Análise de segurança entre contêineres Docker implementados em Linux e Windows

Raphael M. Pennacchi[1], **Charles C. Miers**[1]

[1]Departamento de Ciência da Computação
Universidade do Estado de Santa Catarina

raphael.pennacchi@edu.udesc.br,

charles.miers@udesc.br

Resumo. *O uso do contêineres recebeu uma maior atenção por ser uma solução baseada em virtualização alternativa às máquinas virtuais, realizando o encapsulamento de processos visando uma relação melhor de consumo de recursos computacionais e desempenho. Entre as soluções disponíveis de virtualização conteinerizadas, o Docker obteve uma maior preferência de uso. O Docker foi inicialmente um software de soluções de contêineres desenvolvido para o sistema operacional GNU/Linux. A Microsoft adentrou no mercado disponibilizando a o Docker Windows para as versões de servidores e as versões de uso pessoal do seu sistema operacional. Nas versões específicas para servidores há a possibilidade de execução do Docker Windows nativamente na plataforma. Neste sentido, este artigo apresenta os resultados iniciais de uma análise de alguns aspectos de controle de recursos, segurança de imagem de contêineres e segurança na comunicação entre contêineres.*

1. Introdução

A virtualização pode ser disponibilizada de diversas maneiras, abstraindo diversos recursos. Na virtualização os recursos físicos são abstraídos para múltiplas camadas lógicas. Com a virtualização é possível aprimorar a utilização de recursos computacionais, de comunicação e de armazenamento [Kabbe 2017]. Nas abstrações para múltiplas camadas lógicas, tipicamente máquinas virtuais (MVs), realizam a abstração do hardware hospedeiro. Quando o hardware é abstraído é necessário a instalação de um sistema operacional (SO). Essa duplicação de serviços operacionais consomem recursos. A virtualização em nível de SO, conhecida como contêineres, criam um ambiente isolado sem a abstração do hardware. Contêineres utilizam funcionalidades do núcleo para criar os ambientes isolados. Eliminando a camada de abstração de hardware e a necessidade de instalação de um SO. Os contêineres receberam uma maior atenção pelo menor consumo de recursos ao eliminar a camada de abstração de hardware, a qual permite a criação de aplicações e microsserviços de forma modular. Dentre as tecnologias de conteinerizações disponíveis o Docker se destacou [DIAMANTI 2019]. O Docker foi inicialmente projetado para realizar conteinerizações no SO Linux. A Microsoft em parceria com o Docker disponibilizou uma versão para os SOs de servidor e uso pessoal da empresa. Com o surgimento desta nova versão para um SO com base diferente, surgem questões relativas a aspectos de segurança. Falhas de segurança existentes no núcleo do Linux podem ser sanados por esta nova versão do Docker, e outro problemas podem surgir. Há estudos sobre segurança de contêineres Docker na plataforma GNU/Linux, mas não foram identificados estudos

analisando a segurança do Docker Windows até o momento da submissão deste artigo. O presente trabalho tem como objetivo definir um escopo de análise de segurança de contêineres Docker, tanto GNU/Linux como Windows, utilizando como base os critérios de [Sultan et al. 2019]: *(i)* Controle e limitação de recursos e *(ii)* Segurança das imagens de contêineres.

2. Arquitetura Docker: Linux vs Windows

As soluções a base de contêineres receberam uma atenção considerável como uma solução de virtualização para a criação de aplicações e micro-serviços. Sendo uma solução inicialmente exclusiva do SO GNU/Linux, a Microsoft ao criar um serviço de contêineres para seu SO visa permitir a utilização de contêineres em sua plataforma de forma similar a qual já era permitida. Para realizar tal feito é necessário uma interface que permita executar os mesmos comandos independente da plataforma. Para isso o Docker utiliza uma camada independente do SO. Essa camada independente trabalha como uma abstração, permitindo os comando serem executados de forma semelhante em ambos SOs Linux e Windows. A camada independente é uma *Application Programming Interface* (API) *Representational State Transfer* (REST) a qual se comunica com o mecanismo Docker de cada sistema para realizar a criação e gerenciamento na camada específica que depende do SO utilizado. Assim, o Docker atua como *middleware* nas arquiteturas. Na Figura 1 são listadas as arquiteturas independente e dependente da plataforma. Na parte da arquitetura dependente da plataforma são listados os recursos do núcleo dos respectivos SOs utilizados que o mecanismo Docker utiliza para realizar as restrições de recursos. Os diferentes SOs possuem funcionalidades distintas sendo empregadas nos seus núcleos para conteinerizar.

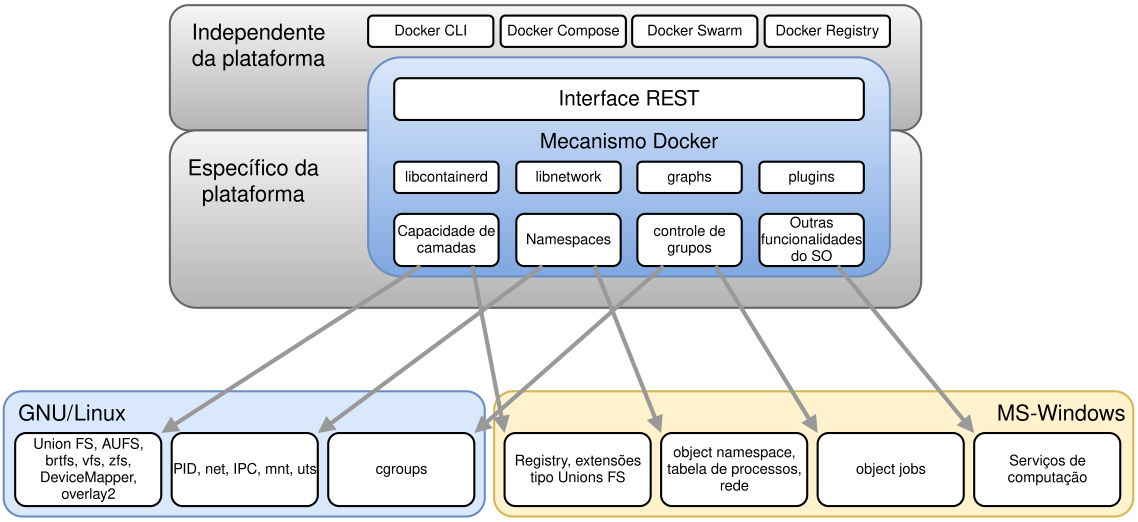

Figura 1. Arquitetura Docker: GNU/Linux e Windows.

A Figura 1 evidencia os principais componentes do MS-Windows Server para fornecer o serviço de conteinerização. Neste contexto, surge a questão relativa a problemas de segurança já detectados na arquitetura do GNU/Linux possuírem ou não a suscetibilidade de ocorrer na arquitetura MS-Windows. Já foram detectados alguns problemas referentes ao chroot [Details 2017] [NIST 2019]. Sendo realizados testes sobre alguns cenários de ataques possíveis, com foco em restrição de recursos que realizam o isolamento dos contêineres.

Ressalta-se, que o ambiente de execução de contêineres Docker no MS-Windows Server 2016 / 2019 é diferente do ambiente das versões do MS-Windows para *desktop* (Figura 2). O Docker Desktop, para versões do SO MS-Windows *desktop*, executa os contêineres dentro de uma MV (MS-Hyper-V) otimizada (baseada em GNU/Linux Ubuntu ou GNU/Linux CentOS) cuja parte do isolamento dos recursos já pode ser realizada pela forma de operação da MV. Apenas na versão MS-Windows 10 Pro que há um modo de execução de contêineres similar ao MS-Windows Server 2016/2019 [Docker 2020].

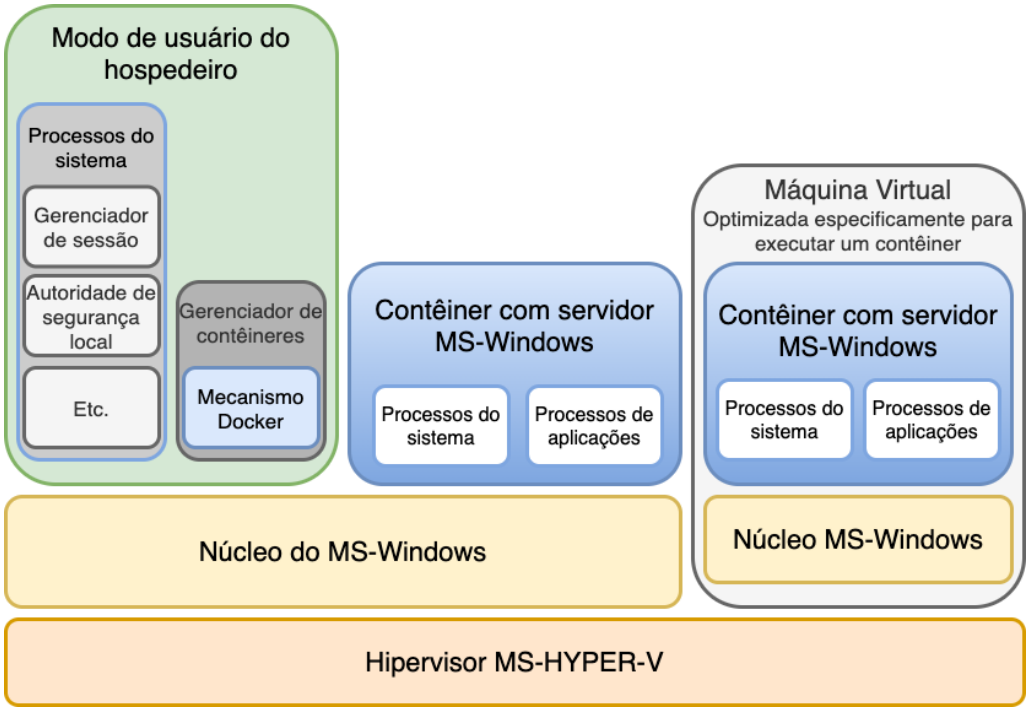

Figura 2. Diferentes abordagens da Microsoft para Docker Windows.

3. Critérios de Análise

Os critérios utilizados foram baseados no guia de segurança de contêineres [NIST 2017], no modelo de ameaças [Sultan et al. 2019] e na revisão de falhas de segurança em contêineres Linux[Lin et al. 2018]. Uma análise visando restringir a quantidade de critérios foi realizada para definir um foco inicial na pesquisa deste artigo, que pode ser ampliado a medida que as análises de cada critério são finalizadas. Critérios iniciais definidos:

1. Controle e limitação de recursos: Realizar a análise de mecanismo de controle de acesso e escalação de recursos.
2. Segurança das imagens de contêineres: Verificar a existência de vulnerabilidades em imagens de contêineres hospedadas em repositórios da comunidade a partir de ferramentas de busca de vulnerabilidades, e recomendar soluções baseadas no modelo de ameaças.
3. Isolamento e restrição de acesso ao tráfego de rede: contêineres normalmente são usados para implementar microsserviços que precisam transmitir dados (outros contêineres, MVs, *hosts*, etc.) gerando tráfego de rede ou comunicações interprocessos.

A partir dos critérios apresentados foram elaborados experimentos para produzir resultados que possibilitem analisar o comportamento do contêiner em ambos SOs (GNU/Linux e MS-Windows Server).

4. Proposta e Ambiente de experimentação

A proposta consiste em implementar contêineres Docker Linux e Windows em uma MV com as mesmas configurações de alocação de memória e espaço de armazenamento a fim de verificar as capacidades de controle e limitação de recursos. Os experimentos tem por finalidade não apenas verificar o isolamento mas também a degradação ou não do desempenho do *host* e do contêiner envolvido no experimento. Os experimentos estão sendo realizados em um computador (processador i7-9750h, 12GB RAM, 6 cores). Foram criadas duas MVs, sendo disponibilizado 50GB de armazenamento em um disco rígido e 4GB RAM para cada. As MVs estão com os SOs GNU/Linux Ubuntu 20.04 Focal Fossa e Microsoft Windows Server 2019.

Em ambas MVs estão sendo aplicados testes específicos para a análise de segurança referente a cada critério (Seção 3). Para o experimento de controle e limitação de recursos, estão sendo utilizados dez contêineres ativos simultaneamente, em um dos contêineres ativos é utilizado um ataque de negação de serviço (Dos) baseado no princípio de *fork-bomb*. Com esse experimento pode ser analisado se a limitação padrão de segurança dos contêineres está ativada e se essa limita o uso de memória ou permite um ataque de negação de serviço interno, assim afetando os outros contêineres.

Os experimentos planejados são:

1. Analisar a configuração padrão de capacidades do Docker Windows e Docker Linux e suas capacidades de restrição e limitação de recursos.
2. Verificar a possibilidade de alterar no nome do hospedeiro com capacidades padrões e com capacidades de administrador seguindo o fluxo apresentado na Figura 3.
3. Analisar o comportamento de cada SO ao executar um *forkbomb* em um contêiner criado com a configuração padrão e com uma limitação de memória explícita seguindo o fluxo apresentado na Figura 4.
4. Analisar o comportamento do gerenciador do sistema operacional e do gerenciador do contêiner durante todos os experimentos.

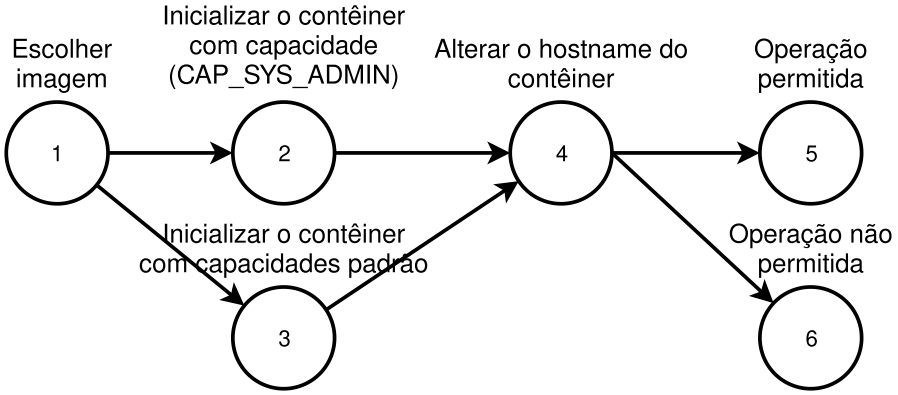

Figura 3. Sequência de experimento: contêineres com capacidade CAP_SYS_ADMIN.

Para realizar o Experimento 2 (Figura 3) é selecionada uma imagem para inicializar o contêiner, são criados dois contêineres. O primeiro com a capacidade padrão do Docker, e o segundo com a capacidade CAP_SYS_ADMIN adicional, após serem inicializados é utilizado um comando para alterar o *hostname* do contêiner, a operação de alterar o *hostname* pode ser permitida ou não.

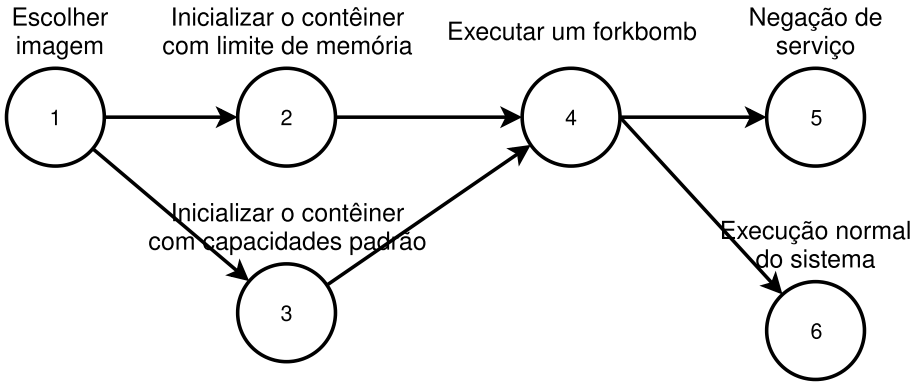

Figura 4. Sequência de experimento: contêineres com limitação de memória.

Para realizar o Experimento 3 (Figura 4) é selecionada uma imagem para inicializar o contêiner, são criados dois contêineres. O primeiro contêiner com a capacidade de restrição de memória padrão do Docker. O segundo contêiner com uma restrição de memória explicitamente definida, após serem inicializados é executado um *forkbomb* em cada contêiner, o *forkbomb* pode realizar uma negação de serviços a partir do sequestro de recursos.

5. Resultados Parciais

Os resultados iniciais já revelaram diferenças entre os SOs. O experimento com escalação de privilégios para alteração do *hostname* dos contêineres apresentou resultados distintos entre os SOs. No Docker Linux só é possível a execução do comando quando há a capacidade de *SYS ADMIN* adicionada ao contêiner. No Docker Windows é possível realizar a troca de *hostname* sem a capacidade de *SYS ADMIN* adicionada ao contêiner. Porém, inspecionando o contêiner foi verificado que não havia alterações no *hostname* tanto no contêiner com a capacidade de *SYS ADMIN*, quanto o sem a capacidade.

No teste de restrição de recursos os resultados foram similares. Ambos Docker Windows e Docker Linux não apresentaram uma limitação de memória pré definida nos contêineres. Caso o lançamento do contêiner seja realizado ignorando tal pré-configuração, ou até mesmo uma configuração indevida, o hospedeiro pode tornar-se vulnerável a ataques de *Denial of Service* (DoS). No experimento, com ambos os sistemas, ao executar um *forkbomb* com a configuração padrão do contêiner Docker pararam de responder, esgotando todos os recursos de memória. Quando os experimentos foram realizado com a especificação de um limite de 256MB para cada contêiner, ao demandarem mais recursos que o pré-definido o contêiner é finalizado (não afetando o funcionamento do hospedeiro e dos outros contêineres).

Os experimentos iniciais realizados já revelaram que algumas opções padrão de contêineres Docker GNU/Linux que apresentavam falhas de segurança apresentaram resultados distintos no Docker Windows. O teste de alteração de *hostname* apresenta uma

execução peculiar, no qual o comando é aceito mas não há alterações aparentes no *host-name* do contêiner no Docker Windows. Os contêineres executados em ambos SO não possuem definida uma limitação padrão do limite de quantidade de memória principal que um contêiner pode utilizar, o que pode acarretar em ambientes mal configurados um ataque de DoS.

6. Considerações & Trabalhos futuros

Os experimentos realizados foram conduzidos com as versões padrões dos SOs, instalados e configurados pelos autores empregando a documentação oficial do Docker e Microsoft, usando as recomendações padrão de configuração. Os resultados preliminares mostram diferentes resultados para um mesmo experimento nos diferentes SOs. Mais experimentos serão realizados usando imagens de MVs específicas para execução de contêineres. Além disso, planeja-se testar em ambientes de nuvens preparados para conteinerização *bare-metal*, e.g., serviço Zun disponível no OpenStack para contêineres Docker.

Além disto, a interface de rede nos contêineres Docker Windows tem distinções com o Docker Linux. Os experimentos realizados até o momento não empregaram testes com as diferentes opções de comunicação entre contêineres usando interface de rede. Em um trabalho futuro será analisado o comportamento das interfaces de rede dos contêineres do Docker Windows e do Docker Linux, e se houver, verificar as diferenças de comportamento entre estas.

Agradecimentos: Os autores agradecem o apoio do LabP2D/UDESC e a FAPESC.

Referências

Details, C. (2017). Cve-2015-6240. "https://www.cvedetails.com/cve/CVE-2015-6240/".

DIAMANTI (2019). Container adoption benchmark survey. https://diamanti.com/wp-content/uploads/2019/06/Diamanti_2019_Container_Survey.pdf.

Docker (2020). Get started with Docker for Windows. https://docs.docker.com/docker-for-windows/.

Kabbe, J.-A. (2017). Security analysis of docker containers in a production environment. Master's thesis, Norwegian University of Science and Technology. https://ntnuopen.ntnu.no/ntnu-xmlui/bitstream/handle/11250/2451326/17303_FULLTEXT.pdf?sequence=1&isAllowed=y.

Lin, X., Lei, L., Wang, Y., Jing, J., Sun, K., and Zhou, Q. (2018). A measurement study on linux container security: Attacks and countermeasures. In *Proceedings of the 34th Annual Computer Security Applications Conference*, ACSAC '18, page 418–429, New York, NY, USA. Association for Computing Machinery.

NIST (2017). Application container security guide. https://csrc.nist.gov/csrc/media/publications/sp/800-190/draft/documents/sp800-190-draft.pdf.

NIST (2019). Cve-2019-3811 detail. "https://nvd.nist.gov/vuln/detail/CVE-2019-3811".

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