Uma análise de segurança no uso de contêineres Docker em máquinas virtuais

Kerolayne de S. V. de Oliveira[1], **Charles C. Miers**[1]

[1]Departamento de Ciência da Computação – Universidade do Estado de Santa Catarina
Centro de Ciências Tecnológicas – Joinville, SC – Brasil

kerolayne.oliveira@edu.udesc.br,

charles.miers@udesc.br

Abstract. *The search for new ways to data centers save money has led to new technologies, and containers are one of the most proemninent related to virtualization. However, kernel vulnerabilities pose a risk to the data and service of all entities sharing the same hardware. Therefore, it is relevant to identify and classify threats, risks, and potential vulnerabilities for the security of the container-based environment. The purpose of this article is to identify and define security issues in Docker Community 18.09.6 containers, presenting approaches to mitigate them.*

Resumo. *A procura por novas formas de economizar com data centers proporcionou a entrada de novas tecnologias no mercado e entre estas estão os contêineres. Porém, vulnerabilidades nos mecanismos do núcleo são um risco para os dados e o funcionamento do serviço de todas as entidades que compartilham o mesmo hardware. Portanto, torna-se relevante identificar e classificar ameaças, riscos e possíveis vulnerabilidades para a segurança do ambiente baseado em contêineres. O objetivo deste artigo é identificar os problemas de segurança em contêineres Docker Community 18.09.6, indicando meios de mitigação.*

1. Introdução

A virtualização é uma técnica amplamente utilizada por plataformas do tipo *Platform-as-a-Service* (PaaS) e *Infrastructure-as-a-Service* (IaaS) a fim de provisionar um ambiente isolado, seguro e escalável para usuários / organizações [Bui 2015]. A virtualização de recursos computacionais, tipicamente, passou a ser proporcionada através de duas abordagens principais: contêiner e hipervisor. Com o surgimento do Docker em 2013, a conteinerização teve uma considerável adoção por desenvolvedores e organizações [Docker 2016]. Seu uso combinado com máquina virtuals (MVs) proporciona um considerável grau de isolamento em aplicações na nuvem.

O objetivo deste trabalho é analisar a segurança do uso de contêineres para a computação em nuvem. A relevância da análise contribui para os esforços da comunidade em explorar as principais vulnerabilidades, riscos e ameaças da tecnologia de conteinerização *Docker*, sendo uma replicação aprimorada do trabalho dos mesmos autores [Miers et al. 2019] com uma versão mais recente do Docker e ferramentas. O artigo está organizado como segue. A Seção 2 trata da computação em nuvem, contêineres e segurança de contêineres. A Seção 3 descreve os critérios de análise e resultados.

2. Computação em nuvem & Contêineres

O uso de virtualização em plataformas dos tipos PaaS e IaaS tem como objetivo prover um ambiente isolado, seguro e escalável aos seus usuários [Bui 2015]. Essa tecnologia está presente em renomadas infraestruturas de computação em nuvem, *e.g.,* Amazon EC2 e Google App Engine. Através dos contêineres é possível a virtualização de aplicações implementando gestores de tarefas, escalonadores e multiplexadores de recursos. Já a partir de uma arquitetura pensada para contêineres, possibilita a execução de diversos serviços alocados em um mesmo *host*, separando assim cada função em um contêiner diferente [Panizzon et al. 2019]. A Figura 1 representa as arquiteturas possíveis de contêiner.

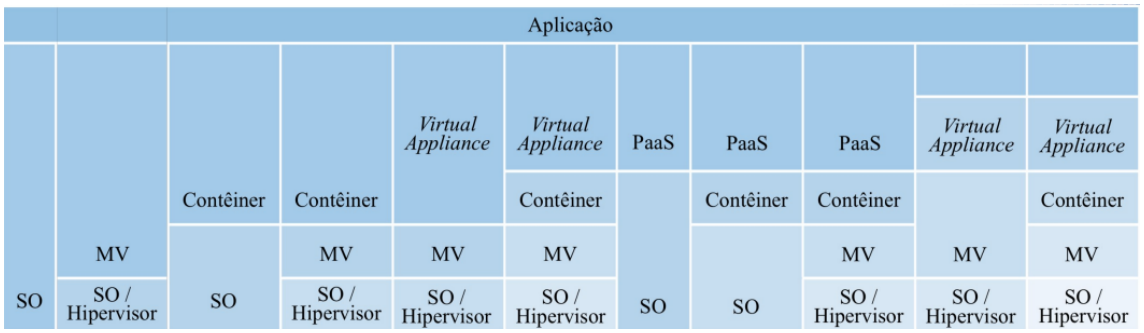

Figura 1. Arquiteturas de execução de uma aplicação [Miers et al. 2019].

Com o uso da virtualização é possível garantir algumas características importantes como: elasticidade, escalabilidade e segurança [NCC GROUP 2016]. Se as demais camadas do ambiente não estão sobrecarregadas, os contêineres são capazes de mostrar um bom desempenho comparado com o sistema operacional sobre o *bare metal*, o que provoca um maior rendimento em relação aos sistemas baseados em MV [Gao et al. 2017]. Porém, há as dúvidas sobre a segurança dos contêineres, desde o isolamento até a realização segura de operacões.

O conceito da virtualização baseada em contêineres visa utilizar os recursos do núcleo para dispor um ambiente isolado para os processos. Ao contrário da virtualização baseada em hipervisor, contêineres não abstraem o hardware do hospedeiro [Eder 2016]. Portanto, os contêineres e as MVs são vistos como técnicas de virtualização, mas são implementadas de formas diferentes. No contexto de IaaS, é possível utilizar contêineres como substituto de MVs através de soluções como *Linux Container Hypervisor* (LXD) que atuam como um hipervisor para contêineres. Além disso, contêineres e MVs são tecnologias complementares. É possível executar contêineres sob uma MV, garantindo um nível a mais de isolamento entre componentes de uma aplicação [Bernstein 2014]. Apesar de os contêineres terem se popularizado entre desenvolvedores e administradores de rede, sempre existem as preocupações sobre segurança e privacidade para a execução de múltiplos contêineres, presumivelmente pertencentes a inquilinos diferentes, no mesmo núcleo do sistema operacional [Gao et al. 2017]. [NCC GROUP 2016] salienta questões de segurança para as soluções existentes, responsáveis por reduzir a superfície de ataques à tecnologias de contêineres como Docker e *Linux Containers* (LXC). Os principais recursos compõem a virtualização baseada em contêineres: *Namespaces* do núcleo, *Control groups*, capacidades, *pivot_root* e Controle de acesso mandatório.

3. Análise de Segurança de Contêineres

De acordo com os guias de segurança, os trabalhos correlatos e os aspectos de segurança, evidencia-se a relevância em identificar e classificar as responsabilidades e mecanismos que constatem a segurança em contêineres. De acordo com as preocupações levantadas, são definidos três critérios [Panizzon et al. 2019]:

1. **Controle e limitação de recursos**: Verificar mecanismos de controle de acesso, as formas de limitação de recursos e capacidades que irão garantir o isolamento e o controle de acessos à recursos entre o núcleo e o sistema operacional (SO) e SO com contêineres vizinhos. Assegurar o isolamento e limitação de recursos é relevante para a segurança do contêiner. O estimulo para o padrão é a ativação por repetições destes mecanismos na inicialização do contêiner segundo [Docker 2019a].
2. **Segurança das imagens de contêineres**: Analisar a existência de fragilidade 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). Segundo [Shu et al. 2017], sabe-se que acima de 80% do conteúdo gráfico do repositório *Docker Hub* tem uma ou mais vulnerabilidades de alto risco, baseado no 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**: Estudar a segurança na comunicação entre contêineres que utilizam o mesmo núcleo. Segundo os padrões de segurança da [CSA 2017], é de conhecimento que os contêineres devem respeitar o perímetro de segurança para a efetiva capacidade de proteger, detectar e evitar contra ataques, assim como de sua rede virtual utilizada. Feito que isso é o contrário da configuração padrão do contêiner Docker, no qual é concedida a troca de informações sob um mesmo núcleo. Estas configurações podem resultar no aumento do risco de vazamento de informações para contêineres não familiares. Portanto, é importante assegurar que o fluxo de informações seja apenas para os contêineres desejados, reduzindo significativamente a área que pode ser afetada visto a restrição de acesso, garantindo a integridade e a confidencialidade do fluxo da rede [CIMCOR 2017].

3.1. Cenários, experimentos e resultados

Os experimentos foram efetuados em quatro cenários, todos de acordo com a configuração do ambiente de teste Figura 2. A topologia escolhida possui uma MV (`vm1 - IP 10.0.0.5`) com GNU/Linux Ubuntu 18.04 *Long Term Support* (LTS) com o *Docker Community* 18.09.6. O contêiner possui instalado o GNU/Linux Ubuntu Xenial 18.04.2 e responde no IP `172.17.0.2`.

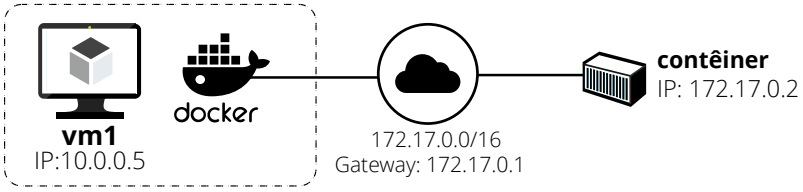

Figura 2. Ambiente de Experimentação.

- **Cenário 1**. O objetivo dessa experimentação é investigar se contêineres *Docker* possuem o mecanismo *Mandatory Access Control* (MAC) *AppArmor* configurado por padrão. O mecanismo de MAC *AppArmor* protege o *SO GNU/Linux* e aplicações reforçando políticas de segurança conhecidas como *AppArmor profile*. O Docker permite configurar um *profile* com todas as políticas de seguranças não permitidas para um contêiner específico. Caso não seja criado um *profile* personalizado, o contêiner deve carregar automaticamente o *profile* padrão denominado *docker-default* [Docker 2018a]. O comando *inspect* do Docker foi utilizado para a obtenção de informações detalhadas do contêiner [Docker 2018b]. É constatado que $7698f282e524$ é o ID abreviado do contêiner e –format é utilizado para filtrar o ID completo e o *profile* utilizado pelo contêiner. Como saída tem-se: $sha256$: $7698f282e5242af2b9d2291458d4e425c75b25b0008c1e058d66b717b4c06fa9$: $AppArmorProfile$=$docker\text{-}default$. Pode-se verificar que é utilizado o *profile* padrão do Docker. O impacto da não configuração correta das políticas de segurança pode gerar o um funcionamento inadequado, como erros nas ações de leitura e escrita que podem não ser realizadas. Na pior das hipóteses, a não restrição ao acesso de um determinado arquivo ou diretório pode afetar na segurança do núcleo do próprio contêiner e até do *SO*.
- **Cenário 2**. O objetivo dessa experimentação é verificar quais são as capacidades do núcleo que por padrão são restringidas ao contêiner (Figura 3). Para isso, é realizado a mudança do *hostname* do contêiner com as capacidades padrões e após isso, refazer o teste com a capacidade responsável concedida. Para as capacidades padrão, o resultado esperado é a limitação dessa ação dentro do contêiner devido a falta de capacidades de superusuário. Após a concessão espera-se que seja possível mudar o *hostname*.

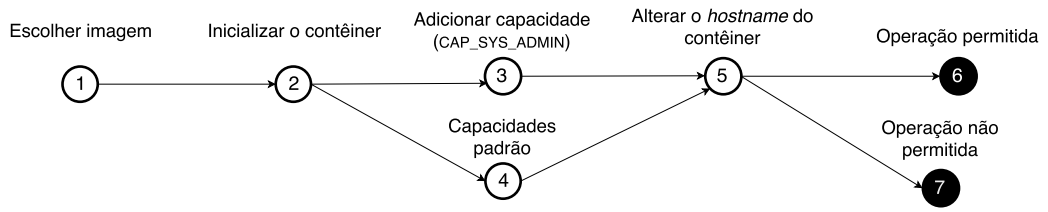

Figura 3. Fluxo de execução do experimento no Cenário 2.

No Docker, todo contêiner inicializado por padrão é configurado de modo a não possuir privilégios de superusuário, tal que o mesmo recebe um conjunto restrito de capacidades do núcleo do SO. A atualização para as versões superiores a 2.2 do núcleo, o SO fracionou os processos em objetos conhecidos como capacidade, que podem ser ativados ou desativados de forma independente [MAN7.0RG 2017]. Portanto, podem ser oferecidas as capacidades para as tarefas e não necessariamente os privilégios de superusuário. A execução do teste foi efetuado com o contêiner inicializado utilizando as capacidades padrões (*e.g., AUDIT_WRITE, KILL e NET_RAW*) do Docker e logo após foi realizado a tentativa de alterar o *hostname* do contêiner. O *CAP_SYS_ADMIN* é a capacidade do contêiner responsável por alterar o *hostname*. É possível concluir que a ação foi negada visto a falta de privilégios devido a ausência dessa capacidade. Em sequência um novo contêiner com a capacidade *CAP_SYS_ADMIN* foi habilitado e instanciado, e portanto, a realização da alteração do *hostname* foi possível. Portanto, fica claro que o mal uso das capacidades acarreta em uma falha de segurança do núcleo e do próprio contêiner. Conferir os privilégios de aplicações e serviços é definitivamente importante para assegurar a segurança e integridade dos dados de seus usuários

que compartilham a mesma unidade.

- **Cenário 3**. A partir de uma MV com a distribuição GNU/Linux Ubuntu Xenial 18.04 LTS, são inicializados dez contêineres *Docker* e atribuído uma faixa de IPs que variam entre `172.17.0.2 - 172.17.0.11`. O contêiner `cont_bomb` recebe a execução do código `forkbomb.c` na linguagem C (Figura 4). A finalidade do código é consumir toda a memória disponibilizada pelo núcleo.

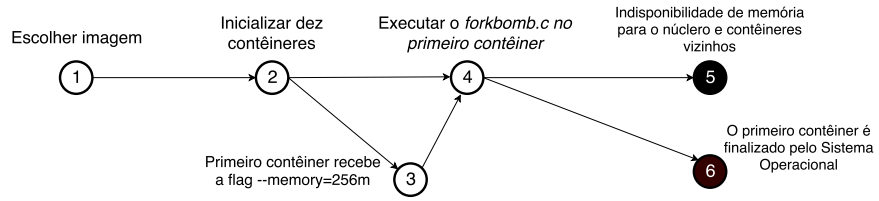

Figura 4. Fluxo de execução do experimento no Cenário 3.

Os contêineres Docker por configuração padrão não são executados com restrições a respeito do uso de capacidade do núcleo. Portanto, cada contêiner pode utilizar a quantidade necessária de recursos para a execução da aplicação dentro de seu contexto. Existem ferramentas fornecidas pelo Docker para regular o consumo de *CPU*, memória, armazenamento e operações do sistema através de *flags* na inicialização do contêiner [Docker 2018c]. A importância de não permitir que o contêiner utilize de toda capacidade disponível é crucial para que no núcleo *Linux* não seja necessário coibir funções ou ações fundamentais para a execução do sistema, visto que a exceção *Out Of Memory Exception* em sua inicialização finalizará processos em execução para aumentar a memória disponível, podendo encerrar os contêineres. Se tratando de segurança, quando não é imposto limites no uso de memória da aplicação, ataques tais como o *Denial of Service* (DoS) são bem mais suscetíveis, acarretando na indisponibilidade do serviço para todos os usuários dependentes.

O principal objetivo deste experimento é verificar a restrição em limitar o uso de memória na inicialização do contêiner. Com o objetivo de testar os mesmos, 10 contêineres foram executados com o intuito de simular um ambiente multi-usuário. Um dos contêineres foi inicializado com o código `forkbomb.c`. A rotina de execuções deste algoritmo se baseia na incessante execução de processos através do `fork()`. O algoritmo é conhecido pela utilização para ataques de DoS buscando resultar em um uso excessivo de recursos disponíveis, prejudicando o uso da plataforma por todos os seus usuários e até mesmo para tarefas do *núcleo*.

A execução desta tarefa afetou drasticamente o uso da plataforma por todos os usuários e até mesmo a MV. Portanto, fica claro a importância em restringir a memória para cada contêiner inicializado. Limitando a memória restringindo o seu uso no contêiner, o teste realizado utilizando o `forkbomb.c` não deixou os usuários vulneráveis.

A restrição no contêiner do uso de memória fez com que a execução do algoritmo não tivesse influência sobre a disponibilidade dos recursos para os contêineres em mesma unidade, o próprio contêiner finalizou a aplicação quando o limite definido de 256MB foi atingido.

- **Cenário 4**. O objetivo desse experimento é verificar se o mecanismo *Seccomp*, responsável pela filtragem de chamadas da *Application Programming Interface* (API) do núcleo está configurado por padrão e investigar o impacto da sua desativação na segurança do contêiner (Figura 5).

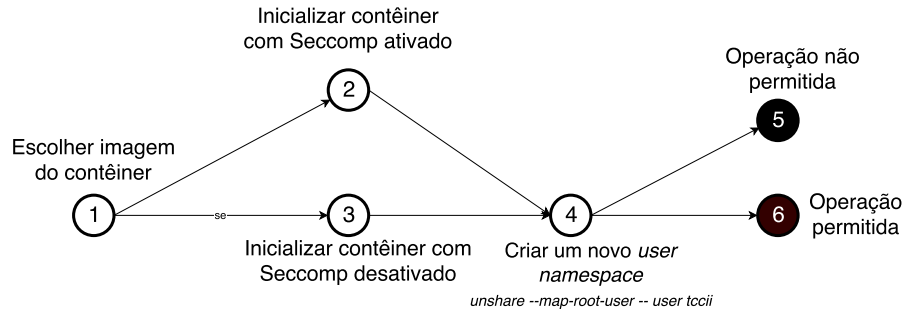

Figura 5. Fluxo de execução do experimento no Cenário 4.

É possível observar as ações de ativar ou desativar o mecanismo *Seccomp*, na qual impactará no resultado da operação. É relevante mencionar que no SO Apple MacOS Sierra, o Docker Community Edition não suporta a ativação do *Seccomp* [Schanzju 2017]. O problema motivou para a incorporação desse cenário de teste. Continuando, o esperado desse experimento é determinar se o *Seccomp* é utilizado por padrão na imagem da MV base do presente Cenário.

Na plataforma do Docker existe um mecanismo que, através do *profile* definido na inicialização, possibilita a personalização do sistema quando necessário. O *profile* padrão do Docker desativa cerca de 44 chamadas de sistema das mais de 300 disponibilizadas pela API [Docker 2019b]. O objetivo desse experimento é verificar o impacto na segurança do contêiner quando o *seccomp* é desativado. Possuindo 40 chamadas de sistema em que o contêiner limita, a chamada do sistema *unshare* foi a base para testes. A *unshare* é a chamada utilizada para replicar novos *namespaces* eliminando a necessidade de iniciar um novo processo, atuando para que outros processos ou *threads* não estejam vinculados ao contexto compartilhado [Docker 2019b]. A primeira parte do teste foi inicializado com um contêiner utilizando o *profile* padrão do *seccomp*. Já dentro do contêiner, foi realizado essa chamada em conjunto com a *flag* `unshare --map-root-user --user` para criar um novo *user namespace*. Essa *flag* concede ao novo *namespace* capacidades com um certo grau de privilégio, mesmo quando a execução do contêiner é de modo não-privilegiado [SYSTUTORIALS.COM 2019]. Como resultado da execução do comando, foi obtido como retorno a mensagem: *Operation not Permitted*. O motivo é que o *profile* padrão do Docker filtra essa chamada bloqueando a operação.

Na segunda parte do experimento foi desativada a filtragem de sistema *seccomp* e repetido o processo da chamada do método *unshare*. Como resultado, foi obtido êxito na criação do novo *namespace* do usuário. Como garantia de êxito na operação, foi executado o comando `whoami`, tal que é obtido como retorno: `root`. Isso quer dizer que o novo *namespace* foi criado com sucesso e ainda, para esse novo usuário foi concedido operações de superusuário. A documentação oficial do Docker recomenda a não alteração do *profile* padrão do *seccomp* já que configurações indevidas podem aumentar a superfície de ataque do tipo escalação de privilégios.

Percebe-se, que algumas das vulnerabilidades identificadas em [Miers et al. 2019] não foram identificadas na versão analisada do Docker neste artigo, mostrando um evolução em termos de segurança. Entretanto, alguns dos problemas persistem e alguns novos foram revelados:

- Quanto a vulnerabilidade escalação de privilégios, é fundamental verificar se o mecanismo está ativado. Utilizar o *profile* padrão oferecido pelo Docker.

- Relativo a ataques de negação de serviço por esgotamento de recursos (*e.g.*, memória principal), recomenda-se definir explicitamente as capacidades no *profile*, visto que alguns *profiles* padrão do contêiner não colocam limites.
- A Execução de código arbitrário pode levar a obtenção de informações privilegiadas e até mesmo corrupção de memória do sistema. O uso sistemático de ferramentas para análise estática de imagens de contêineres (*e.g.*, Docker Security Scanning, Blackduck) podem identificar vulnerabilidades conhecidas para que configurações sejam corrigidas ou software atualizado pra versão não vulnerável.
- Questões de rede em contêineres merecem mais estudo e atenção, ataques de *ARP Spoofing* e *Man-in-the-Middle* [Nath Nayak and Ghosh Samaddar 2010] dependem de segurança a nível de aplicação por falta de mecanismos padrões de segurança no Docker. Cifragem do canal de comunicação e criação de redes virtualizadas podem mitigar esse problema.

4. Considerações & Trabalhos futuros

A análise apontou que o Docker evolui em relação aos critérios adotados para realizar a análise de segurança. A definição dos critérios para a experimentação foi definida com base nos guias de segurança, na abordagem utilizada pela comunidade e nos diversos pontos de comunicação entre contêineres. Dessa forma, os critérios definidos foram: controle e limitação de recursos, segurança na imagem de contêineres e a comunicação segura entre contêineres. Com base nesses critérios, a análise apoiou a preocupação com a segurança da conteinerização em um ambiente de produção. Pelo resultado da experimentação, foi evidenciado a quantidade considerável de vulnerabilidades encontradas nos componentes do contêiner. Entre os resultados dos experimentos, critério de controle e limitação de recursos evidenciam a complexidade de garantir a segurança num ambiente multi-inquilino, devido a quantidade de políticas de seguranças que o núcleo disponibiliza. Já no experimento que explorou as restrições de recursos do núcleo pode-se visualizar a suscetividade do contêiner em ataques do tipo DoS e a sua implicação para a disponibilidade tanto do contêiner atacado, quanto de vizinhos e do próprio núcleo. No critério da segurança na comunicação entre contêineres evidenciou a exposição de dados para todos os contêineres vinculados à interface de rede de um contêiner Docker. Neste caso, foi possível a captura do tráfego de rede com o uso da ferramenta TCPDump em contêiner dentro de uma mesmo *pod*. No geral, grande parte das soluções para os problemas encontrados podem ser distribuídos em ações como: auditar políticas de segurança, caso não seja necessário, não alterar as regras impostas por padrão pelo Docker para a filtragem de chamadas do sistema e configurar corretamente a comunicação de contêineres através de redes definidas pelo usuário.

Como trabalhos futuros está sendo realizada uma análise similar usando o serviço Magnum do OpenStack. Além disso, são necessários mais estudos sobre a segurança na comunicação entre contêineres nos seguintes cenários: dentro de um mesmo *pod*, entre *pods* distintos em um mesmo inquilino e entre um *pod* distribuído em mais de um *host*.

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

5. Referências

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.

CIMCOR (2017). Docker security and containerization. Technical report, CIMCOR.

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. https://www.docker.com/sites/default/files/caaSwhitepaper_V6_0.pdf.

Docker (2018a). Apparmor security profiles for docker. https://docs.docker.com/engine/security/apparmor/.

Docker (2018b). Docker inspect. https://docs.docker.com/engine/reference/commandline/inspect/.

Docker (2018c). Limit a container's resources. https://docs.docker.com/engine/admin/resource_constraints/.

Docker (2019a). Docker security. https://docs.docker.com/engine/security/security/.

Docker (2019b). Seccomp security profile. https://docs.docker.com/engine/security/seccomp/.

Eder, M. (2016). Hypervisor- vs. container-based virtualization. *Network Architectures and Services*, pages 11–17.

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.

MAN7.0RG (2017). Capabilities. http://man7.org/linux/man-pages/man7/capabilities.7.html.

Miers, C., Panizzon, G., Oliveira, K., Pillon, M. A., Koslovski, G., and Mimura, N. (2019). Uma análise de segurança no uso de contêineres Docker em nuvens IaaS OpenStack. In *Computer on the Beach 2019*.

Nath Nayak, G. and Ghosh Samaddar, S. (2010). Different flavours of man-in-the-middle attack, consequences and feasible solutions. In *2010 3rd International Conference on Computer Science and Information Technology*, volume 5, pages 491–495.

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

Panizzon, G., Battisti, J. H. F., Koslovski, G. P., Pillon, M. A., and Miers, C. C. (2019). A Taxonomy of container security on computational clouds: concerns and solutions. *Revista de Informática Teórica e Aplicada*, 26(1):47–59.

Schanzju, C. (2017). Docker engine does not support the parameter security-opt seccomp. https://github.com/docker/for-mac/issues/1946.

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 (2019). Unshare (1) - linux man pages. https://www.systutorials.com/docs/linux/man/1-unshare/.