o modo *rootless*, a infraestrutura reduz significativamente o *blast radius* de ataques baseados em falhas de configuração, garantindo que o atacante permaneça restrito ao contexto do usuário comum, preservando o *kernel* e os arquivos sensíveis do sistema (VS et al. 2023; Sultan et al. 2019; Wong et al. 2023).

5. Limitações do Estudo

Apesar dos resultados demonstrarem a eficácia da ferramenta *IntegrityGuard* na detecção de riscos em *hosts* multiusuário, este estudo apresenta limitações técnicas e metodológicas que devem ser consideradas para a interpretação dos dados, especialmente no que se refere à generalização dos resultados para ambientes heterogêneos, à abrangência dos cenários analisados e à ausência de validação em infraestruturas de produção com diferentes níveis de maturidade em segurança:

1. Escopo restrito ao vector *docker.sock*: A análise foca-se primordialmente na exploração da interface do *daemon* e na pertença inadequada ao grupo *docker* . Outros vectores críticos, como falhas de segurança no *kernel Linux*, o abuso de *capabilities* específicas do *runtime* ou vulnerabilidades em imagens provenientes do *Docker Hub*, não foram integrados no roteiro de testes automatizados da versão atual da ferramenta.
2. Dependência de mapeamento estático no *IEVI*: Embora o cálculo do índice seja automatizado pelo serviço *IEVICalculator*, as pontuações baseiam-se num mapeamento de severidade pré-definido (*Severity DREAD Map*) implementado no código-fonte. Esta abordagem, embora consistente, pode não capturar nuances específicas de infraestruturas com políticas de segurança compensatórias ou monitoramento dinâmica ativa.
3. Ambiente de experimentação controlado: A validação foi conduzida em uma *Virtual Machine* desenhada para replicar as vulnerabilidades observadas nos laboratórios do *IFMG*. Em cenários de produção de larga escala, fatores como a presença de *Intrusion Detection Systems* (*IDS*), latência de rede e sistemas de *logging* centralizado podem influenciar tanto o *Time-to-Exploit* (*TTE*) quanto a facilidade de descoberta do ataque.
4. Foco exclusivo na mitigação via *Rootless*: O estudo validou o impacto da transição para o modo *rootless* como principal contramedida. No entanto, outras estratégias de *hardening* complementares, tais como a implementação de perfis de *AppArmor*, *SELinux* ou a utilização de *runtimes* de isolamento como o *gVisor*, não foram objeto de análise comparativa nesta fase da pesquisa.
5. Confiabilidade do Diagnóstico (Falsos Positivos e Negativos): A abordagem da ferramenta baseia-se em testes determinísticos de Prova de Conceito (PoC), o que minimiza a ocorrência de Falsos Positivos. Uma vez que o diagnóstico de vulnerabilidade é emitido apenas após o sucesso real na criação ou modificação de arquivos no hospedeiro, não há margem para alarmes falsos derivados de suposições estáticas (Rajyashree et al. 2024; Sultan et al. 2019). Entretanto, a ocorrência de Falsos Negativos é uma possibilidade técnica caso camadas externas de segurança, como perfis restritivos de *AppArmor* ou políticas de *SELinux*, estejam ativas. Nestes cenários, a ferramenta pode falhar ao executar a PoC e reportar o sistema como seguro, embora o *socket* do Docker continue exposto para outros métodos de exploração não cobertos pelas rotinas atuais de teste (VS et al. 2023; Wong et al. 2023). Essa limitação reforça que a ferramenta deve ser utilizada como parte de uma estratégia de defesa em profundidade, e não como único validador de segurança da infraestrutura.

6. Trabalhos Futuros

A partir dos resultados alcançados com o desenvolvimento da ferramenta *IntegrityGuard* e da aplicação da métrica *IEVI*, identificam-se diversas oportunidades para a evolução desta pesquisa, especialmente no que diz respeito à ampliação da cobertura de ameaças, ao refinamento da modelagem de risco e à adaptação da solução a cenários mais complexos e dinâmicos, como ambientes distribuídos e infraestruturas de larga escala:

1. Ampliação do escopo de vulnerabilidades: Expandir os módulos de teste para avaliar vetores adicionais de ataque, como o abuso de *capabilities* do *Linux*, a execução de *containers* privilegiados e a exploração de potenciais *escapes* de *kernel*.
2. Evolução do *IEVI* e do *IEVICalculator*: Refinar a lógica de mapeamento de severidade para incorporar fatores dinâmicos de detecção e reduzir a dependência de ponderações estáticas, permitindo que o índice se adapte a diferentes contextos operacionais.
3. Integração em *Pipelines* de *DevSecOps*: Desenvolver conectores para que a ferramenta possa ser utilizada em fluxos de *Continuous Integration* e *Continuous Deployment* (*CI/CD*), permitindo a auditoria automatizada de *hosts* antes do *deployment* de novas aplicações.
4. Remediação Automática e *Hardening*: Integrar mecanismos que não apenas detectem a exposição, mas apliquem automaticamente medidas de mitigação, como a configuração assistida do modo *rootless* ou a geração de perfis restritivos de *AppArmor* e *SELinux*.
5. Validação em infraestruturas Heterogêneas: Conduzir estudos experimentais em *clusters* de larga escala e ambientes de nuvem pública para avaliar a eficácia do *IEVI* em sistemas com alta rotatividade de usuários e políticas de acesso complexas.
6. Modelagem Avançada de Ameaças: Aplicar formalismos complementares ao *STRIDE* e ao *DREAD*, como o *ATT&CK for Containers*, para enriquecer a análise de risco e a base de conhecimento da ferramenta.

Estas direções evidenciam o potencial de evolução contínua na proteção de ambientes *Docker* multiusuário, contribuindo para o desenvolvimento de mecanismos mais robustos, adaptativos e integrados aos fluxos modernos de engenharia de software, além de reforçar o alinhamento das práticas de governança de segurança com o estado da arte em tecnologias de conteinerização e defesa em profundidade.

7. Conclusão

A segurança em infraestruturas de conteinerização compartilhadas exige um equilíbrio rigoroso entre flexibilidade operacional e governança de privilégios. Esta conclusão sintetiza as principais contribuições técnicas do estudo e reafirma a importância da transição para modelos de acesso baseados em privilégios mínimos.

Este trabalho demonstrou que a configuração inadequada do *Docker* em ambientes multiusuário, especificamente a inclusão de usuários comuns no grupo *docker*, constitui uma vulnerabilidade crítica que compromete a integridade do sistema operacional. Através da execução da ferramenta *IntegrityGuard*, validou-se que o acesso ao *socket* privilegiado permite que atores mal-intencionados realizem ataques de *Data Tampering* e *Privilege Escalation* com baixo esforço técnico e elevado impacto sistêmico.

A principal contribuição técnica deste estudo foi o desenvolvimento do ecossistema *integrityguard-cli*, publicado no repositório *PyPI*, que automatiza o diagnóstico de segurança e provê relatórios estruturados em *JSON* e *HTML*. A introdução do Índice de Exposição à Violação de Integridade (*IEVI*) permitiu uma quantificação objetiva do risco, fundamentada no *framework DREAD*, oferecendo aos administradores de sistemas uma métrica precisa para priorizar ações de *hardening*.

Os resultados experimentais confirmaram que a adoção do modo *rootless* reduz drasticamente a superfície de ataque, baixando o *score* do *IEVI* de um nível crítico para um patamar tolerável. Esta mitigação alinha-se aos princípios de *Zero Trust*, garantindo que mesmo em caso de falha no isolamento do *container*, o atacante permaneça restrito aos privilégios de um usuário comum, protegendo o *kernel* e arquivos sensíveis do hospedeiro.

Em última análise, este trabalho provê uma base metodológica e uma ferramenta prática para aprimorar a segurança em infraestruturas compartilhadas. A conscientização sobre os riscos do *socket* privilegiado e a transição para arquiteturas sem privilégios de *root* são passos fundamentais para a consolidação de políticas de segurança robustas no uso de tecnologias de conteinerização.

Referências

Docker, I. (2026). Docker documentation. *https://docs.docker.com/*.

Jarkas, O., Ko, R., Dong, N., and Mahmud, R. (2025). A container security survey: Exploits, attacks, and defenses. *ACM Computing Surveys*, 57(7):1–36.

Merkel, D. et al. (2014). Docker: lightweight linux containers for consistent development and deployment. *Linux j*, 239(2):2.

Raftopoulos, C. (2025). Cloud security with docker and kubernetes. Master's thesis, Πανεπιστήμιο Πειραιώς.

Rajyashree, R., Mathi, S., Saravanan, G., and Sakthivel, M. (2024). An empirical investigation of docker sockets for privilege escalation and defensive strategies. *Procedia Computer Science*, 233:660–669.

Shostack, A. (2014). *Threat modeling: Designing for security*. John wiley & sons.

Sultan, S., Ahmad, I., and Dimitriou, T. (2019). Container security: Issues, challenges, and the road ahead. *IEEE Access*, 7:52976–52996.

VS, D. P., Sethuraman, S. C., and Khan, M. K. (2023). Container security: precaution levels, mitigation strategies, and research perspectives. *Computers & Security*, 135:103490.

Wong, A. Y., Chekole, E. G., Ochoa, M., and Zhou, J. (2023). On the security of containers: Threat modeling, attack analysis, and mitigation strategies. *Computers & Security*, 128:103140.