Universidade Federal do Maranhão
Centro de Ciências Exatas e Tecnologia
Curso de Ciência da Computação

Ulisses dos Santos Jacinto

Blindagem de Ambientes Virtuais Baseados em Contêineres: *Docker Hardening*

São Luís
2022

Ulisses dos Santos Jacinto

Blindagem de Ambientes Virtuais Baseados em Contêineres: *Docker Hardening*

Monografia apresentada ao curso de Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Mário Antonio Meireles Teixeira

São Luís
2022

Ficha gerada por meio do SIGAA/Biblioteca com dados fornecidos pelo(a) autor(a). Diretoria Integrada de Bibliotecas/UFMA

dos Santos Jacinto, Ulisses.
Blindagem de Ambientes Virtuais Baseados em Contêineres: Docker Hardening / Ulisses dos Santos Jacinto. - 2022.
52 f.

Orientador(a): Mário Antonio Meireles Teixeira.
Monografia (Graduação) - Curso de Ciência da Computação, Universidade Federal do Maranhão, São Luís, 2022.

1. Blindagem. 2. Contêiner. 3. Docker. 4. Linux. 5. Segurança de redes. I. Teixeira, Mário Antonio Meireles. II. Título.

Ulisses dos Santos Jacinto

Blindagem de Ambientes Virtuais Baseados em Contêineres: Docker Hardening

Monografia apresentada ao curso de Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Aprovado em: 26/07/2022

BANCA EXAMINADORA

Prof. Dr. Mário Antonio Meireles Teixeira
Orientador
Universidade Federal do Maranhão

Prof. Dr. Geraldo Braz Junior
Examinador 1
Universidade Federal do Maranhão

Prof. Dr. Francisco José da Silva e Silva
Examinador 2
Universidade Federal do Maranhão

São Luís
2022

AGRADECIMENTOS

Agradeço primeiramente aos meus pais Racquel e Hmenon, por sempre me darem a melhor educação possível, conforto e oportunidade para o bem do meu futuro. Agradecimento especial a minha mãe, que nos últimos anos demonstrou estar comigo sempre e me ofereceu muito amor e apoio.

A minha tia, Gisele, pelas oportunidades, pelo apoio, compreensão, acompanhamento, acolhimento e por me guiar nessa jornada de estudante acadêmico.

Ao meu irmão, Aquiles, que apesar de estar muito distante, sempre me ajudou com conselhos sobre a vida acadêmica, e sempre demonstrou preocupação com meu futuro buscando sempre o melhor para mim.

Ao meu amigo Vinícius, que me apoiou muito em momentos difíceis da vida, me ajudou muito nos últimos períodos acadêmicos, sempre foi uma pessoa compreensível e um bom amigo.

Aos meus velhos amigos Diego, Felipe, João Marco e João Pedro que me propiciaram momentos muito felizes junto com Aécio, Francisco, Rodrigo Albuquerque, Rodrigo Alexandre, Pedro Paulo, Luís, Rafael Bessa e Sávio.

Agradeço a minha companheira de vida, Larissa, esta pessoa incrível que esteve comigo desde o começo, sempre me apoiou, me ajudou nos momentos mais difíceis, me encheu de amor, carinho e determinação e me proporcionou os momentos mais felizes da minha vida.

Agradeço ao corpo docente do curso de Ciência da Computação da Universidade Federal do Maranhão especialmente ao meu orientador Prof. Mário Meireles, aos seus ensinamentos, seu tempo, disponibilidade e todo o suporte necessário no desenvolvimento deste trabalho e agradeço a ex-coordenadora do curso Profª Simara Rocha pela compreensão, paciência e atenção.

A minha amiga Rosélia que está comigo há muito tempo, uma pessoa que organizou muito minha vida, me apoiou muito, tem sido uma excelente companheira e uma pessoa de extrema confiança.

Aos meus colegas passageiros de curso que dividiram tempo, conhecimento, alegria comigo e que de alguma forma colaboraram para que esse momento fosse possível.

Agradeço a mim mesmo por ter persistido nessa jornada, encontrado caminhos de seguir em frente e nunca ter desistido.

"Se o conhecimento pode criar problemas,
não é através da ignorância que
podemos solucioná-los."
Isaac Asimov

RESUMO

As práticas de conteinerização e virtualização se tornaram extremamente populares para atender uma demanda de implementação de sistemas com acesso remoto, principalmente aplicações em nuvem. Nesse contexto, a plataforma Docker foi criada com o objetivo de implantar sistemas baseados em contêineres de forma rápida, prática, altamente compatível com diversos sistemas operacionais e facilmente escalável, que se tornou popular e em seguida se manteve como padrão de contêineres na comunidade de TI. Entretanto, aplicações em contêineres introduzem novos riscos, pois compartilham recursos de rede, kernel e sistema de arquivos com a máquina host. Neste trabalho, são discutidas as principais vulnerabilidades de contêineres e métodos para blindá-los sobre o contexto de segurança de redes, Docker e Linux. Por fim, é apresentado um estudo de caso comparando um sistema contendo configurações padrões do Docker com as configurações utilizando técnicas de Docker *hardening*.

Palavras-Chave: Docker; Blindagem; Segurança de redes; Contêiner; Linux

ABSTRACT

The practices of containerization and virtualization became extremely popular to supply the demand of implementation of systems with remote access, mainly with cloud applications. In this context, the Docker platform was created with the objective to deploy container-based systems in a quick and practical way, highly compatible with several operating systems and easily scalable, which became popular, then it stood as standard containers on IT community. However, container applications introduce new risks, because they share same resources of network, kernel, and file systems with host machine. In this work, it is discussed the major container vulnerabilities and methods to harden them on the context of network security, Docker, and Linux. Finally, it is presented a case study comparing a system containing standard Docker configurations with another using Docker hardening techniques.

Keywords: Docker; Hardening; Network security; Container; Linux

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

PaaS	Plataform as a Service
MiTM	Man in The Middle
DDoS	Distributed Denial of Service
CVE	Common Vulnerabilities and Exposures
DMZ	Zona Desmilitarizada
DAC	Controle de Acesso Discricionário
MAC	Controle de Acesso Obrigatório
SaaS	Software as a Service
API	Interface de Programação de Aplicações
DCT	Docker Content Trust
TLS	Transport Layer Security
HTTP	Hypertext Transfer Protocol

SUMÁRIO

1 INTRODUÇÃO

Devido à necessidade de utilização de sistemas de larga escala remotamente, a popularidade dos contêineres vem crescendo por utilizar menos recursos comparado com uma máquina virtual (outro tipo de virtualização), além de ser mais simples de escalar, desenvolver e fazer manutenções. Muitas empresas estão adotando soluções baseadas em contêineres devido a sua praticidade e custo baixo, o que aumenta o número de desenvolvedores trabalhando com essa tecnologia que recebe suporte constantemente.

A conteinerização é uma técnica de virtualização que se conecta por toda camada de infraestrutura, fornecendo um sistema operacional virtualizado ou simulado que isola o ambiente de aplicação da máquina local (PAHL, 2015). Contêineres funcionam como ferramentas *plataform-as-a-service* (PaaS) que consistem em hospedar e implementar *software* que provém de aplicações (como serviço) para internet. Ao mesmo tempo que os contêineres são portáteis, eles possuem uma grande interoperabilidade, dado que utilizam princípios de virtualização.

Entretanto, os contêineres introduzem um novo risco por compartilhar o mesmo sistema de arquivos, rede e *kernel* da máquina hospedeira, permitindo que os atacantes invadam a máquina local e comprometam outros contêineres alocados (RAJ et al., 2016). Portanto, é essencial o uso de técnicas de blindagem (*hardening*) sobre contêineres, na implantação de sistemas, para protegê-los de possíveis ataques e acessos não autorizados.

Com base no contexto de vulnerabilidade de redes, este trabalho visa demonstrar a importância da segurança de sistemas virtuais, apresentar os principais riscos de ataques e mostrar o desenvolvimento de camadas de proteção para contêineres utilizando a plataforma Docker.

Para avaliação de segurança, pretende-se instalar a ferramenta na máquina local, configurar um sistema com múltiplos contêineres (que se comunicam entre si) e realizar testes de segurança, através de uma ferramenta oficial do Docker, para comparar com a configuração padrão de implantação e com a configuração de blindagem.

1.1 Objetivos

É objetivo geral deste trabalho:

Propor uma configuração de implantação do Docker, que seja mais segura que as configurações padrões desta plataforma. A proposta de configuração consiste em proteger o sistema contra ataques conhecidos, impondo uma resistência maior a ataques que explorem falhas de segurança.

1.1.1 Objetivos Específicos

- Aplicar os princípios de segurança da informação na prática;
- Compreender virtualização de contêineres usando Docker e suas características;
- Apresentar os riscos de ataques e como eles podem comprometer sistemas;
- Propor uma solução de blindagem para arquiteturas *web* e banco de dados baseadas na plataforma Docker;

1.2 Estrutura do trabalho

Este trabalho contém 5 capítulos que se dividem em suas respectivas seções e subseções. No capítulo 2, Fundamentação teórica, contém os princípios de segurança de redes, introduz os tipos de ameaças e apresenta a base do trabalho contendo blindagem de Linux.

No capítulo 3 é apresentado os conceitos de ambientes virtuais, a diferença entre contêiner e máquina virtual e introdução da ferramenta Docker que é utilizada como referência para estudo dos contêineres virtuais. Neste capítulo ainda são pontuadas boas práticas de segurança junto com as principais técnicas de Docker *hardening*.

No capítulo 4 é discorrido sobre o estudo de caso que usa um sistema de contêineres contendo uma página *web* em PHP, conectado com um servidor Apache e um banco de dados MySQL. Esta estrutura inicialmente está com as configurações padrões do Docker e ao decorrer do estudo foi sendo mais protegida por técnicas de segurança apresentadas no capítulo 3. Após aplicação das técnicas, os dois sistemas são comparados através do *script Docker Bench for Security*, uma ferramenta de avaliação de segurança oficial do Docker.

Por fim, o capítulo 5 recapitula os tópicos abordados no trabalho, do que foi estudado e apresentado, discorre sobre os resultados do estudo de caso e o impacto geral em segurança de Docker, reporta a importância deste trabalho, realiza as considerações finais e cita sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Todos os sistemas de informação possuem suas funcionalidades, características, estruturas e finalidades diferentes. Redes de computadores em cada universidade, empresa ou órgão tem suas determinadas arquiteturas, protocolos, tecnologias e diferentes meios de comunicação. Entretanto, para manter a integridade desses sistemas, é necessário seguir um conjunto padrão de regras de segurança, em que as implementações variam de acordo com cada sistema.

Em um sistema de contêiner, essas regras continuam se mantendo para segurança e para protegê-lo são utilizadas técnicas baseadas em Linux *hardening*, pois a ferramenta Docker utiliza recursos do núcleo do Linux. Além de manter a integridade seguindo os princípios de segurança, é necessário se alertar das ameaças expostas ao sistema. Nas seguintes seções serão explanados estes conceitos que servem de base para o trabalho.

2.1 Princípios de Segurança de Redes

Segundo Tanenbaum (2010), os problemas de segurança de redes podem ser divididos nas seguintes áreas: sigilo, autenticação, não repúdio e controle de integridade. Embora essas definições sejam especificadas para redes de computadores, elas se entrelaçam também nas áreas de sistemas operacionais. O próprio Tanenbaum (2016) define segurança de sistemas de informação em três componentes: confidencialidade, integridade e disponibilidade, também conhecidos como “CIA” (*confidentiality, integrity, availability)* resumidos na Tabela 1.

Dado as propriedades fundamentais da segurança, a primeira, confidencialidade, consiste em que os dados secretos, permaneçam assim. Se o proprietário dos dados os disponibilizar apenas para um determinado grupo de pessoas, o sistema deve garantir que o acesso a esses dados não seja liberado para pessoas não autorizadas.

A segunda propriedade, integridade, não permite que usuários não autorizados sejam capazes modificar algum dado sem a permissão do proprietário. Esta modificação inclui alterar tais dados, acrescentar dados (podendo conter dados falsos) ou removê-los.

Enquanto a terceira propriedade, disponibilidade, certifica que ninguém pode tornar o sistema inutilizável, consumindo recursos que desacelerem ou travem o sistema.

Tabela 1 - Metas e ameaças de segurança

Meta	Ameaça
Confidencialidade	Exposição de dados
Integridade	Manipulação de dados
Disponibilidade	Recusa de serviço

Fonte: (TANENBAUM, 2016)

Essas três propriedades são consideradas como os pilares da segurança da informação, no entanto, em um sistema de redes é incrementado mais duas propriedades: não repúdio e autenticação. O não repúdio trata de assinaturas, para certificar a identidade de um remetente e/ou destinatário, enquanto a autenticação, para determinar com quem está se comunicando antes de revelar informações.

Rankin (2017) complementa com mais três regras: o princípio do privilégio mínimo, defesa em profundidade e compartimentalização. A regra do privilégio mínimo consiste em que alguém somente precisa ter o mínimo de privilégio para realizar seu trabalho e nada mais. Essa regra serve para usuários, que somente podem acessar e executar diretórios, arquivos e tarefas específicas, mas também se aplica a serviços em que, se não há necessidade de comunicação entre eles, esta deve ser desabilitada. Enquanto o controle de usuários pode ser feito pelo administrador de redes, o controle de serviços é feito pelo *firewall*.

A ideia do princípio de defesa em profundidade é de não criar somente um tipo de defesa, mas sim várias camadas de proteção. Se somente uma estratégia de defesa for usada e esta for comprometida, então o sistema estará desprotegido.

E por fim, a compartimentalização, que propõe a divisão da disposição dos dados e da rede para diferentes serviços. O objetivo é isolar a base de dados e adicionar restrições e divisões na rede. Se uma rede for atacada, ela é comprometida por completo, porém se ela estiver dividida, o ataque não irá se espalhar pelas partes da infraestrutura a que o ataque não tem acesso.

2.2 Tipos de Ameaças

Ataques cibernéticos são muito comuns atualmente devido à diversidade de sistemas e tecnologias, acesso remoto à informação, velocidade de transmissão de dados e infraestruturas desatualizadas. Todos esses fatores potencializam a cobertura de ataque

e facilitam a descoberta de vulnerabilidades para infecção de sistemas, pois o atacante necessita somente burlar um dos princípios de segurança para realizar o ataque.

Há diversas maneiras de atacar um sistema, que depende dos fatores citados acima e a gravidade de cada ataque depende do nível de proteção da rede. Entre as ameaças, as mais comuns são:

- *Malware*: Um software malicioso que altera o comportamento de uma máquina, compromete a integridade de arquivos e pode infectar uma rede inteira de computadores (*ransomware)*, impedindo o acesso a diversos arquivos.

- *Man-in-the-middle* (MiTM): Refere-se à interceptação do tráfego de uma rede em que o atacante rouba informações, em algumas situações podendo conter dados sigilosos ou credenciais.

- Negação de serviço: Um tipo de ataque que pretende esgotar recursos do sistema, rede ou servidor para torná-lo inutilizável. Esse tipo de ataque muito comum pode ocorrer com múltiplos dispositivos sendo agentes do ataque conhecido com *Distributed Denial of Service* (DDoS)

- Injeção de código: Uma forma de inserir código malicioso através de uma API que recebe uma entrada do usuário, num website por exemplo, se não tratado corretamente, o código é executado no servidor, permitindo o atacante comprometer a rede.

- Escalonamento de privilégio: Quando um atacante invade uma rede, ele pode invadir tendo acesso restrito, ainda sim é uma situação indesejável, porém, ele pode ganhar acesso adicional se ele explorar algum processo que execute com privilégios maiores.

- Vulnerabilidade de dia zero: Acontece quando um ataque ocorre depois que uma vulnerabilidade conhecida é descoberta, mas antes de uma correção de segurança ser implementada.

Os tipos de ataques ainda podem se estender para qual tipo de aplicação está em execução, o tipo de rede que está sendo utilizada, as possibilidades são infinitas, porém o foco deste trabalho é explorar falhas de segurança do Docker.

2.3 Técnicas de Hardening

O termo "*hardening"* vem do verbo em inglês "*to harden*" que significa endurecer, dificultar. Em computação, esse termo se refere a blindar ou proteger um sistema qualquer de ameaças desconhecidas. Compreende técnicas para fortalecer os princípios de segurança que tornam um sistema blindado e o diferenciam da instalação padrão. O objetivo destas técnicas é mitigar as ameaças dos ataques apresentados acima.

Quando uma técnica de hardening é aplicada, há três fatores que devem ser levados em consideração, são eles: Segurança, Risco e Flexibilidade (REIS; JULIO, 2010). Ter um sistema totalmente seguro, não só não é possível como também impraticável, porém, quanto mais segura for a rede, menos riscos irá correr, contudo menos flexibilidade estará disponível, como ilustrado na Figura 1. O equilíbrio entre esses fatores não segue uma regra bem definida, cabe ao administrador de redes avaliar bem a situação antes de qualquer implementação.

Figura 1 - Representação gráfica dos três fatores: segurança, risco e flexibilidade.

Fonte: (REIS; JULIO, 2010).

2.4 Linux Hardening

De acordo com o CVE (*Common Vulnerabilities and Exposures*), os cinco produtos que apresentaram mais vulnerabilidades distintas, em 2021, na ordem foram: Debian, Android, Ubuntu, Fedora e Mac OS. Desses cinco sistemas citados, três deles são distribuições Linux, sendo o Debian em primeiro lugar com mais de 6.000 vulnerabilidades distintas.

Isso não significa que o Linux seja um sistema pouco protegido ou que não deveria ser usado, mas por ser de código aberto, a comunidade de desenvolvedores acaba reportando muitas vulnerabilidades, enfatizando as mais severas. Entretanto, qualquer que seja o sistema operacional, as falhas de segurança geralmente acontecem por má configuração de servidores e para melhorá-la, as técnicas de *Linux Hardening* são utilizadas.

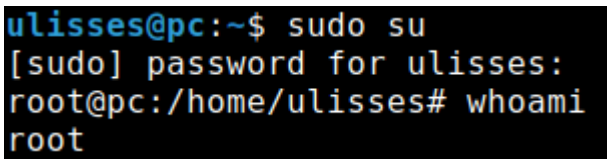
As técnicas apresentadas a seguir podem variar de acordo com a finalidade do sistema e os tipos de aplicação em desenvolvimento. Antes de tudo, recomenda-se sempre instalar o sistema na versão mais atual contendo os *patches* de segurança, corrigindo vulnerabilidades conhecidas. É altamente recomendado também não utilizar sistemas que foram descontinuados e que não são mais atualizados.

2.4.1 Definindo Grupos para o Comando Sudo

Seguindo o princípio do privilégio mínimo, o comando *sudo* permite que um usuário execute um comando como outro usuário. No contexto de configuração de redes, serve para que o usuário comum realize uma tarefa específica de administrador relacionada especificamente ao seu trabalho (sem que comprometa sua senha) e serve para criar grupos de administradores com privilégios completos.

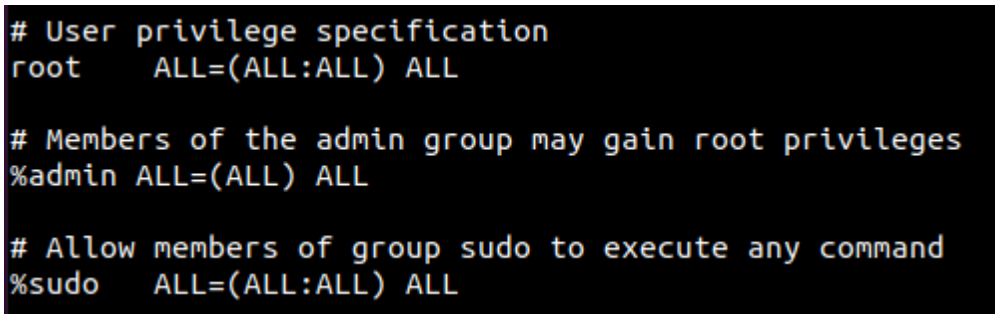
Na Figura 2 é apresentado um exemplo de uso de *sudo*, onde o usuário ulisses obtém acesso ao sistema como *root* (privilégios máximos de raiz). Na Figura 3 é possível ler e alterar o arquivo /etc/sudoers que contém as especificações de privilégio, grupo de membros administrativos e membros do grupo sudo que podem executar qualquer comando.

Figura 2 - Exemplo do comando sudo.



Fonte: O autor.

Figura 3 - Configuração do arquivo /etc/sudoers.



Fonte: O autor.

Observa-se que assim como o *root* possui permissão total (root ALL=(ALL:ALL) ALL), os membros do grupo sudo e *admin* também possuem. O

primeiro “ALL” indica que essa regra se aplica a todos os *hosts*, o segundo e o terceiro indicam respectivamente que este usuário pode executar comandos como todos os usuários e grupos, e por último “ALL” indicam que esta regra se aplica a todos os comandos.

A partir da alteração desse arquivo, pode-se criar níveis de privilégios para cada usuário, grupo e tipos de comando que poderão ser executados. Uma boa configuração do /etc/sudoers garante que cada admin, grupo e usuário tenham seus respectivos privilégios e sejam impedidos de fazer algum acesso não autorizado.

2.4.2 Firewall

Firewall é um dispositivo físico ou programa que aplica um conjunto de regras para proteger parte de uma rede, nas corporações é usado para proteger os servidores de uma zona desmilitarizada (DMZ), região externa não confiável (a internet), sendo a primeira barreira de proteção de uma rede. O objetivo é manter um *firewall* o mais rígido possível para proteger a rede interna.

O Linux possui o comando *iptables* para configurar o *firewall* netfilter que qualquer distribuição Linux possui. O *iptables* consiste em três tabelas, e elas são:

- *Filter*: Para a proteção de servidores e clientes, tabela normalmente mais usada.
- NAT: *Network Address Translation* usada para conectar a internet com a rede privada.
- *Mangle*: Alterar os pacotes de rede que passam pelo *firewall*.

Na Figura 4 observa-se a estrutura do iptables que compõem a categoria, o protocolo e a ação. A categoria define se o tráfego está vindo, saindo ou se deslocando de uma rede para outra (INPUT, OUTPUT ou FORWARD). O protocolo usado para conexão (TCP, UDP, ICMP, HTTP, SSH etc) e a ação sobre os pacotes: se irá permitir, rejeitar, ignorar ou informar no log do sistema. Essas regras variam de acordo com o endereço IP de fonte ou destino, porta usada e tipo de serviço.

Figura 4 - Composição básica do iptables na ordem: categoria, protocolo e ação.

```
sudo iptables -A <category> -p <protocol> -j <action>
```

Fonte: (RANKIN, 2017).

2.4.3 Ativando Criptografia

Quando a criptografia é ativada, se tem dois níveis de proteção: Os atacantes não podem ler qualquer informação sensível da rede e os atacantes não podem modificar ou inserir tráfego numa conexão criptografada (RANKIN, 2017). A criptografia serve tanto para proteger dados estáticos (alocados numa base de dados), como dados correntes (numa transmissão) podendo ser aplicada de duas maneiras: simétrica e assimétrica.

Numa criptografia de chave simétrica, é usada a mesma chave para codificação e descodificação da mensagem, enquanto na criptografia assimétrica é utilizada uma chave pública, para cifrar e uma chave privada para decifrar a mensagem.

Pode-se usar uma chave simétrica para criptografia de arquivos localmente, mas para transferência deles se um atacante interceptar a conexão, ele pode roubar as chaves e utilizar essa mesma chave para descriptografar os arquivos. Numa conexão de rede é recomendado utilizar a criptografia assimétrica. A chave pública é distribuída na rede e serve para decifrar a mensagem e confirmar a assinatura de uma chave privada, enquanto a chave privada é guardada, usada para decifrar a mensagem e assinar para garantir quem mandou a mensagem.

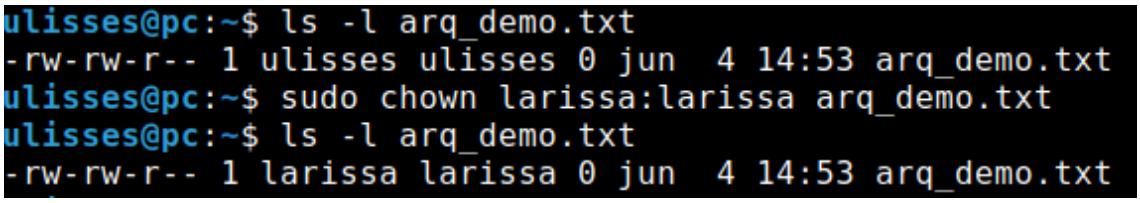
No Linux é utilizado o comando “gpg” para criptografar arquivos e o SSH para criar temporariamente túneis de conexão criptografados, que pode ser usado para acessar terminais de forma remota e transferir arquivos.

2.4.4 Controle de Acesso Discricionário

O Controle de Acesso Discricionário (DAC) permite que qualquer usuário tenha acesso aos diretórios abertos especificados, controlando o acesso autorizado a somente aqueles que podem acessá-los. O Linux possui os comandos “chown” e “chmod” para mudar o proprietário e as permissões dos arquivos e diretórios.

O comando chown, que exige privilégios sudo, permite mudar o usuário dono do arquivo ou diretório. Na Figura 5 é mostrada a execução do comando “ls -l” para mostrar as permissões com o proprietário do arquivo e o comando “chown” para mudar o acesso do arquivo “arq_demo.txt” de ulisses para larissa. No primeiro argumento “larissa” de “larissa:larissa” serve para indicar quem será o novo dono do arquivo e o segundo parâmetro para o grupo associado ao arquivo.

Figura 5 - Uso do comando “chown” para mudar o acesso aos arquivos.



Fonte: O autor.

Qualquer arquivo do Linux possui três permissões básicas: “r”, “w” ou “x” que significa ler, escrever ou executar respectivamente. No comando “ls -l” no arquivo “arq_demo.txt” (ver Figura 5), é possível ver algumas informações do arquivo. No primeiro caractere, representa se é um diretório (d) ou caractere (-), um traço nesse caso. Os próximos três caracteres “rw-” indicam se pode ser lido e escrito pelo usuário proprietário, os três seguintes “rw-” as permissões do grupo e o “r--” as permissões para outros.

Essas permissões podem ser alteradas pelo comando “chmod” usando uma representação octal dos valores 4, 2, 1 para ler, escrever e executar e 7 para os três ao mesmo tempo (ver Figura 6).

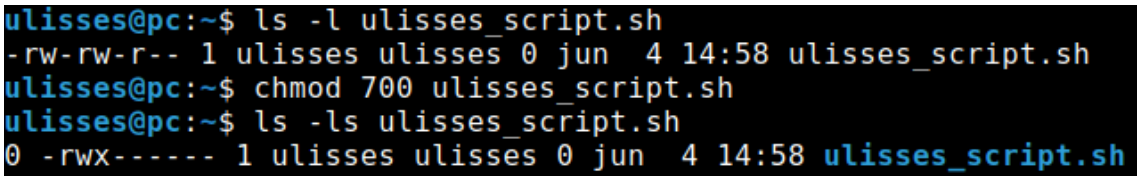
Figura 6 - Relação octal dos números e das permissões.

User	Group	Others
rwx	rwx	rwx
421	421	421
7	7	7

Fonte: (TEVAULT, 2018).

No exemplo da Figura 7, mudam-se as permissões do arquivo “ulisses_script.sh” para somente o usuário “ulisses” poder ler, escrever e executar. Com a combinação dos comandos “chown” e “chmod” é possível refinar o acesso a diretórios e arquivos, evitando que pessoas não autorizadas façam alterações indesejadas, leiam informações sensíveis ou se aproveitem de programas com privilégio maior.

Figura 7 - Execução do comando “chmod”.



Fonte: O autor.

2.4.5 Acesso de Controle Obrigatório

Controle de Acesso Obrigatório (MAC) é um recurso que utiliza módulos do *kernel*, em conjunto com atributos estendidos do sistema de arquivos, para garantir que somente usuários e processos autorizados possam, acessar arquivos sensíveis ou recursos do sistema (TEVAULT, 2018). É um método que funciona muito bem em conjunto com DAC, da seção acima, que rotula alguns arquivos com uma classificação de segurança através de políticas ou *policies*, um conjunto de regras que permitem acesso a objetos (arquivos, diretórios, portas, dispositivos, serviços, etc) a somente sujeitos (chamadas de sistema, processos, *threads*, etc) que possuem um rótulo de segurança.

O conjunto de regras estão dispostas em um arquivo chamado *profile* e podem ser gerenciados pelos programas SELinux, AppArmor ou Seccomp. O programa a ser escolhido para proteção irá depender da preferência do administrador de redes, eles fazem basicamente a mesma coisa, o que muda entre eles é a sintaxe das *policies*.

2.4.6 Ferramentas de Auditoria

Enquanto os arquivos de *log* do sistema Linux são bons, eles não nos dão um bom panorama de “quem faz o quê” ou “quem acessa o quê”. Pode ser intrusos ou pessoas de dentro (da rede) que não são autorizados a acessar tais arquivos (TEVAULT, 2018). Para tanto, existem ferramentas de auditoria de sistemas que informam sobre os acessos não autorizados.

Lynis é uma ferramenta de segurança para sistemas rodando Linux, macOS ou sistemas operacionais baseados em Unix (CISOFY, 2007). Esta ferramenta é usada para teste de penetração, auditoria de segurança, detecção de vulnerabilidades e *hardening* de sistemas. Após uma varredura pelo sistema, são reportadas as falhas de configuração e o que precisa ser corrigido para proteger o sistema.

3 AMBIENTES VIRTUAIS

Os ambientes virtuais foram criados para atender uma demanda de execução de várias aplicações diferentes. Isso ocorreu devido a uma variedade de tipos de *hardwares* e sistemas operacionais, paralelo a tal fato, *hardwares* foram ficando mais baratos e o multiprocessamento de sistemas operacionais emergiu.

Com estas tecnologias, é possível proteger e isolar do sistema base que serve de prática para os seguintes cenários: execução de múltiplos sistemas operacionais, desenvolvimento, execução e teste de *softwares*, simulação de *hardwares*, construção de redes virtuais e proteção de um ambiente seguro e isolado (*sandbox*).

3.1 Virtualização

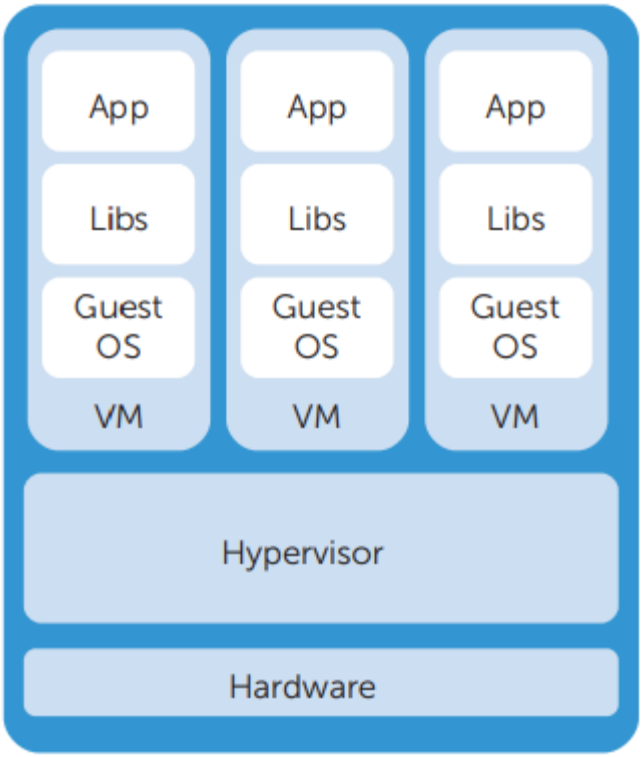
Virtualização é uma tecnologia que combina ou divide recursos computacionais para apresentar um ou mais ambientes operacionais usando metodologias como particionamento e agregação de *software* e *hardware*, parcial ou simulação de máquina completa, emulação, tempo compartilhado e muitos outros (CHIUEH; SUSANTA; BROOK, 2005).

O *software* capaz de realizar a virtualização é chamado de máquina virtual (VM), que roda a partir de uma camada capaz de criar múltiplas máquinas virtuais independentes e isoladas uma das outras. Essa camada de virtualização é chamada de hipervisor que se encontra entre a máquina virtual e a infraestrutura do sistema operacional.

O uso muito comum de uma VM é de hospedar sistemas operacionais, e para esse tipo de virtualização, é necessário um consumo considerável de *hardware*. Segundo Pahl (2015), as instâncias do sistema operacional na VM usam uma quantidade extensa de arquivos isolados no *host*, para armazenar todo o sistema de arquivo e tipicamente executar um único, grande processo no host.

No modelo apresentado na Figura 8 é possível ver que cada máquina virtual age de forma independente sobre a camada de hipervisor. Entretanto, cada aplicativo depende dos recursos, como bibliotecas, do sistema operacional virtualizado para conseguir executar, o que demanda muitos recursos da máquina hospedeira.

Figura 8 - Visualização do esquema da máquina virtual.



Fonte: (COMBE; MARTIN; PIETRO, 2013).

3.2 Conteinerização

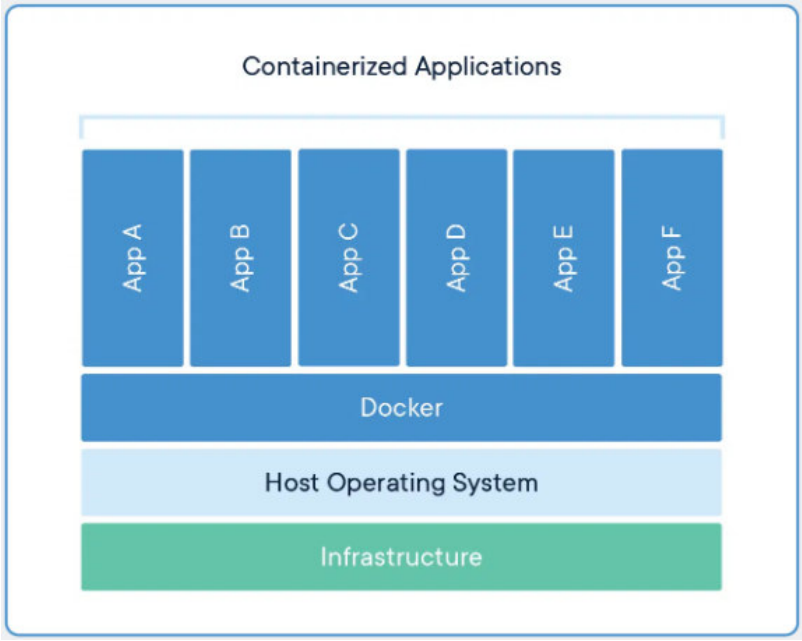
Outro tipo de virtualização que se popularizou muito nos últimos anos é a conteinerização. O objetivo é o mesmo das máquinas virtuais, entretanto com diferenças estruturalmente cruciais.

Um contêiner é uma unidade padrão de software que empacota um código e todas as suas dependências, para que uma aplicação possa executar rapidamente, de forma confiável, de um ambiente computacional para outro (DOCKER, 2013). Esta técnica surgiu como uma alternativa de virtualização mais rápida para o desenvolvimento de aplicações. A plataforma Docker estabeleceu o padrão da indústria de contêineres, por isso o termo *container* no contexto de computação é quase sempre associado a esse *software*.

Para rodar uma aplicação, é necessária uma imagem, que é um pacote de software, leve, autônomo, que inclui tudo o que uma aplicação precisa para executar: código, tempo de execução, ferramentas do sistema, bibliotecas do sistema e configurações (DOCKER, 2013). Portanto, os contêineres executam isolados do seu

ambiente, independentemente do sistema operacional da infraestrutura, como pode ser observado na Figura 9.

Figura 9 - Arquitetura das aplicações em contêineres.



Fonte: (DOCKER, 2013).

A conteinerização surgiu para se ter uma inicialização mais rápida das aplicações do que em uma máquina virtual. Múltiplos contêineres podem rodar no mesmo sistema operacional, as imagens são leves, consomem menos memória e armazenamento, as aplicações são mais práticas de serem compartilhadas pela nuvem e são altamente escaláveis. Enquanto as VMs precisam de uma cópia inteira do sistema operacional, o que atrasa sua inicialização, demanda mais recursos de memória e torna o processo de escalabilidade mais difícil à medida que as aplicações aumentam as funcionalidades.

3.2.1 Introdução ao Docker

A plataforma Docker surgiu em 2013 como um projeto de código aberto que permite executar aplicações em contêineres de forma leve e isolada. O Docker utiliza uma arquitetura cliente-servidor formada por três unidades: Docker *client*, Docker *daemon* e o Registro mostrados na Figura 10.

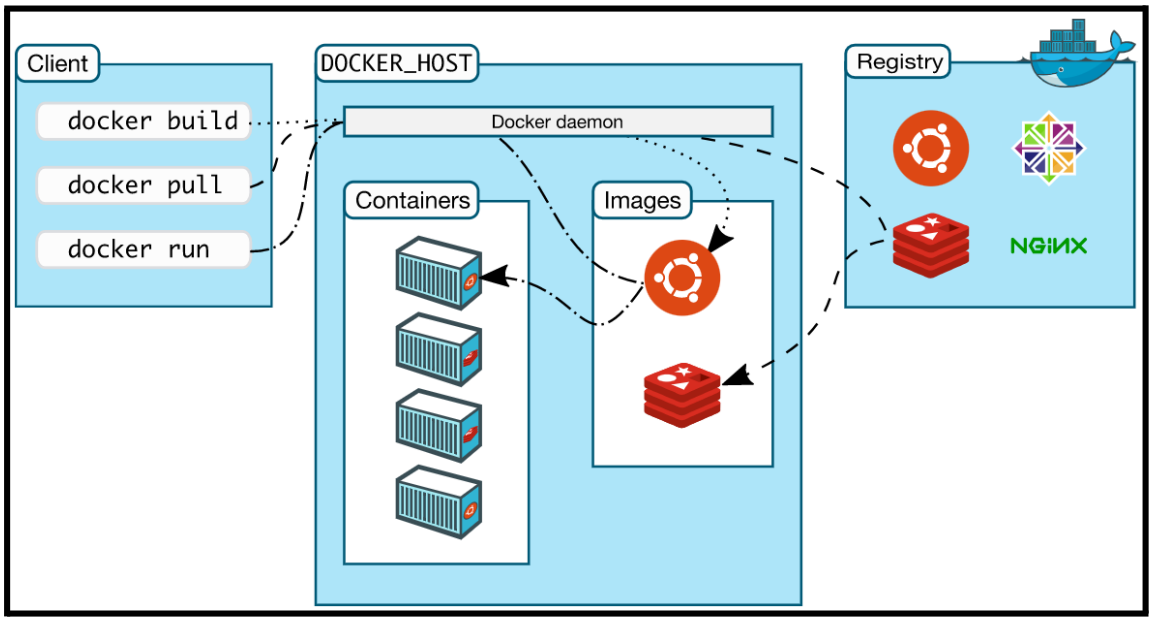
- Docker *client*: Modo como os usuários interagem com o Docker. São comunicados através de comandos como “docker run”, “docker build”, “docker

pull" ou "docker-compose" quando trabalhando com múltiplos contêineres. Esses comandos são enviados para o Docker *daemon* usando API do Docker.

- Docker *daemon*: Escuta requisições vindo da API do Docker e gerencia objetos como contêineres, imagens, redes e volumes (método para salvar dados persistentes armazenados no host).

- Registro: Um espaço que funciona como *Software as a Service* (SaaS) para baixar e compartilhar imagens Docker. A configuração padrão procura imagens no Docker *Hub*, um registro público que qualquer um pode usar, mas é possível baixar e enviar imagens de um registro privado.

Figura 10 - Arquitetura geral do Docker.

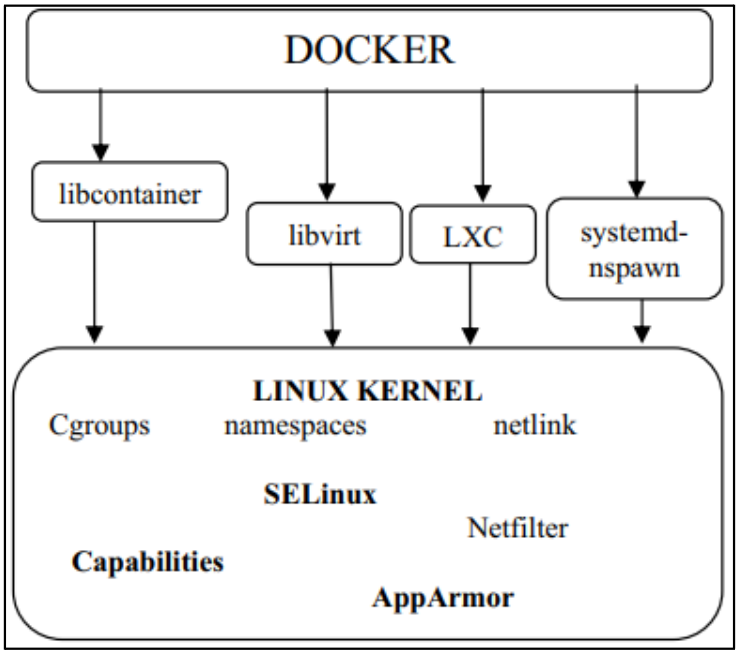


Fonte: (DOCKER, 2013).

Docker foi inicialmente designado para utilizar diferentes interfaces para acessar virtualização caracterizadas pelo *kernel* do Linux. Na versão 0.9, em março de 2014, o Docker criou sua própria biblioteca de execução, a *libcontainer*, escrita na linguagem Go (YASRAB, 2021).

A partir destes recursos do Linux, o Docker consegue isolar os contêineres através de *namespaces*, onde cada contêiner tem seu espaço de execução e *cgroups* (grupos de controle), para garantir que nenhum contêiner consuma todo o recurso do computador. O Docker é feito num sistema unificado de arquivo, mostrado na Figura 11, compilando vários ramos e sistemas de arquivo em um único.

Figura 11 - Utilização dos recursos de virtualização do *kernel* no Docker.



Fonte: (RAJ et al., 2016).

Na Figura 11 é possível observar ainda que o Docker utiliza os módulos de segurança do *kernel* (SELinux e AppArmor) e o *firewall* (Netfilter).

3.2.2 Comandos básicos do Docker

O cliente Docker utiliza comandos para gerenciar objetos, podendo enviar os comandos localmente no *host* ou de forma remota. Os comandos apresentados a seguir servem de base para realizar qualquer tipo de operação nos objetos:

- *docker run*: Comando para executar um Docker *container* (ver Figura 12). É necessário especificar a imagem, as opções incluem rodar o processo em background, definir CPU e memória, configurações de portas de rede, identificação de contêiner etc.

Figura 12 - Estrutura do comando “docker run”.

```
$ docker run [OPTIONS] IMAGE[:TAG|@DIGEST] [COMMAND] [ARG...]
```

Fonte: (DOCKER, 2013).

- *docker build*: Utilizado para montar uma imagem a partir de um Dockerfile (um arquivo que contém os comandos para montar uma imagem). Com o “docker build” é possível executar vários comandos em sequência a partir do Dockerfile.

- *docker-compose up*: Comando para criar e rodar vários contêineres a partir das especificações do arquivo docker-compose.yml, em linguagem YAML. Este arquivo contém as configurações para rodar um ou mais contêineres, altamente recomendável para criar um sistema em que dois ou mais contêineres interajam entre si. Na Figura 13, é possível ver um serviço de webapp que contém especificações para rodar a aplicação: imagem, mapeamento de portas e volume.

Figura 13 - Exemplo de um arquivo docker-compose.yml.

Fonte: (DOCKER, 2013).

Observa-se que o Docker oferece uma virtualização altamente customizável para rodar diversos tipos de aplicações. Portanto, é necessário que o desenvolvedor entenda os requisitos para subir um contêiner, restringindo o máximo possível sua capacidade, para que não se abra uma possível vulnerabilidade de uma utilização desnecessária de recurso.

3.3 Falhas de Segurança em Docker

Contêineres conseguem diretamente se comunicar com o *kernel* do *host*, permitindo um atacante economizar uma grande quantidade de esforço para invadir o sistema hospedeiro. Isso substancialmente aumenta uma preocupação de segurança sobre contêineres. (CHELLADHURAI; CHELLIAH; KUMAR, 2016). A partir desse ponto, é possível gerar uma abertura a ataques, entretanto há outras preocupações que precisam ser levadas em consideração, dentre elas são:

- Imagens envenenadas: Docker *Hub* fornece uma conexão criptografada e as imagens são digitalmente assinadas, entretanto é importante sempre mantê-las na sua última versão e somente baixar imagens confiáveis.

- Escape de contêiner: O atacante pode escapar de um contêiner e acessar a máquina *host* através de um diretório compartilhado.

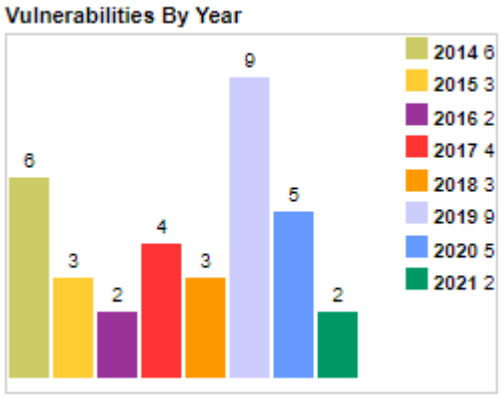
- Privilégios *root*: Para rodar o Docker *daemon* é necessário privilégios de root, se mal configurado, o atacante pode se apropriar desse processo e escalonar privilégios.

- Ataques DoS: O objetivo desse tipo de ataque é consumir todo recurso da máquina, para tanto, o atacante se abdica de um contêiner para esgotar os recursos de outros contêineres e da máquina *host*.

- Ataques *Man in the Middle*: Por padrão, o Docker não possui conexão criptografada, o que deixa uma abertura para um atacante se infiltrar numa conexão e ler dados sensíveis ou alterar informação.

Segundo CVE (2021), na plataforma Docker foram identificadas somente 34 vulnerabilidades desde 2014, mostradas na Figura 14. É um excelente resultado, mostrando que os desenvolvedores da plataforma têm cuidado em consertar suas vulnerabilidades para manter um ambiente mais seguro possível.

Figura 14 - Quantidade de vulnerabilidades conhecidas do Docker por ano.



Fonte: (CVE, 2021).

O fato de o Docker ser considerado uma plataforma segura, não significa que sua instalação padrão não contém riscos. Um dos fatores de risco continua sendo a configuração insegura, o administrador de redes deve ter atenção para se proteger das ameaças. Felizmente, o Docker possui uma boa documentação de segurança e há vários mecanismos de blindagem que a plataforma suporta.

3.4 Docker Hardening

As medidas de *hardening* tem o propósito de evitar, proteger e diminuir os danos causados pelos ataques. São configurações que passam pelo *host*, Docker *daemon*, imagens e contêineres. A maioria delas não vem configurada por padrão, algumas precisam ser feitas manualmente ou com ajuda de *softwares* oficiais do Docker ou de terceiros. A blindagem do Docker é composta pelas seguintes técnicas:

- Auditar arquivos do Docker *host*: Utilizar ferramentas de auditoria para manter registrados os acessos de todos os arquivos (binários, bibliotecas, configurações ou *sockets*) do Docker, para detectar acessos indesejados ou origem de ataques.

- Imagens assinadas digitalmente e verificadas: Por padrão, o Docker busca imagens do Docker *Hub*, que possui imagens assinadas digitalmente garantindo a integridade dela. Para o desenvolvedor, além de buscar imagens assinadas, é importante escanear imagens existentes para verificar vulnerabilidades e utilizar as versões mais recentes possíveis.

- Impedir uso excessivo e privilegiado: Garantir que os contêineres tenham o mínimo de privilégio possível para realizar suas tarefas, que não comprometam o sistema *host* ou outros contêineres rodando.

- Limitar recursos do contêiner (memória e CPU): Configurar os recursos de cada contêiner para que não esgote os recursos do *host* e desequilibre prioridades entre outros contêineres.

- Restringir a rede: Evitar na medida do possível a intercomunicação entre contêineres, impedir o acesso de fora da rede e habilitar o uso de criptografia nas comunicações com o Docker *daemon*.

- Módulos de segurança (Apparmor / SELinux): Habilitar módulos de segurança que implementam controle de acesso obrigatório, adicionando mais uma camada de segurança.

No próximo capítulo é apresentado como são aplicadas essas técnicas de forma prática, incluindo comandos do Docker, comandos Bash do Linux, manuseio de arquivos, utilização de programas, dentre outros.

4 ESTUDO DE CASO: SERVIDOR WEB COM BANCO DE DADOS

Aplicando as técnicas de Docker *hardening* na prática, foi instalado o Docker numa máquina Ubuntu 20.04 64-bit e implementado um sistema de contêineres contendo um servidor *web* com banco de dados. Para testar o Docker *hardening*, foi utilizado a ferramenta *Docker Bench for Security* (DOCKER, 2015), um *script* que verifica dezenas das práticas de segurança no ambiente Docker.

Ao longo deste capítulo serão mostrados os comandos necessários para aplicação de técnicas e no final, um comparativo entre um sistema blindado e um sistema desprotegido.

4.1 Instalação do Docker e Implantação de Contêineres

A Docker *Engine* foi instalada na máquina seguindo a instalação de repositório. Após a instalação, foi realizado um teste para verificar se a Docker *Engine* foi instalada corretamente utilizando o comando "docker run" para rodar o contêiner com a imagem "*hello-world*" (ver Figura 15).

Figura 15 - Verificação básica de instalação.

```
ulisses@pc:~$ sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
2db29710123e: Pull complete
Digest: sha256:80f31da1ac7b312ba29d65080fddf797dd76acfb870e677f390d5acba9741b17
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash
```

Fonte: O autor.

Na Figura 16, é disposto o arquivo "docker-compose.yml" contendo as especificações básicas para rodar um sistema de contêineres de servidor *web* com banco de dados. Não são necessários mais comandos que estes para subir as aplicações, entretanto elas não possuem nenhuma restrição.

Foram utilizadas as imagens Php-apache, MySql e Adminer para compor o sistema. A imagem Php-apache compõe o servidor *web*, a imagem MySql para o banco de dados e imagem Adminer para gerenciamento do banco.

Figura 16 - Primeira versão do arquivo docker-compose.yml.

```
version: '2.4'
services:
  php:
    image: php:7.4-apache
    command: docker-php-ext-install mysqli
    ports:
      - 80:80
    volumes:
      - ./src:/var/www/html/
  db:
    image: mysql
    command: --default-authentication-plugin=mysql_native_password
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD:
    volumes:
      - mysql-data:/var/lib/mysql
  adminer:
    image: adminer
    restart: always
    ports:
      - 8080:8080
volumes:
  mysql-data:
```

Fonte: O autor.

É necessário primeiro informar a versão do comando docker-compose (p.ex 2.4) utilizada, dependendo da versão, a sintaxe pode ser diferente. Em seguida é informado os serviços e volumes, onde os dados são salvos permanentemente. Em cada serviço é especificado a imagem, o comando executado dentro do container para inicializar a aplicação, as portas de conexão mapeadas do host e do contêiner, volumes e variáveis de ambiente.

Após a finalização das especificações do arquivo.yml, é necessário somente o comando “docker-compose up” na linha de comando para rodar as aplicações. Na primeira execução as imagens serão buscadas localmente e se não encontradas, serão baixadas do Docker *Hub* e em seguida os contêineres são executados em conjunto.

4.2 Primeira Varredura de Segurança

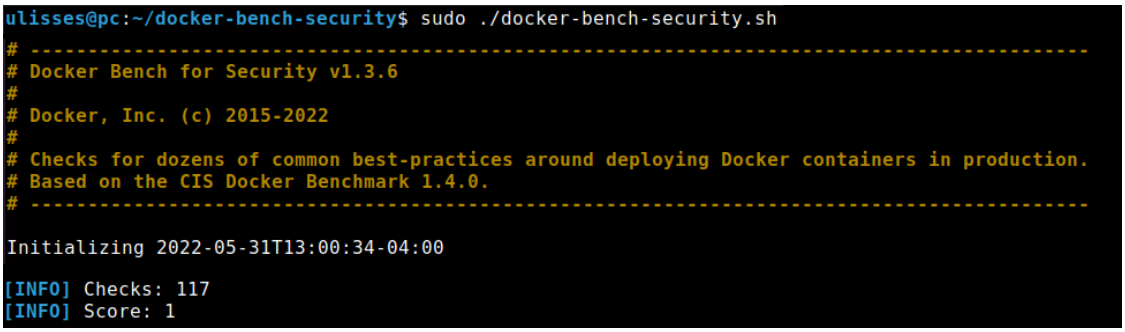
Após os procedimentos de instalação e de subir as aplicações, foi executado o *script* “docker-bench-security.sh” para verificar a segurança do sistema. O programa verifica as configurações de instalação do *host*, modo *swarm*, imagens e contêineres em execução. O *script* também lista recomendações de segurança para os determinados componentes, além de categorizar cada item de checagem em rótulos.

A ferramenta contém quatro tipos de rótulos: INFO, NOTE, WARN E PASS. Somente os dois últimos influenciam na pontuação total.

- INFO: Representado pela cor azul, contém informações coletadas automaticamente pela ferramenta que mostram nome das seções, arquivos não encontrados e ações optativas que podem ser melhoradas.
- NOTE: Representado pela cor amarela, contém recomendações desejáveis de segurança que precisam ser feitas manualmente e que não estão no alcance da ferramenta (p.ex, senhas fortes, armazenamento de segredos nas imagens e certificar que a máquina hospedeira está protegida).
- WARN: Destacado em vermelho, apresenta as falhas de segurança detectadas e pontuam negativamente na pontuação total do *benchmark*.
- PASS: Destacado em verde, apresenta as falhas corrigidas e adiciona um ponto positivo na pontuação total.

Na Figura 17 é mostrado a pontuação da primeira varredura feita pela ferramenta do *benchmark*. A pontuação indica uma referência da quantidade de aberturas encontradas sobre o Docker configurado, porém, além de ponderar as aberturas, é preciso compreender a gravidade de cada vulnerabilidade e corrigir as falhas expostas.

Figura 17 - Resultado da primeira varredura.



Fonte: O autor.

Percebe-se que no primeiro resultado, pelas configurações padrões do Docker *host*, nos contêineres e nas imagens, é apresentado um resultado 1 de 117 que indica a abertura de muitas brechas.

Foram agrupados os resultados de cada seção na tabela 2, cada uma contendo suas respectivas notas de acordo com cada rótulo. Observação sobre o modo *swarm* que foi ignorado, pois pontuou em todas as checagens, sendo esta seção segura por padrão.

Tabela 2 - Resultado de cada seção a partir dos rótulos

Primeiro teste				
	Rótulos			
Seções	INFO	NOTE	WARN	PASS
Host	6	1	15	1
Docker *daemon*	15	1	7	19
Imagens	1	7	3	1
Contêineres	2	2	11	16

Fonte: O autor.

4.3 Implementação da Blindagem

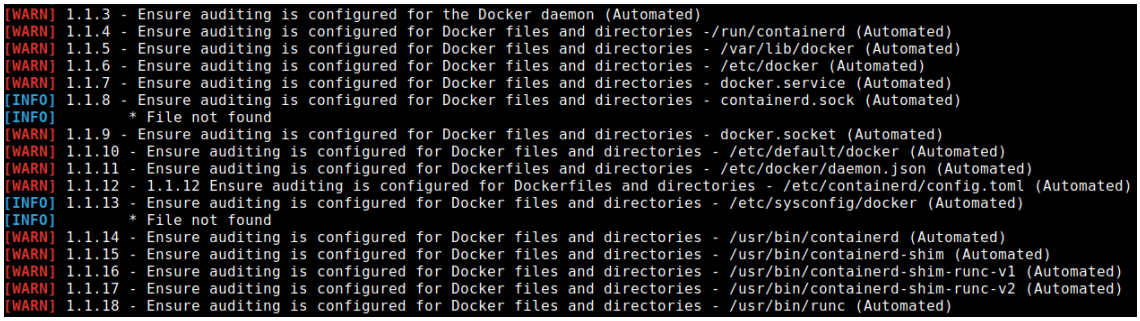
Diante da detecção de falhas, foram aplicadas as técnicas de Docker *hardening* para correção destas falhas, proteção do sistema contra ataques e manter um ambiente mais rígido. As técnicas implicam em configurações dos arquivos de imagens, contêineres e do Docker *daemon*. Nas seções a seguir, são apresentadas estas configurações de forma prática demonstrando os comandos e ferramentas necessárias.

4.3.1 Auditoria dos Arquivos Docker

Manter os registros de acesso aos arquivos não conduz uma camada de proteção, entretanto é uma boa prática de segurança para detectar acessos indesejados e origens de ataques, facilitando na remediação de ameaças futuras.

Uma das checagens do *Docker Bench for Security* é a configuração do host, que detecta os arquivos que precisam ser auditados (manter os acessos registrados). Na Figura 18 é mostrado a seção de configuração do *host* dos arquivos do Docker (binários, diretórios e serviços) que precisam ser auditados.

Figura 18 - Resultado do *script* do *benchmark* na seção de configuração do *host*.



Fonte: O autor.

Para auditar esses arquivos, foi utilizado uma ferramenta do sistema de auditoria do Linux chamada Auditd. É possível ser acessada pelo comando "audictl*"* especificado na Figura 19, para criação de regras sobre os artefatos especificados.

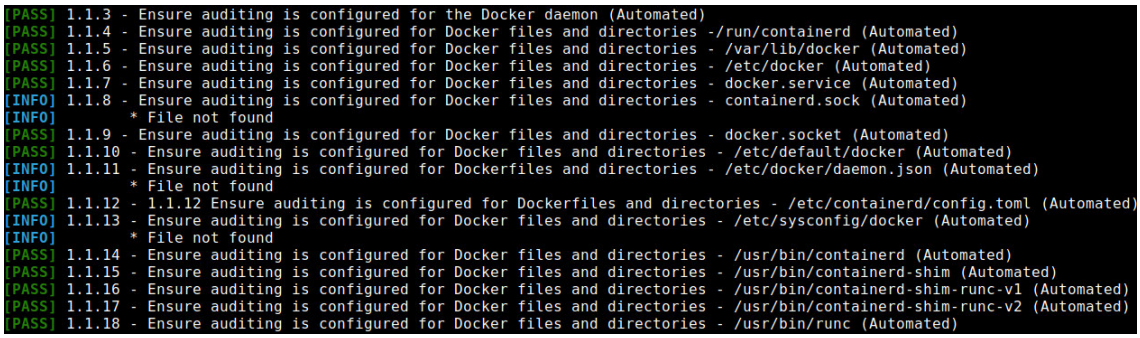
Figura 19 - Comando do "auditcl" para auditar os arquivos do Docker.

```
sudo auditctl -w <path to artifact> -k docker
```

Fonte: (AHMED, 2021).

O argumento "-w" serve para especificar um arquivo a ser vigiado, e o argumento "-k" para especificar uma palavra chave (nesse caso o Docker), que pode ser aplicado para diferentes regras. Essas regras são gravadas no arquivo */etc/audit/audit.rules* e após a aplicação destas para arquivo do Docker, todos os acessos de cada administrador, usuário ou cliente são registrados. Na Figura 20 é visto o resultado novamente do *script* do Docker com as regras aplicadas para cada artefato.

Figura 20 - Resultado da seção do host após aplicação de regras pelo Auditd.



Fonte: O autor.

4.3.2 Verificação Imagens

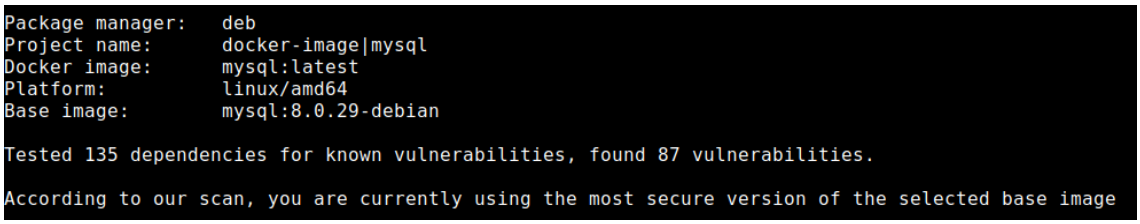
É possível verificar a integridade das imagens com a ferramenta *Docker Content Trust* (DCT), que providencia a habilidade de usar assinaturas digitais para enviar e receber dados remotamente de registros (DOCKER, 2013).

Imagens que são baixadas diretamente do Docker *Hub* já possuem as assinaturas digitais, no entanto, algumas imagens podem ser criadas e armazenadas em repositórios privados (p.ex dentro da rede de uma empresa), e por isso é importante usar sempre o DCT para verificar a assinatura de imagens.

A ferramenta DCT garante a integridade das imagens, possibilitando ver assinatura dos autores delas, porém, para verificar vulnerabilidades, o Docker possui uma outra ferramenta. O comando *docker scan* lista as CVEs detectadas, junto com as severidades de cada uma delas e recomendações para remediar a vulnerabilidade.

Outro fator importante desta ferramenta, é mostrar se a imagem está atualizada, indicando se a imagem selecionada está ou não na sua versão mais segura. Na Figura 21 é mostrado um exemplo de uma varredura utilizando uma das imagens usadas neste trabalho.

Figura 21 - Exemplo do comando "*docker scan mysql:latest*".



Fonte: O autor.

No desenvolvimento de uma aplicação, é interessante realizar este tipo de varredura antes de iniciar qualquer implementação, além de repetir este procedimento periodicamente para avaliar a segurança da aplicação, podendo ser necessário aplicar possíveis atualizações de imagens.

4.3.3 Impedir Uso Excessivo e Privilegiado

Como a virtualização de contêineres não possui isolamento total, um dos principais fatores para prevenir que um atacante se aproprie de um contêiner e invadir a máquina hospedeira é evitar que os processos sejam executados como *root*. Além de privilégios de usuários, é importante manter as capacidades do *kernel* (leitura, escrita e execução) mínimas para funcionamento de uma aplicação. Portanto é necessário proteger as superfícies de ataques das seguintes maneiras:

- Docker *daemon*: Para executar o Docker daemon, é necessário privilégios de *root* no Linux hospedeiro ou na instalação do *kernel* (RAJ et al., 2016). Contudo, os contêineres podem ser isolados com *namespace* e rodar como usuários não privilegiados usando a opção de inicialização do Docker daemon "--userns-remap=default".

- Dockerfile: É possível criar um arquivo Dockerfile para criar uma imagem personalizada a partir de uma imagem existente. Por padrão, quando um *container* é criado, este é executado como *root* dentro dele. Mas com o Dockerfile é possível executar alguns comandos para rodar um contêiner como usuário qualquer ao invés de *root* através de: "RUN groupadd -r newuser && useradd -r -g newuser newuser" e "USER newuser".

- Contêineres: Com a opção "--no-new-privileges" na execução de um Docker *container*, garante que os processos dentro do contêiner não irão ganhar privilégios adicionais. As capacidades do *kernel* podem ser removidas usando o "cap_drop=all" e adicionar somente as capacidades necessárias com "cap_add". Para restringir ainda mais os privilégios, a opção de rodar com "read_only" garante que o sistema de arquivos permite somente leituras.

Agregando estes comandos juntos, se torna bem mais difícil um atacante escalonar privilégios pois, além de restringir os privilégios dentro do contêiner, caso ele

ainda consiga escapar para máquina hospedeira, os privilégios ainda serão de um usuário comum que terá muitas de suas ações bloqueadas.

4.3.4 Limitar Recursos de Contêineres

Um dos princípios de segurança é a disponibilidade, os recursos do sistema não podem estar inacessíveis. Em um sistema de contêineres, é crucial que cada contêiner somente utilize recursos necessários para execução e não torne o sistema inutilizável.

Por padrão, o Docker não possui restrições de recursos e os contêineres podem utilizar até todo recurso da máquina hospedeira, abrindo uma brecha para ataques DoS que podem inutilizar um sistema a partir do esgotamento de recursos. Contudo, o Docker permite limitar a quantidade de recursos como memória, CPU, descritores de arquivos, processos e reinicializações para cada contêiner.

Com a opção de “cpu_percent”, o container é restrito a somente usar uma porcentagem total da CPU. Para limitar a memória, utiliza-se “mem_limit” e “mem_reservation”, sendo o segundo ativado como um limite mínimo caso a máquina hospedeira detecte pouca memória disponível.

Outra maneira de restringir recursos é utilizando “pids_limits”, para que a aplicação rodando tenha limite de quantos processos criar. Um tipo comum de ataque DoS é chamado de *fork bomb*, quando um processo cria vários processos continuamente até esgotar os recursos do sistema. A visualização desses recursos pode ser visto com o comando “docker stats”, apresentado na Figura 22, com ele é possível ter uma referência de quantos recursos estão sendo utilizados no processo de execução.

Figura 22 - Visualização do comando *docker stats* dos *containers* em execução.

```
CONTAINER ID   NAME                CPU %     MEM USAGE / LIMIT   MEM %     NET I/O       BLOCK I/O       PIDS
973668eb8900   hardened_php_1      0.01%     8.223MiB / 10MiB    82.23%    4kB / 0B      110MB / 0B      6
b20fbf290afd   hardened_db_1       0.25%     353.1MiB / 380MiB   92.91%    6.49kB / 0B   79MB / 13.4MB   38
824318aa2dc9   hardened_adminer_1  0.01%     6.867MiB / 10MiB    68.67%    7.14kB / 0B   22.8MB / 0B     1
```

Fonte: O autor.

Para ter um controle sobre o estado do contêiner, utiliza-se a opção de “healthcheck” para detectar se um contêiner está funcionando ou se está travado em um loop infinito recusando conexões. Uma política usada com esta opção é “restart: on_failure:5” que reinicia automaticamente o contêiner no máximo cinco vezes (limite recomendado pelo Docker) depois da aplicação ter terminado com estado de erro. Após atingir esse limite, o contêiner somente é iniciado de forma manual.

4.3.5 Restringir a Rede

São práticas de restrição de uma rede de computadores: limitar o fluxo de informação a somente usuários incluídos na rede, permissão à leitura das mensagens para quem possui as chaves privadas, segmentar a rede em subredes e definir o sentido do fluxo de informação para remetente e destinatário.

Nesse contexto, o objetivo é proteger a comunicação entre o cliente e o *host*, entre o *host* e os contêineres e a intercomunicação de contêineres. Os passos para restringir uma rede usando Docker são:

- Habilitar TLS (*Transport Layer Security*): Gerar os certificados assinados, chaves públicas, privadas e habilitar comunicação utilizando TLS usando "--tlsverify" na inicialização do Docker *daemon*.

- Desativar a intercomunicação de contêineres: Comunicação entre processos pode gerar uma brecha de segurança, processos que não precisam se comunicar entre si, não devem ter acesso um ao outro. Com a opção "--icc=false" na inicialização do Docker *daemon*, é desabilitado a comunicação entre contêineres. Caso os contêineres precisem dessa comunicação, informa-se os *links* necessários dos serviços para cada contêiner durante o tempo de execução.

- Especificar a interface de conexão do *host*: Rodar os contêineres especificando o IP, a porta mapeada no host e no contêiner.

- Abrir portas de conexão somente necessárias: Manter conexões abertas somente de serviços que utilizem elas.

4.3.6 Habilitar Módulos de Segurança

O módulo de segurança do *kernel* é uma das funcionalidades mais importantes que o Docker utiliza, integrando uma barreira protetora a mais no sistema. Implementam o Controle de Acesso Obrigatório (MAC) limitando capacidades, chamadas de sistemas, restringindo acesso à diretórios e diminuindo privilégios através de um perfil de segurança que contém as políticas de MAC.

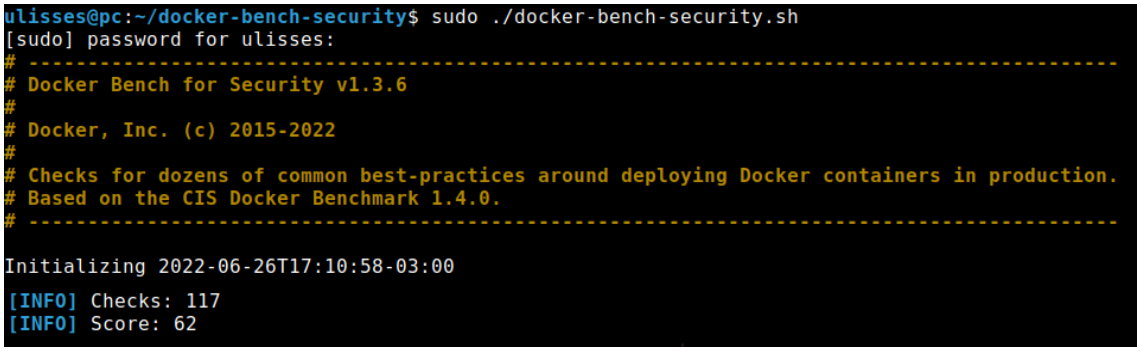
O Docker, por padrão, carrega perfis pré definidos na execução de contêineres através dos módulos Seccomp e AppArmor. Os perfis padrões são relativamente seguros enquanto fornecem uma boa flexibilidade das aplicações e portanto, não devem

ser desativados. Contudo, perfis customizados podem ser sobrepostos caso seja desejado uma segurança mais personalizada.

4.4 Panorama Final após a Blindagem

Após a aplicação da blindagem, foi executado novamente o script "*docker-bench-security.sh*" para uma nova varredura, mostrado na Figura 23. Percebe-se que a pontuação total melhorou drasticamente, aumentando em 61 pontos e pontuando 62 de 117, indicando que muitas brechas foram corrigidas.

Figura 23 - Varredura final do *benchmark*.



Fonte: O autor.

Na tabela 3, é mostrado o número de ocorrências de acordo com cada rótulo. Percebe-se uma grande melhora nas pontuações do rótulo PASS, o que indica que muitas aberturas encontradas foram corrigidas.

Tabela 3 - Ocorrências de acordo com cada rótulo.

Teste após aplicação da blindagem				
	Rótulos			
Seções	INFO	NOTE	WARN	PASS
Host	6	1	1	14
Docker *daemon*	11	1	4	28
Imagens	1	7	1	3
Contêineres	2	2	2	25

Fonte: O autor.

O arquivo de configuração de inicialização do *daemon* é mostrado na Figura 24. Nele contém configurações de TLS com os certificados rodando na porta apropriada 2376, *namespace* habilitado, comunicações entre contêineres desabilitados e bloqueio de execução de contêineres privilegiados.

Figura 24 - Arquivo de inicialização do Docker *daemon*.

```
[Service]
ExecStart=
ExecStart=/usr/bin/dockerd -D -H unix:///var/run/docker.sock --tlsverify --tl
scert=/home/ulisses/.docker/server-cert.pem --tlscacert=/home/ulisses/.docker
/ca.pem --tlskey=/home/ulisses/.docker/server-key.pem --userns-remap="default
" --icc="false" --no-new-privileges -H tcp://0.0.0.0:2376
```

Fonte: O autor.

Com isso, cada serviço (php, mysql e adminer) contido no arquivo "docker-compose.yml" teve alterações majoritárias. Na Figura 25 se apresenta a configuração YAML do servidor php-apache, junto com alguns comentários para cada comando.

Figura 25 - Serviço do servidor web no arquivo "docker.compose.yml".

```
php:
  links:  #Definir o serviço que precisa de conexão (banco de dados)
    - "db"
  build:
    context: .
    dockerfile: Dockerfile
  ports:
    - 127.0.0.1:80:80 #Especificar interface de host
  read_only: true
  tmpfs:
    - /tmp
    - /run/apache2
    - /run/lock
  volumes:
    - ./src:/var/www/html/
  cap_drop: #Controlar capacidades
    - all
  cap_add:
    - NET_BIND_SERVICE
  restart: on-failure:5
  cpu_percent: 20 #Restringir CPU
  mem_limit: 10m #Restringir memória
  mem_reservation: 6m
  pids_limit: 100 #Restringir PID
  healthcheck: #Verificar status do container
    test: ["CMD", "curl", "-f", "http://localhost"]
    interval: 1m30s
    timeout: 10s
    retries: 3
    start_period: 40s
  security_opt: #Garantir que nao ganhe privilegios
    - no-new-privileges
```

Fonte: O autor.

Percebe-se que este contêiner tem restrições de leitura, conexões, capacidades, recursos (memória, CPU e processos), privilégios e ainda uma checagem de estado.

Estes padrões seguem para outros contêineres (MySQL e Adminer) neste arquivo e cada um deles conta com uma imagem personalizada num Dockerfile especificado (visto na Figura 26).

Figura 26 - Dockerfile personalizado da imagem php-apache.

```
FROM php:7.4-apache
RUN groupadd -r ulisses && useradd -r -g ulisses ulisses
RUN docker-php-ext-install mysqli
USER ulisses
```

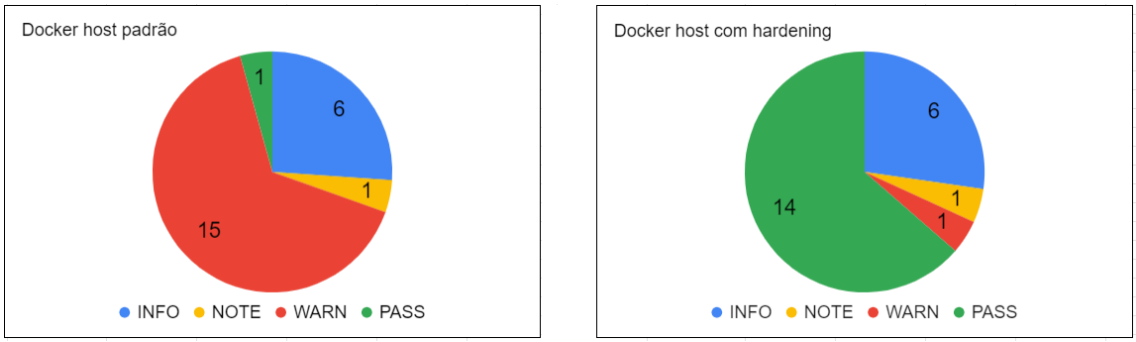
Fonte: O autor.

Neste Dockerfile, contém os comandos necessários para rodar a aplicação (e.g instalar mysqli), contudo, os comandos mais importantes desta imagem são para rodar como um usuário não privilegiado (e.g usuário ulisses) dentro do contêiner.

4.5 Resultados e Comparativos

Para ter uma perspectiva visual da comparação entre um sistema Docker com configuração padrão e um sistema blindado, foram gerados gráficos a partir das tabelas 2 e 3. As próximas figuras apresentam o resultado de cada seção, em que a primeira varredura, feita do sistema sem blindagem, se encontra nos gráficos da esquerda, enquanto a última varredura, após aplicada a blindagem, na direita. Os gráficos em pizza estão divididos em rótulos com cada cor representando seu respectivo rótulo.

Figura 27 - Comparação entre os testes do Docker *host*.



Fonte: O autor.

Na Figura 27, observa-se que havia muitos arquivos do Docker que precisavam ser auditados. Na seção do Docker *host*, a ação de segurança aplicada foi criar regras para tais arquivos.

Figura 28 - Comparação entre os testes de Docker *daemon*.

Fonte: O autor.

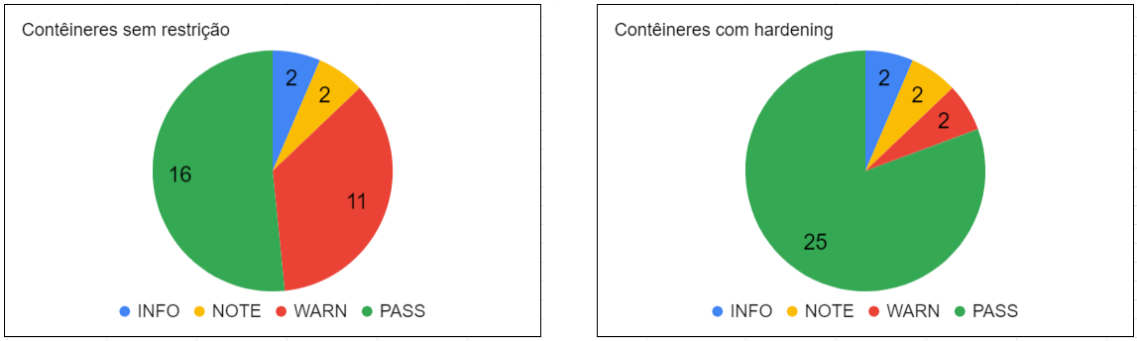
As ações aplicadas no Docker *daemon* se basearam em habilitar TLS, restringir o tráfego de rede entre contêineres e bloquear a opção de privilégios nos contêineres. Percebe-se principalmente a diminuição de ocorrências do tipo INFO (ver Figura 28), pelo fato da criação dos arquivos de certificados, chaves públicas e privadas na comunicação criptografada.

Figura 29 - Comparação entre os testes da seção de imagens.

Fonte: O autor

As aplicações de *hardening* nas imagens consistiram em criar um usuário no Dockerfile e habilitar o *Docker Content Trust*. Percebe-se na Figura 29, que a maioria das recomendações são do tipo NOTE, onde se incluem checagens que o *script* não consegue mensurar e necessitam de revisão manual dos desenvolvedores. As recomendações do tipo NOTE encontradas nessa seção são: usar somente imagens confiáveis, manter imagens atualizadas, não instalar pacotes desnecessários e não incluir conteúdos sensíveis nos Dockerfiles.

Figura 30 - Comparação entre os testes dos contêineres.

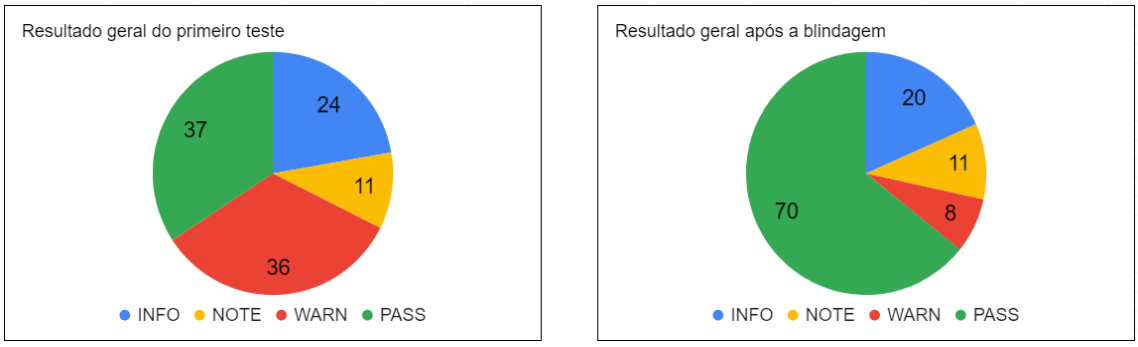


Fonte: O autor.

A maioria das aberturas encontradas na seção dos contêineres foram corrigidas através de restrições incluídas no arquivo “docker-compose.yml”, aplicadas no tempo de execução. Os contêineres são uma parte crítica da superfície de ataque, pois nesse caso, incluem portas abertas à internet possibilitando acesso externo. É possível observar na Figura 30 que com as técnicas de *hardening*, poucas aberturas ficaram expostas.

Por fim, apresenta-se o gráfico geral, percorrendo por todas as seções do teste, como observado na Figura 31.

Figura 31 - Resultado geral de antes e depois da blindagem.



Fonte: O autor

É notável que a configuração padrão do Docker possui uma quantidade moderada de brechas abertas. Observa-se um aumento de pontuação drástica após aplicação das práticas de segurança, indicando que o sistema está mais rígido, consistente e resistente à ataques.

Diante da exibição dos gráficos, algumas observações precisam ser ponderadas:

- Foi observado uma diminuição leve de ocorrências do rótulo INFO, indicando a criação de certificados, arquivos de configurações e permissões desses arquivos principalmente na seção do Docker *daemon*.

- A quantidade de ocorrência do rótulo NOTE se mantém inalterado, pois este rótulo serve apenas para recomendações de segurança que o *script* não consegue alcançar. O administrador tem que configurar e verificar manualmente estas recomendações.

- As ocorrências de WARN foram minimizadas ao máximo dentro do possível, algumas delas não puderam ser contidas porque iriam comprometer o funcionamento do sistema de contêineres ou não estavam no escopo do estudo.

- Cada abertura está associada a uma vulnerabilidade, que pode ser explorada por ataques e cada ataque gera um tipo de consequência. A ferramenta *Docker Bench for Security* dá uma referência da quantidade de aberturas expostas, mas cabe ao administrador de redes corrigi-las e averiguar a consequência de cada uma delas.

Considerando os resultados da aplicação das técnicas de Docker *hardening* obtidos a partir de uma ferramenta de segurança oficial do Docker, conclui-se que o sistema como um todo apresenta menos inconsistência, se edifica sobre princípios de segurança da informação, possui acesso mais rígido e está menos propenso a sofrer danos de ataques.

As aberturas ainda mantidas foram:

- Acesso remoto e centralizado desconfigurado: Se tratando do teste somente usar a máquina local, esta opção foi ignorada. Em uma rede que múltiplas máquinas tem acesso ao Docker, esta opção deveria estar configurada

- Instruções de "*healthcheck*" nas imagens: Esta opção foi configurada somente no tempo de execução dos contêineres.

- Opção *live restore* desabilitada: Quando esta opção é habilitada, ela permite que os contêineres continuem rodando mesmo após o encerramento do processo do Docker *daemon*. Por motivos de garantir que nenhum contêiner estivesse

consumindo recursos da máquina no processo de configuração do *daemon*, esta opção foi ignorada.

- Uso de portas privilegiadas: A porta 80 foi utilizada para a página *web* e mantida pois, esta porta é usada especialmente para o protocolo HTTP, caso fosse mudada, a aplicação teria de rodar fora do contêiner, acarretando em uma exposição desnecessária.

Na tabela 4, é apresentado um esquema entre ataque *versus* defesa sobre o Docker, citando as vulnerabilidades expostas, os ataques a partir dessas vulnerabilidades, a consequência desses ataques e a mitigação dessas vulnerabilidades.

Tabela 4 - Esquema de Ataque x Defesa no Docker.

Vulnerabilidade	Ataque	Consequência	Correção no Docker
Recursos ilimitados	DdoS	Sistema inutilizável	Limitar recursos: Memória, CPU, processos
Imagens infectadas	Malware	Infecção de vírus na máquina hospedeira	Inspecionar através do DCT
Imagens desatualizadas	*Zero-day exploit*	Exposição do sistema de forma crítica	Realizar varredura de imagens
Conexões não criptografadas	*Man-in-the-middle*	Roubo de informações sensíveis	Habilitar TLS
Compartilhamento de recursos com a máquina hospedeira	Escalonamento de privilégios	Controle do atacante sobre a máquina hospedeira	Habilitar *user namespace*, não rodar processos como *root* e reduzir privilégios dos contêineres

Fonte: O autor.

A tabela exibe um resumo do que foi apresentado neste trabalho a partir de segurança de redes sobre Docker *hardening*. Cada tipo de ataque possui sua finalidade e podem causar danos críticos ao sistema, felizmente, os desenvolvedores do Docker tiveram certo cuidado ao criar uma plataforma que compartilhasse recursos com a máquina hospedeira, dando o suporte necessário para proteção do sistema.

5 CONCLUSÃO

Este trabalho apresentou técnicas de blindagem de sistemas virtuais baseados em contêineres, especificamente sobre a plataforma Docker. Para fundamentação do trabalho, foi discorrido sobre os princípios de segurança da informação, navegando entre sistemas operacionais e redes de computadores, associando medidas de defesa com técnicas de *hardening* no sistema operacional Linux. Foi introduzido o conceito de contêineres junto com a plataforma Docker, mostrando sua arquitetura, comunicação entre componentes e objetos, comandos básicos e preocupações de segurança. Por fim, foi apresentado um estudo de caso, utilizando como avaliação a ferramenta *Docker Bench for Security*, mostrando comparações de um sistema padrão e um blindado.

Percebe-se que o Docker possui algumas vulnerabilidades herdadas da conteinerização e, portanto, é tarefa do administrador conhecer os componentes da plataforma e restringir o acesso ao máximo sem comprometer o funcionamento dos serviços. Qualquer que seja a plataforma de desenvolvimento escolhida, independente das vulnerabilidades conhecidas, o fator humano ainda é crucial para a segurança do sistema, felizmente, o Docker possui uma vasta documentação de segurança que os programadores têm suporte.

Portanto, a plataforma dispõe meios para manter o ambiente mais seguro, como ferramentas de análise e proteção, além de manter constantes atualizações. Contudo, foi observado que a instalação padrão do Docker e as configurações minimalistas de contêineres e imagens deixam muitas brechas de segurança abertas. A pontuação da ferramenta de segurança do Docker mostra uma diferença drástica entre um sistema padrão e um sistema blindado, ressaltando a importância da aplicação das técnicas de *hardening*.

A pontuação total do *benchmark* poderia ter sido um pouco diferente caso fosse utilizado outro tipo de aplicação de contêineres, entretanto, as configurações do *host* e do *daemon* envolvendo o Docker *hardening* iriam permanecer. O arquivo de configuração do tipo YAML iria conter comandos parecidos, e muitos dos princípios aplicados neste trabalho se estenderiam em outros sistemas de contêineres.

Contudo, sob a óptica de edificar um sistema de contêineres mais rígido, baseado nos princípios de segurança da informação e de Linux *hardening*, este trabalho atingiu seu objetivo.

Por fim, no contexto de segurança de redes, proteção do sistema operacional Linux, utilização da plataforma Docker e estudo de ambientes virtuais destacando a conteinerização, este trabalho deixa sua contribuição com a literatura dessas áreas por meio de um estudo de segurança da tecnologia.

5.1 Trabalhos Futuros

É sugerido alguns estudos para complementação e seguimento deste trabalho:

- Realizar um estudo mais aprofundado sobre Linux *hardening*, demonstrando na prática as técnicas para proteção da máquina hospedeira.
- Testar outras aplicações de contêineres e realizar outros tipos de testes.
- Discutir sobre segurança da internet a fim de apresentar boas práticas de desenvolvimento *web* e vulnerabilidades mais relevantes neste cenário.
- Manejar um estudo sobre segurança em Kubernetes, um produto *Open Source* utilizado para automatizar a implantação, o dimensionamento e o gerenciamento de aplicativos em contêiner (KUBERNETES, 2014).

REFERÊNCIAS

PAHL, C. *Containerization and the PaaS Cloud* in IEEE Cloud Computing, vol. 2, no. 3, pp. 24-31, 2015.

RAJ MP, A; KUMAR, A; PAI, S. J; GOPAL, A. *Enhancing security of Docker using Linux hardening techniques*. 2016 2nd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT), 2016.

COMBE, T; MARTIN, A; DI PIETRO, R. *To Docker or Not to Docker: A Security Perspective*. in IEEE Cloud Computing, vol. 3, no. 5, pp. 54-62, 2016.

TANENBAUM, A.S. Sistemas Operacionais Modernos: 4ª edição. São Paulo, SP. Pearson Education, 2016.

RANKIN, K. *Linux Hardening in Hostile Networks: Server Security from TLS to Tor*. Boston, Massachusetts. Addison-Wesley, 2017.

TANENBAUM, A.S. Redes de Computadores: 4ª edição. Rio de Janeiro, RJ. Campus, 2010.

CYNET. *Network Attacks and Network Security Threats*. Disponível em: <https://www.cynet.com/network-attacks/network-attacks-and-network-security-threats/>. Acessado em maio de 2022.

CISCO SYSTEMS. *Most Common Cyberattacks*. Disponível em: <https://www.cisco.com/c/en/us/products/security/common-cyberattacks.html>. Acessado em maio de 2022.

REIS, F.A; JULIO, E.P. Hardening em Sistemas Operacionais GNU/LINUX. Revista eletrônica da Faculdade Metodista Granbery, 2010.

CVE. *Top 50 Products By Total Number Of "Distinct" Vulnerabilities.* Disponível em: <https://www.cvedetails.com/top-50-products.php>. Acessado em maio de 2022

LINUX KERNEL ORGANIZATION. *Sudo(8) - Linux Manual Page*. Disponível em: <https://man7.org/linux/man-pages/man8/sudo.8.html>. Acessado em abril de 2022.

TEVAULT, D.A. *Mastering Linux Security and Hardening*. Birmingham, Reino Unido. Packt Publishing Ltd, 2018.

CISOFY. *Lynis, and Introduction*. Disponível em: <https://cisofy.com/lynis/>. Acessado em maio de 2022

CHIUEH, T; SUSANTA, N; BROOK, S. *A Survey on Virtualization Technologies*. *Rpe Report* 142 (2005), 2005.

DOCKER. *Use Containers to Build, Share and Run your Applications.* Disponível em: <https://www.docker.com/resources/what-container/>. Acessado em abril de 2022.

YASRAB, R. *Mitigating Docker Security Issues*. Universidade de Oxford, 2021.

DOCKER. *Docker Run Reference.* Disponível em: <https://docs.docker.com/engine/reference/run/>. Acessado em maio de 2022.

CHELLADHURAI, J; CHELLIAH, P. R; KUMAR, S. A. *Securing Docker Containers from Denial of Service (DoS) Attacks*. *2016 IEEE International Conference on Services Computing (SCC)*. 2016.

DOCKER. *Docker Overview.* Disponível em: <https://docs.docker.com/get-started/overview/>. Acessado em novembro de 2021.

CVE. *Docker: Vulnerability Statistics*. Disponível em: <https://www.cvedetails.com/product/28125/Docker-Docker.html?vendor_id=13534>. Acessado em maio de 2022.

DOCKER. *Docker Bench for Security.* Disponível em: <https://github.com/docker/docker-bench-security>. Acessado em maio de 2022.

DOCKER. *Install Docker Engine on Ubuntu*. Disponível em: <https://docs.docker.com/engine/install/ubuntu/#install-using-the-repository>. Acessado em abril de 2022.

DOCKER. *Post-Installation Steps for Linux*. Disponível em: <https://docs.docker.com/engine/install/linux-postinstall/>. Acessado em abril de 2022

DOCKER. *Protect the Docker Daemon Socket*. Disponível em: <https://docs.docker.com/engine/security/protect-access/>. Acessado em maio de 2022.

LINUX KERNEL ORGANIZATION. *Auditd(8) - Linux Manual Page*. Disponível em: <https://man7.org/linux/man-pages/man8/auditd.8.html>. Acessado em maio de 2022.

AHMED A. *Docker Security Essentials*. Philadelphia, Pennsylvania. Linode LLC, 2021.

DOCKER. *Content Trust in Docker.* Disponível em: <https://docs.docker.com/engine/security/trust/>. Acessado em junho de 2022.

DOCKER. *Vulnerability Scanning for Docker Local Images*. <https://docs.docker.com/engine/scan/>. Acessado em junho de 2022.

KUBERNETES. Orquestração de Contêineres Prontos para Produção. Disponível em: <https://kubernetes.io/pt-br/>. Acessado em junho de 2022.