Uma análise de segurança no uso de contêineres Docker em nuvens IaaS OpenStack

Kerolayne de S. V. de Oliveira[1], Guilherme Panizzon[1], Maurício A. Pillon[1], Guilherme P. Koslovski[1], Charles C. Miers[1], Nelson M. Gonzalez[2]

[1]Programa de Pós Graduação em Computação Aplicada (PPGCA)
Universidade do Estado de Santa Catarina – Joinville, SC – Brasil

kerolayne.oliveira@edu.udesc.br, panizzonguilherme@gmail.com,

mauricio.pillon,guilherme.koslovski,charles.miers}@udesc.br

[2]IBM Watson Research Center, Yorktown Heights, NY, USA

nmimura@us.ibm.com

Abstract. *In the past years, the use of virtualization on the cloud has grown considerably. Virtualization solutions based on hypervisor and containers became widely used in combination with PaaS and IaaS environments through considerable performance, isolation and scalability provisioning. However, the container's adoption by the market is recent when compared to VMs. As consequence, concern with container's security is relevant. Vulnerabilities in the kernel mechanisms and in container's solutions are a risk for the data and the service's functioning of all entities that share the same hardware. Therefore, there is a need to identificate and classify threats, risks and vulnerabilities for container's security. We perform a security analysis on Docker containers for the OpenStack clouds.*

Resumo. *Nos últimos anos, o uso de virtualização na computação em nuvem aumentou consideravelmente. Soluções de virtualização baseada em hipervisores e contêineres passaram a ser utilizadas em combinação em ambientes PaaS e IaaS, devido às suas características de desempenho, isolamento e escalabilidade. Contudo, a adoção do uso de contêineres pelo mercado é recente quando comparada à MVs. Como consequência, a preocupação com a segurança de contêineres possui relevância. Vulnerabilidades em mecanismos do núcleo do SO e em soluções para contêineres são um risco para os dados e o funcionamento do serviço de todas as entidades que compartilham um mesmo hardware. Sendo assim, há a necessidade de identificar e classificar ameaças, riscos e vulnerabilidades em contêineres. Este trabalho relata uma análise de segurança do uso de contêineres Docker para nuvens OpenStack.*

1. Introdução

A virtualização é uma das técnicas largamente utilizadas em nuvens computacionais, tanto em *Platform-as-a-Service* (PaaS) quanto em *Infrastructure-as-a-Service* (IaaS), permitindo que usuários/organizações tenham ambientes com compartilhamento de hardware, isolados, seguros e escaláveis [Bui 2015]. Recursos computacionais podem ser virtualizados a nível de sistema operacional (SO) (*OS-based*) ou hipervisor (*H-based*).

A conteinerização (*OS-based*) passou a ser amplamente difundida com o surgimento do Docker em 2013 [DOCKER 2016]. Quando comparados com máquinas virtuais (MVs) (*H-based*), contêineres destacam-se, positivamente, pelo desempenho e pelo flexibilidade, e são questionados nos quesitos isolamento e segurança [Alles et al. 2018], [Arango et al. 2017],[Kozhirbayev and Sinnott 2017].

Neste contexto, o presente trabalho tem como objetivo analisar a segurança do uso de contêineres em um ambiente de computação em nuvem real. O ambiente de nuvem analisado tem como gerenciador de recursos o OpenStack, tipo IaaS, e conteinerização via Docker. A principal contribuição deste trabalho baseia-se na relevância de uma análise de vulnerabilidades, riscos e ameaças em um ambiente amplamente utilizado pela comunidade. O artigo está organizado da seguinte maneira: a Seção 2 trata da computação em nuvem, contêineres, segurança de contêineres, OpenStack, configuração de contêiner através do Magnum e manualmente. A Seção 4 descreve a análise nos cenários definidos, assim como os critérios de análise e os resultados.

2. Virtualização de Recursos: Conteinerização

A virtualização de recursos a nível de SO (*OS-based*), através de contêineres, possibilita a utilização de um mesmo hospedeiro físico por dois ou mais usuários ou aplicações. Esta abordagem de virtualização de recursos apoia-se nos serviços do SO *namespace* e *cgroups*, gerenciados diretamente pelo núcleo deste sistema. Aplicações conteinerizadas compartilham bibliotecas e recursos virtuais orquestradas pelo próprio SO. Esta categoria de virtualização é reconhecida por ter *overhead* baixo ou inexistente, mesmo quando comparado a execuções de aplicações idênticas e com as mesmas cargas em ambientes nativos. Se comparada com virtualização baseada em hipervisor (*H-based*), contêineres destacam-se na maioria dos aspectos [Gao et al. 2017].

Contudo, o compartilhamento de um mesmo sistema operacional e, consequentemente, suas estruturas, ampliam as possibilidades de ataques. A disponibilização dos serviços de virtualização, com base na conteinerização, por grandes provedores de nuvem, *e.g.,*Amazon EC2 e Google App Engine, torna a análise de segurança essencial a usuários/organizações [NCC GROUP 2016]. Contêineres e máquina virtuals (MVs) são técnicas de virtualização de recursos complementares, podendo, em alguns casos, serem utilizadas concomitantemente. No modelo de serviço de nuvem computacional IaaS é possível se ter: (i) somente contêiner (com o auxílio do *Linux Container Hypervisor* (LXD) [LINUXCONTAINERS.ORG 2016]); (ii) somente MV; ou ainda, (iii) contêineres sob MVs [Bernstein 2014]. Pode-se afirmar que o nível de isolamento entre aplicações conteinerizadas difere de acordo com a forma de implantação do contêiner. O isolamento e o grau de compartilhamento de recursos e bibliotecas afetam diretamente a segurança da abordagem de virtualização de recursos.

2.1. OpenStack

O OpenStack é uma plataforma de orquestração de recursos em nuvem computacional que, desde 2014, tem incorporado módulos e ferramentas para uso de contêineres [OPENSTACK 2018]. Tem como foco, possibilitar a seus usuários, o gerenciamento de um contêiner da mesma forma que uma MV. Atualmente, OpenStack possui três módulos para conteinerização: (i) *Kolla* responsável pelo *deploy* de contêineres Docker associado ao Kubernetes para gerenciamento; (ii) Murano responsável pela execução de uma

instância conteinerizada; e (iii) Magnum que é uma API que disponibiliza a orquestração de contêineres Docker e *engines* de contêineres [OPENSTACK.ORG 2017].

O módulo responsável pelo isolamento entre os contêineres é o Magnum. Ele segue o mesmo modelo do módulo Nova, responsável pelo isolamento entre MVs, utilizando a solução de composição de *bays*(clusters). Esses *clusters* são isolados entre os inquilinos, garantindo que contêineres de diferentes *clusters* não serão executados no mesmo núcleo. Para a associação de recursos a contêineres, Magnum solicita os recursos necessários ao Heat, se for orquestração de imagens, vCPUs e memória, ao Neutron, se for rede, e ao Cinder, se o recurso for volume de armazenamento.

3. Conteinerização: Segurança

A tecnologia de virtualização baseada em hipervisor (MVs) está atualmente difundida, mesmo no ambiente de produção. A conteinerização apoia-se em propriedades semelhantes às das MVs [OPENSTACK SECURITY 2016]. Todavia, a adoção desta tecnologia junto ao ambiente de produção ainda tem como maior barreira a segurança [CLUSTERHQ 2016]. O princípio básico da conteinerização, compartilhamento de recursos através de um mesmo núcleo, gera, junto a comunidade, desconfiança sobre a existência velada de riscos e vulnerabilidades.

Preocupações de segurança relacionadas às MVs são conhecidas da comunidade, e guias de recomendação de segurança para ambientes de computação em nuvem, com virtualização via hipervisor, já são públicos [NCC GROUP 2016]. No âmbito de contêineres, os principais elementos descritos por este documento referem-se a segurança envolvendo os recursos do núcleo, problema com código legado e complexidade na orquestração em escala. A compreensão das ameaças e a origem dos principais ataques só é possível com a análise de funcionamento e vulnerabilidade dos principais componentes de contêineres, *i.e., namespaces* do núcleo, *control groups*, capacidades, *pivot_root* e *Mandatory Access Control* (MAC).

[SANS.ORG 2015] publicou um informe técnico sobre as melhores estratégias para a execução de contêineres de forma segura. Diferente do guia NCC [NCC GROUP 2016], este possui um enfoque em descrever as estratégias no gerenciamento de servidores com a virtualização baseada em MV e em contêiner. Dependendo da aplicação, existem várias limitações (*e.g.,* desempenho, compatibilidade e segurança) que são comuns no uso de contêineres, sendo esse o motivo para não ser escolhido dependendo do domínio da aplicação. É possível utilizar de diversas estratégias para hospedar um serviço atualmente, tais como: (i) múltiplos servidores, cada um executando múltiplas aplicações, e (ii) múltiplos servidores baseados em contêineres, cada um executando apenas uma aplicação [SANS.ORG 2015].

Embora este guia descreva as duas estratégias, o seu maior enfoque é na de múltiplos servidores utilizando contêineres e contendo somente uma aplicação por contêiner. [SANS.ORG 2015] explicita que criar uma camada de separação e isolamento é a chave para assegurar qualquer impacto causado por possíveis contêineres comprometidos. As sugestões para contornar possíveis ameaças é a implementação de *Discretionary Access Control* (DAC) e MAC. DAC se refere às políticas de segurança que podem ser sobrescritas dependendo do contexto de execução do processo, tal que seu conceito é o contrário de MAC no qual uma política só será alterada caso for desativada completamente.

4. Análise de Segurança de Contêineres em OpenStack

Com base nos guias de segurança, nos trabalhos correlatos e nos aspectos de segurança, constata-se a relevância em identificar e classificar as preocupações e mecanismos de segurança para contêineres. De acordo com as preocupações levantadas, são definidos três critérios:

1. **Controle e limitação de recursos**: Analisar os mecanismos de controle de acesso, de limitação de recursos e capacidades que garantem o isolamento e controle de acessos à recursos entre o núcleo do SO e contêineres vizinhos. Garantir o isolamento e limitação de recursos é relevante para a segurança do contêiner. Uma motivação para o critério é a ativação por padrão desses mecanismos na inicialização do contêiner de acordo com [DOCKER 2018].
2. **Segurança das imagens de contêineres**: Verificar a existência de vulnerabilidades em imagens de contêineres hospedadas em repositórios da comunidade e por fim, recomendar soluções para auditar tais vulnerabilidades que já são conhecidas e catalogadas por banco de dados de vulnerabilidades como *Common Vulnerabilities and Exposures* (CVE) e *National Vulnerability Database* (NVD). De acordo com [Shu et al. 2017], mais de 80% das imagens do repositório Docker Hub possuem pelo menos uma vulnerabilidade de severidade alta, de acordo o índice de risco da CVE. Mais de 50% das imagens da comunidade do Docker e do repositório oficial não são atualizadas a mais de 200 dias e cerca de 30% não são atualizadas a mais de 400 dias.
3. **Segurança dos dados na comunicação entre contêineres**: Analisar a segurança na comunicação entre contêineres que utilizam o mesmo núcleo. De acordo com [CSA 2017], contêineres devem possuir um perímetro de segurança para proteger, detectar e prevenir contra ataques como também qualquer outra camada de rede virtual utilizada. Tal que isso é o contrário da configuração padrão do contêiner, na qual é concedida a troca de informações entre contêineres sob um mesmo núcleo. Tais configurações podem resultar no aumento do nível de risco com o vazamento de informações para contêineres não desejados. Portanto, é necessário garantir o tráfego apenas para contêineres específicos, reduzindo a superfície de ataque pela restrição de acesso e garantindo a confidencialidade e integridade do tráfico de rede [CIMCOR 2017].

4.1. Cenários, experimentos e resultados

Foram realizados os experimentos em 6 cenários, todos de acordo com a configuração do ambiente de testes (Figura 1). A topologia escolhida tem uma MV (`vm1 - IP 10.0.0.5`) com GNU/Linux Ubuntu 16.04 LTS e Docker Community $7.09.0$. O contêiner possui Linux Ubuntu Xenial $16.0.4$ e responde no `IP:172.17.0.2`.

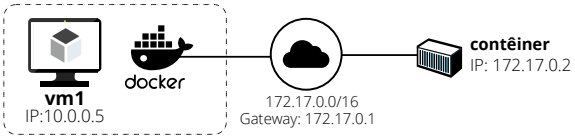

Figura 1. Ambiente de Experimentação.

Cenário 1. O objetivo dessa experimentação é investigar se contêineres Docker possuem o mecanismo MAC AppArmor configurado por padrão. MAC AppArmor tem por objetivo proteger o *SO* e suas aplicações através da política conhecida como *AppArmor profile*. Com *Docker*, usuários podem especificar suas

próprias políticas de segurança por contêiner (*profile*). Na ausência de um *profile* específico, as políticas aplicadas ao contêiner em questão são as especificadas no *docker-default* [DOCKER 2017a]. Para obtenção das configurações carregadas pelos contêineres inspecionados utilizou-se o *inspect*. Assim, obteve-se o ID $684585ae22bc71817b80bcc9c44894fbd018b530e76e2974120ca8455a9f0529$ (abreviado $684585ae22bc$) e *profile* utilizado $AppArmorProfile$=$docker\text{-}default$, neste caso, o *profile* padrão do *Docker*. A falta de especificações de políticas de seguranças personalizadas por contêineres podem permitir operações indesejadas, tais como leituras ou escritas em áreas restritas. A exploração de uma vulnerabilidade pode afetar unicamente o contêiner ou o próprio *SO*.

Cenário 2. Tem como objetivo verificar quais são as capacidades de acesso ao núcleo habilitadas/restritas a um contêiner padrão (Figura 2). O método adotado foi a invocação do comando *hostname* por um contêiner. A execução deste comando exige que o contêiner possua a capacidade de superusuário, o que não deve estar habilitada por padrão. Portanto, espera-se que o comando não tenha sucesso.

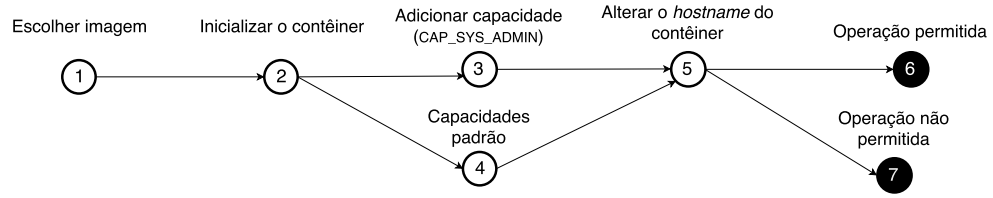

Figura 2. Possíveis combinações do uso de MVs e contêineres.

Com *Docker*, no que se refere a privilégios de superusuário, o padrão é o não consentimento. No Linux, desde a versão 2.2 do núcleo, processos são classificados de acordo com suas capacidades. As capacidades de um processo são independentes umas das outras e são habilitadas individualmente [MAN7.0RG 2017]. Desta forma, um contêiner pode possuir uma capacidade específica do núcleo sem que ele tenha privilégios de superusuário. As capacidades comumente exploradas em contêineres Docker são: *CAP_SYS_ADMIN* e *CAP_NET_ADMIN* [NCC GROUP 2016]. A capacidade *CAP_SYS_ADMIN* engloba 35 operações do núcleo do sistema, tais como: alteração do *hostname*, *domainname*, criação de novos *namespaces*, controle da porta serial e modificação do compartilhamento de memória. Uma vez esta capacidade habilitada, o contêiner pode explorar incrementos de privilégios, por exemplo, tornar-se superusuário. Neste contexto, tentou-se alterar o *hostname* em dois ambientes distintos: (i) contêiner com inicialização padrão (sem a capacidade de *CAP_SYS_ADMIN*); e (ii) contêiner com a capacidade de *CAP_SYS_ADMIN*. Como esperado, o contêiner não foi capaz de alterar o *hostname* no ambiente (i). O teste foi repedido no ambiente (ii), desta vez, o comando foi executado com sucesso. Dessa forma, fica evidente que o mal uso das capacidades pode comprometer a segurança do contêiner e do próprio núcleo. Auditar quais capacidades um serviço ou aplicação necessita é relevante na garantia da segurança dos inquilinos que compartilham um mesmo núcleo.

Cenário 3. Tem-se como objetivo investigar o impacto da segurança de contêiner com foco na equidade de alocação de recursos. O cenário é constituído por dez contêineres Docker GNU/Linux Ubuntu Xenial $16.0.4$ LTS com IPs entre `172.17.0.2 - 172.17.0.11`. O contêiner nomeado de `cont_bomb` recebe a execução do código `forkbomb.c` (Figura 3). Este código fica em um *loop* consumindo toda a memória disponibilizada pelo núcleo.

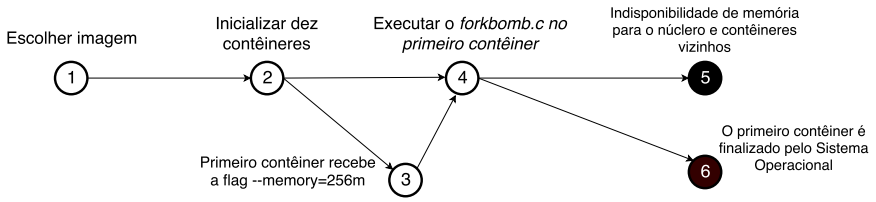

Figura 3. Possíveis combinações do uso de MVs e contêineres.

Em contêineres Docker, o uso de recursos do núcleo, por padrão, é irrestrito. No entanto, o Docker dispõe de *flags* para gerenciar: o consumo de memória, o número de operações de *I/O*, o percentual de utilização CPU e a quantidade de armazenamento em disco [DOCKER 2017b]. A ausência de controle do consumo máximo de memória de um contêiner e, portanto, a utilização inadequada de recursos pode ser prejudicial aos contêineres inquilinos e ao próprio núcleo. A exceção *Out Of Memory Exception* acarreta sobrecarga adicional ao núcleo em busca de liberação de espaço em memória. Quanto ao aspecto de segurança, a possibilidade de acesso total a memória, permite a exploração de ataques do tipo *Denial of Service* (DoS) que, se bem sucedido, pode resultar na queda do hospedeiro ou na indisponibilidade de serviços de outros inquilinos ou do sistema. Portanto, nesse experimento busca-se verificar a possibilidade de limitar do uso de memória na inicialização do contêiner. A primeira constatação, o contêiner `cont_bomb` impediu o funcionamento correto os inquilinos e do núcleo. Pode-se afirmar que a configuração dos *flags* para limitar o uso de memória por contêiner é essencial. O segundo procedimento aplicou a limitação no uso de memória ao contêiner `cont_bomb`. Como resultado, o comportamento nocivo do algoritmo `cont_bomb` não foi capaz de tornar os serviços dos demais inquilinos e do núcleo indisponíveis. Assim que o contêiner atingiu o limite estabelecido de alocação de memória (256MB), o núcleo finalizou o contêiner `cont_bomb`.

Cenário 4. O objetivo deste experimento é investigar o impacto do mecanismo *Seccomp*. Habilitado por padrão em contêineres Docker Linux, ele é responsável pela filtragem de chamadas da API do núcleo (Figura 4). No entanto, cabe mencionar que o *Seccomp* não é suportado pelo SO Apple MacOS Sierra (Docker Community Edition) [Schanzju 2017].

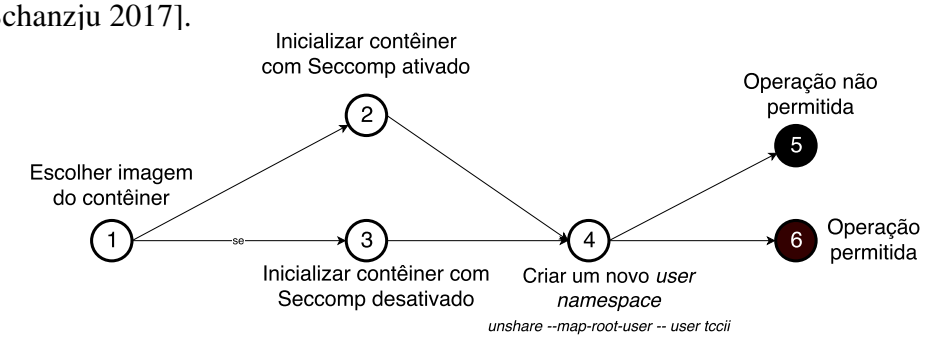

Figura 4. Fluxograma do experimento Cenário 4.

Em ambientes *Docker* Linux, no qual *Seccomp* é suportado, sua configuração é habilitada via *profile*. O *docker-default*, *profile* padrão do Docker, desativa 44 chamadas de um total de 300 disponibilizadas pela API [DOCKER 2017c]. Para esse experimento, analisou-se, a chamada de sistema *unshare*, restrita por padrão. A chamada *unshare* é

responsável pela clonagem de *namespaces*, não exigindo criação um novo processo. Portanto, ela possibilita que processos ou *threads* sejam associados/desassociados a um novo contexto compartilhado [DOCKER 2017c]. Para testar o impacto do mecanismo *Seccomp* com a chamada *unshare*, executou-se, dentro de um contêiner, a própria chamada com as seguintes *flags*: `unshare --map-root-user --user`. A *flag* utilizada concede ao novo *namespace* a escalada de privilégios [SYSTUTORIALS.COM 2017]. Este procedimento foi efetuado em dois ambientes: (i) com *docker-default*; (ii) *profile* com *Seccomp* desativado. Como resultado da execução do chamada no ambiente (i), obtive-se a mensagem de erro *Operation not Permitted*. Reforçando a ideia inicial que *Seccomp* filtra essa chamada, bloqueando a operação. Para o ambiente (ii), o resultado obtido foi a criação do novo *namespace* do usuário e atribuição de escalada de privilégios. A comprovação desta escalada veio com execução do comando `whoami`, cujo retorno foi `root`. Constatou-se que é possível se tornar superusuário através do uso da chamada *unshare*, caso o mecanismo *Seccomp* esteja desabilitado.

Cenário 5. Tem como foco a auditoria de vulnerabilidade das imagens. As duas principais ferramentas de auditoria de imagens de contêineres encontradas foram: Docker Security Scanning e Clair-scanner. O Docker Security Scanning é uma solução proprietária disponibilizada pelo Docker Hub. Clair-scanner é de código aberto e integrada ao repositório do CoreOS. O ambiente de testes é constituído por uma MV (`vm1 - IP 10.0.0.5`) com GNU/Linux Ubuntu 16.04 LTS e Docker Community 7.09.0 (Figura 5). A partir de uma imagem de contêiner com a GNU/Linux Ubuntu Xenial 16.0.4 LTS, inicia-se um contêiner Docker com `IP:172.17.0.2`. A escolha da Docker Security Scannnig, como ferramenta de auditoria, se deu pela facilidade de uso e total integração com o ambiente Docker Linux utilizado.

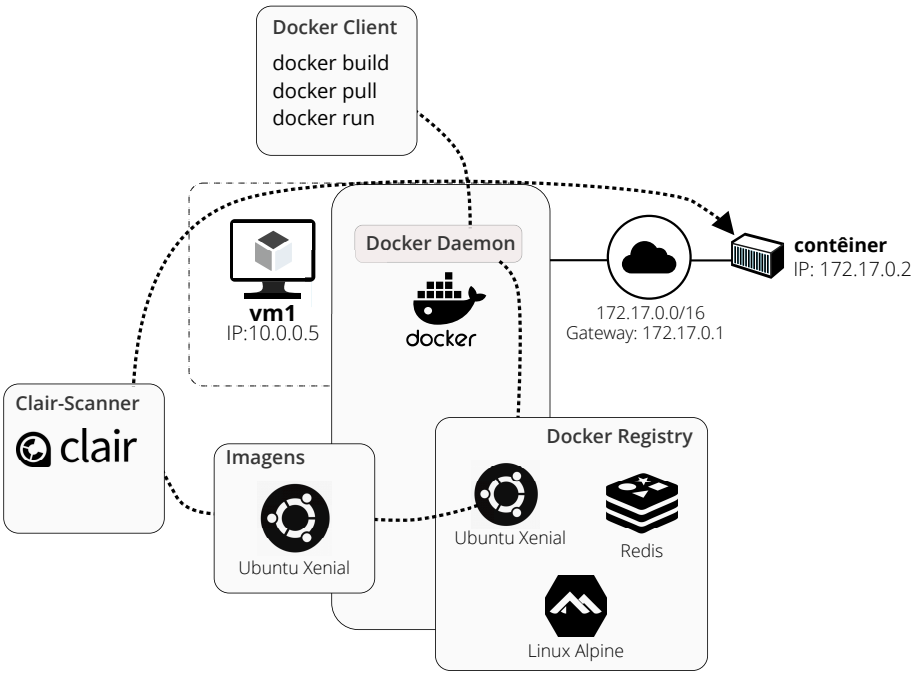

Figura 5. Configuração do Cenário 5 de experimentação.

Para atingir o objetivo de auditoria da imagem, precisa-se esclarecer o fluxo necessário a obtenção de uma imagem. A primeira etapa do processo é o download da imagem. O processo de *download* começa com o Docker Client requisitando ao Docker Daemon

um objeto contêiner. O Docker Daemon repassa a solicitação ao Docker Registry, nesse cenário, o próprio Docker Hub, que, finalmente, fará o *download* da imagem solicitada. De posse da imagem, o Docker Security Scanning entra em ação, elencando potenciais vulnerabilidades da imagem em questão. O funcionamento do auditor Clair-Scanner resume-se ao cruzamento de dados obtidos com a imagem analisada com dados conhecidos e públicos do repositório CVE. A imagem escolhida para auditoria foi a imagem oficial do GNU/Linux *Ubuntu Xenial*. O relatório de auditoria apontou 9 vulnerabilidades na imagem auditada. Pode ser constatado que, cada componente vulnerável encontrado é relacionado com as vulnerabilidades catalogadas pelo CVE em conjunto com o grau de severidade (*e.g., major, critical e minor)*. Dessa forma, é garantido a possibilidade de auditar as imagens de contêineres, tal que a execução de uma imagem vulnerável abre espaço para atacantes realizarem execução de código arbitrário.

Cenário 6. O objetivo dessa experimentação é analisar o impacto na segurança do compartilhamento da rede. O cenário é composto por: uma MV com GNU/Linux Ubuntu Xenial $16.0.3$ LTS e 3 contêineres Docker Community Edition. Como pode ser observado na Figura 6, os contêineres criados têm as seguintes atribuições: (i) *cont_consum*, consumir uma API REST, enviando requisições HTTP ao servidor web conteinerizado (*cont_python*); (ii) *cont_python*, disponibilizar um servidor *web* na linguagem *Python* (porta $TCP/8000$); e (iii) *cont_sniffing*, capturar o tráfego de rede com TCPDump. Dada a necessidade de captura do tráfego, concedeu-se ao contêiner *cont_sniffing* todas as capacidades a chamada ao núcleo.

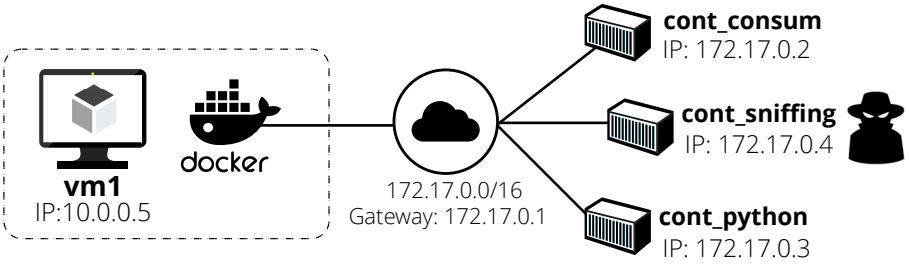

Figura 6. Ambiente de experimentação do Cenário 6.

Quanto as configurações da rede, o padrão do Docker é o estabelecimento de rotas entre as redes de um mesmo núcleo. Portanto, mesmo que um contêiner não pertença a mesma sub-rede, pacotes podem ser encaminhados de uma rede de origem diferente da rede de destino. A rede de contêineres Docker (*docker0*) dispõe de três modos de configuração: *bridge, overlay* e *macvlan* [Church 2016]. No contexto deste trabalho, escolheu-se *bridge* porque ela permite criar redes privadas, o que deve assegurar, mesmo com roteamento entre as redes do mesmo núcleo, o isolamento através das regras de bloqueio a conectividade. Como resultado, constatou-se que mesmo com o uso de *bridge*, contêineres de um mesmo núcleo em sub-redes distintas são capazes de capturar o tráfego de rede de outros contêineres e do próprio núcleo do sistema.

Com o término das análises individuais dos cenários propostos, elaborou-se a Tabela 1. De acordo com os critérios (coluna 1) estabelecidos neste trabalho, é possível identificar as vulnerabilidades (coluna 2) encontradas, seguidas de possível soluções (coluna 3 e boas práticas (coluna 4).

Tabela 1. Visão geral dos resultados da análise do trabalho

Critérios		Vulnerabilidades	Possíveis soluções	Boas práticas
Controle e limitação de recursos	Má configuração do *AppArmor*.	Escalação de privilégios.	Verificar se o mecanismo está ativado; Utilizar o profile padrão oferecido pelo Docker.	Auditar as políticas de segurança baseado no contexto necessário para a aplicação.
	Permissão indevida de capacidades do núcleo.	Negação de serviço; Execução de código arbitrário.	Utilizar apenas as capacidades do profile padrão do contêiner.	Remover as capacidades que não serão utilizadas no contexto da aplicação; Verificar a possibilidade remover a CAP_SYS_ADMIN, responsável pela concessão da maior parte dos privilégios de super usuário;
	Não filtragem de chamadas Seccomp.	Escalação de privilégios;	Utilizar o profile padrão do Docker com a permissão de entorno de 311 chamadas de sistema.	Não modificar o profile padrão;
Imagens de contêiner.		Execução de código arbitrário; Obtenção de informações privilegiadas; Corrupção de memória.	Uso de ferramentas para análise estática de imagens de contêineres (*e.g.*, Docker Security Scanning e Clair-Scanner);	Uso de imagens disponibilizadas pelo repositório oficial Docker Hub;
Comunicação entre contêineres.		ARP *Spoofing*, *Man-in-the-middle*	Cifragem do canal de comunicação; Criação de redes definidas pelo usuário.	Não utilizar a interface *docker0*; Uso de serviços de orquestração (*e.g.*, Docker Swarm).

5. Considerações

Devido a popularização em ambiente de produção compartilhados, a análise de segurança de contêineres Docker, no contexto de nuvem computacional, tornou-se essencial a comunidade científica. A escolha dos critérios de análise e a concepção dos cenários deste trabalho basearam-se nos guias de segurança públicos vigentes. Os critérios destacados foram: controle e limitação de recursos, segurança na imagem de contêineres e a comunicação segura entre contêineres. Os experimentos comprovaram vulnerabilidades em todos os critérios avaliados. Constatou-se a complexidade de configuração para garantir segurança em um ambiente multi-inquilino, a viabilidade de ataques de DoS e a exposição da rede *docker0*.

A análise relatada neste trabalho descreve os impactos gerados com a exploração das vulnerabilidades encontradas, mas, também documentou as técnicas e ferramentas para solucionar os problemas. No contexto específico de contêineres, este documento, reforça a necessidade de auditoria de políticas de segurança de forma sistemática e, se possível, automática, com o auxílio de ferramentas específicas por critério. Observou-se ainda que as restrições sugeridas no *profile* padrão do *Docker* (*docker-default*) devem ser seguidas e, caso seja necessária a adequação, que a liberação seja analisada cuidadosamente em um contexto global com o auxílio de ferramentas de auditoria. No entanto, o uso do *profile* padrão não dispensa a auditoria de segurança através de ferramentas, pois o trabalho revelou que um contêiner iniciado com *profile* padrão Docker é capaz de fazer um ataque DoS através de um `forkbomb`.

Agradecimentos: Os autores agradecem ao LabP2D, UDESC e FAPESC.

Referências

Alles, G. R., Carissimi, A., and Schnorr, L. M. (2018). Assessing the computation and communication overhead of linux containers for hpc applications. *Anais do Simpósio em Sistemas Computacionais de Alto Desempenho (WSCAD)*, 19(1/2018).

Arango, C., Dernat, R., and Sanabria, J. (2017). Performance evaluation of container-based virtualization for high performance computing environments. *CoRR*, abs/1709.10140.

Bernstein, D. (2014). Containers and cloud: From lxc to docker to kubernetes. *IEEE Cloud Computing*, 1(3):81–84.

Bui, T. (2015). Analysis of docker security. *CoRR*, abs/1501.02967.

Church, M. (2016). Understanding docker network drivers and their use cases.

CIMCOR (2017). Docker security and containerization. technical report, SIMCOR.

CLUSTERHQ (2016). Container market adoption.

CSA (2017). Security guidance for critical areas of focus in cloud computing v4.0. technical report, Cloud Security Alliance.

DOCKER (2016). Modern app architecture for the enterprise.

DOCKER (2017a). Apparmor security profiles for docker.

DOCKER (2017b). Limit a container's resources.

DOCKER (2017c). Seccomp security profile.

DOCKER (2018). Docker security.

Gao, X., Gu, Z., Kayaalp, M., Pendarakis, D., and Wang, H. (2017). Containerleaks: Emerging security threats of information leakages in container clouds. In *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 237–248.

Kozhirbayev, Z. and Sinnott, R. O. (2017). A performance comparison of container-based technologies for the cloud. *Future Generation Computer Systems*, 68:175 – 182.

LINUXCONTAINERS.ORG (2016). What's lxd?

MAN7.0RG (2017). Capabilities.

NCC GROUP (2016). Understanding and hardening linux containers. technical report, NCC Group.

OPENSTACK (2018). Kolla overview.

OPENSTACK SECURITY (2016). Exploring opportunities: Containers and openstack. technical report, OpenStack.

OPENSTACK.ORG (2017). Magnum user guide.

SANS.ORG (2015). Securing linux containers. pages 1–25.

Schanzju, C. (2017). Docker engine does not support the parameter security-opt seccomp.

Shu, R., Gu, X., and Enck, W. (2017). A study of security vulnerabilities on docker hub. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, CODASPY '17, pages 269–280, New York, NY, USA. ACM.

SYSTUTORIALS.COM (2017). Unshare (1) - linux man pages.