SARIK - framework para automatizar a segurança em ambientes de orquestração kubernetes

Jonathan G. P. dos Santos[1], **Geraldo P. Rocha Filho**[2], **Vinícius P. Gonçalves**[1]

[1]Departamento de Engenharia Elétrica, Universidade de Brasília – UnB
Caixa Postal 4466 CEP 70910-900 – Brasília – DF – Brasil.

[2]Departamento de Ciência da Computação, Universidade de Brasília – UnB
Caixa Postal 4466 CEP 70910-900 – Brasília – DF – Brasil,

{jonathanti, geraldof, vpgvinicius}@unb.br

Abstract. *This paper demonstrates SARIK, an automatic security framework of Iptables' rules in environments of Kubernetes' orchestration. SARIK was developed in shell script and used Microsoft Visual Studio IDE as a tool for the development of the software; the language was chosen because of its presence in the majority of the cloud platforms, therefore, it does not require the need of dependencies' installation to the framework functionality. Thus, this framework allows fast protection of the nodes layer through the automatic setting of firewall's rules on uncountable PODs (PODs are the smallest and basic objects implementable on the Kubernetes) contained in a cluster. By blocking and opening ports, SARIK inspects each node, storing their ports and blocking the ones that can bring risk to the containers. The developer does not need to protect their containers, this is a task done by SARIK. The functioning of SARIK is evaluated in a controlled environment with minikube and with a voting application containing deployment, namespace, and services. With the use of SARIK, developers reach a reduction in the manual work, due to the automatization of Iptables' rules and, with that, the node layer protection is guaranteed.*

Resumo. *Este artigo apresenta o SARIK, um framework de segurança automática de regras de Iptables em ambientes de orquestração Kubernetes. O SARIK foi desenvolvido em shell script e foi utilizada o IDE Microsoft Visual Studio como ferramenta para codificação do software, a escolha da linguagem está relacionada por sua presença na maioria das plataformas de cloud. Portanto, não haverá a necessidade de instalação de dependência para que o framework funcione. O SARIK possibilita a rápida proteção da camada node através da configuração automática de regras de firewall nos inúmeros PODs (Pods são os menores e mais básicos objetos implantáveis no Kubernetes) contidos em um cluster. Por meio de bloqueio e abertura de portas, o SARIK realiza uma inspeção em cada node, armazenando suas portas e bloqueando aquelas que podem trazer riscos aos containers. Em outras palavras, o desenvolvedor não precisa proteger seus containers, essa tarefa é feita pelo SARIK. O funcionamento do SARIK é avaliado em um ambiente controlado com minikube e uma aplicação de votação contendo deployment, namespace e services. Com a utilização do SARIK, desenvolvedores alcançam a redução de trabalho manual devido a automatização das regras de iptables e com isso, a proteção da camada node é garantida.*

1. Introdução

A crise sanitária mundial provocada pela pandemia do COVID-19 fez aumentar o uso de serviços em nuvem. Diante disso, as três maiores empresas fornecedoras da tecnologia – AWS, Microsoft *Azure* e Google *Cloud* –, viram sua demanda aumentar consideravelmente. No primeiro quadrimestre de 2020, o investimento em *cloud computing* foi ampliado em 37% [Alley 2020] acompanhando o movimento de expansão dos serviços. Consequentemente, cresceu também o número de ataques cibernéticos nesses ambientes.

Nas últimas duas décadas, as tecnologias de *containers* têm crescido exponencialmente, em especial o Docker e Kubernetes, como principais líderes do mercado. Para [Souza et al. 2016] *containers* são um método de virtualização que permite múltiplas instâncias de sistemas executadas em um único *host* enquanto dividem o mesmo *kernel*. Para [da Costa Cordovil et al. 2020] a adoção do uso do Docker está relacionada ao total isolamento de recursos de forma independente e flexível, além de ter nós com diferentes tipos de sistemas operacionais compatível com qualquer protocolo e com possibilidade de emular diferentes tipos de dispositivos. O kubernetes foi desenvolvido com um maior foco na experiência de desenvolvedores que escrevem aplicativos executados em um *cluster*. O principal objetivo do projeto é facilitar a implantação e o gerenciamento de sistemas distribuídos [Burns et al. 2016]. Nesse cenário, usuários mal intencionados exploram os ambientes conteinerizados por meio de falhas no armazenamento das imagens em repositórios oficiais.

Nos relatórios de 2018 e 2019 das empresas de segurança Kromtech [Silva 2018] e Snyk [Vermee and Henry 2019], foram demonstradas as principais vulnerabilidades das imagens contidas no Docker Hub que é apresentado na Figura 1 do relatório *Shifting Docker security left* da snyk[Vermee and Henry 2019]. No relatório da kromtech sobre *security center in Cryptojacking invades cloud. How modern containerization trend is exploited by attackers*, a empresa demonstrou que foram espalhadas 17 imagens maliciosas no repositório do Docker Hub para mineração de criptomoedas. Nesse sentido, com o crescente aumento de plataformas de orquestração mal configuradas e de acesso fácil, como o kubernetes, pode-se facilitar aos criminosos a exploração de brechas de segurança em ambientes conteinerizados que são factíveis de serem exploradas, escondendo *scripts* nas imagens e, por sua vez, usando tais imagens para a mineração de criptomoedas.

Para [Simone 2019], o relatório da Snyk é certamente preocupante, entretanto pode não representar a parte mais difícil do problema. Em muitos casos, corrigir vulnerabilidades, em uma imagem do Docker, é tão fácil quanto reconstruir a imagem utilizando versões mais seguras. De acordo com a Snyk, até 44% das imagens do Docker continham vulnerabilidades que já haviam sido removidas em versões mais recentes de sua imagem base. O que parece ser mais preocupante é a falta de entendimento sobre a segurança por parte dos desenvolvedores, que não parecem estar cientes da criticidade de seu papel. A pesquisa mostrou que 80% dos desenvolvedores não testam suas imagens durante o desenvolvimento, enquanto metade não examina suas imagens em busca de vulnerabilidades.

Segundo os autores [Vermee and Henry 2019], a melhor abordagem para lidar com as vulnerabilidades de imagem do Docker se baseia em três práticas principais, de acordo com Snyk. Primeiro, como regra de higienização, é importante começar com uma imagem menor do Docker, disponível para um determinado propósito, e não adicionar

nenhum pacote que não seja necessário. Em segundo lugar, as imagens devem ser escaneadas regularmente, tanto durante o desenvolvimento quanto em produção. Por fim, as imagens devem ser recriadas como parte de um pipeline de CI/CD[1] e as compilações de vários estágios devem ser preferidas, pois ajudam a otimizar suas imagens.

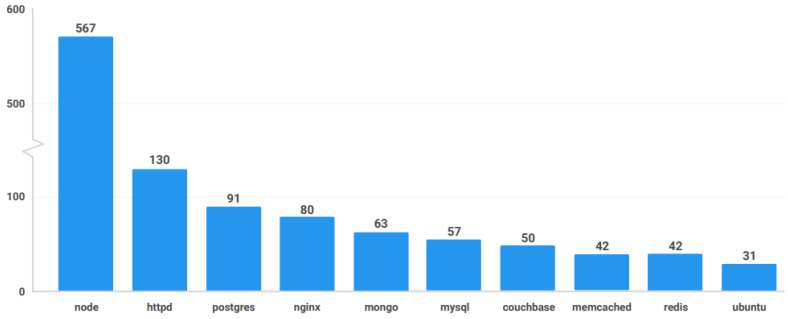

Figura 1. Gráfico das vulnerabilidades do relatório da Snyk de 2019 [Vermee and Henry 2019]

Algumas ferramentas vêm sendo desenvolvidas para dar mais segurança ao ambiente de *container* e orquestração. Por exemplo, o *framework* Tocker, foi criado para restringir a comunicação entre *containers* Docker ao menor nível, bloqueando as portas desnecessárias dos *containers* com a introdução de *firewalls* [Balabanian and Henriques 2019]. Kulathunga [Kulathunga 2021], propôs um modelo de segurança dinâmico em plataformas de orquestração a partir de um sistema de detecção de intrusão – IDS que realiza o monitoramento do tráfego de acordo com as categorias de aplicativos em *namespaces* relevantes. Para isso, foi introduzido um novo componente chamado operador-IDS que estende a API do kubernetes ao nível de monitorar esse tipo de comportamento. Entretanto, o trabalho de Balabanian não explorou cenários em cluster e não aferiu o comportamento das conexões para criar regras de firewall. E, Kulathunga não aplicou regras de proteção interna aos PODs.

Com intuito de avançar nessa frente de pesquisa, este trabalho propõe o SARIK, um *framework* de **S**egurança, **A**utomática, de **R**egras, de **I**ptables em **K**ubernetes, para aferir o comportamento das conexões e criar regras de *firewall* personalizadas nos PODs. O objetivo do trabalho foi criar um *framework*, chamado SARIK, que acessa cada POD presente nos nodes do *cluster* e através de regras de *iptables* personalizadas acessa cada POD e insere regras nesses *containers*. Dessa forma, espera-se que a ferramenta desenvolvida, possa mitigar ataques em ambientes de *cluster*. Portanto, este trabalho possui a seguinte contribuição: (i) restringir a comunicação entre os PODs adjacentes no *cluster* ao mínimo por meio de regras de *firewall*.

Este artigo está organizado da seguinte maneira. A Seção 2 apresenta a descrição da ferramenta mostrando a escolha da linguagem e desafios na solução apresentada. A Seção 3 descreve o roteiro de demonstração da ferramenta em um caso de uso. A Seção 4 apresenta as discussões sobre a ferramenta. A Seção 5 apresenta as conclusões e trabalhos futuros.

[1]CI/CD, *continuous integration/continuous delivery*, é um método para entregar aplicações.

2. Descrição da ferramenta

No desenvolvimento do *framework* SARIK, optou-se pela linguagem *shell script* por estar presente na maioria dos sistemas Unix e nas plataformas de computação em nuvem. Em suma, nas distribuições dos sistemas operacionais Linux, o interpretador shell já estará presente e não haverá a necessidade de instalação de dependência para que o *framework* funcione. Além de ser uma poderosa linguagem, a escolha do *shell script* está relacionada a velocidade que os comandos kubectl irão retornar em comparação a outras linguagens.

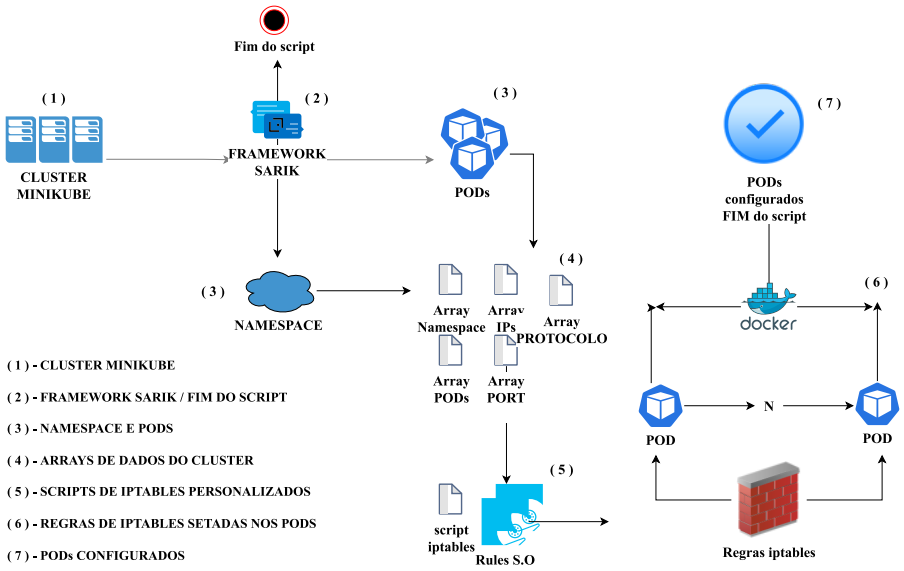

Figura 2. Diagrama de execução do SARIK

A Figura 2 apresenta o diagrama de execução do SARIK, dividido em sete etapas que são descritos a seguir: Nas etapas (1) e (2) são realizados inúmeros testes para verificar se o *framework* pode iniciar a execução do *script*, ou seja, o framework realiza uma condição para saber se os *softwares* Docker e Minikube ou Kubernetes estão instalados no *cluster*. Dependendo do valor retornado, o *script* prossegue com a configuração ou uma mensagem é apresentada ao usuário requerendo a instalação dos *softwares* e o *script* é finalizado. Na etapa (3), o *framework* faz o mapeamento das configurações do *cluster* e armazena esses valores em variáveis ou em *array*. Na etapa (4), após o armazenamento dos dados nas variáveis ou nos *arrays*, os dados que retornam com quebras de linhas são manipulados usando a função *readarray* que armazena os dados em índices de *array*. Por exemplo, o conteúdo da linha 1 é armazenado no índice 0 do *array* e esse processo é feito para todas as quebras de linhas. Na etapa (5), o *framework* utiliza *script* de *iptables* específicos de cada sistema operacional disponível na pasta RuleSO e faz a customização das regras conforme os dados que foram extraídos do mapeamento, alterando os IPs, protocolos, portas e bloqueando os demais protocolos. Na etapa (6), *framework* SARIK acessa cada POD e atualiza a lista de repositórios, faz a instalação do *firewall iptables* e em seguida seta as regras personalizadas para esse POD. Na etapa (7), o *script* finaliza a configuração de *firewall iptables* nos PODs.

Na estrutura do código, foram feitas divisões em cada seção do trecho que o *script* será executado, a saber: Cabeçalho bem estruturado, seção de variáveis, teste de instalação de dependências de software, funções e execução.

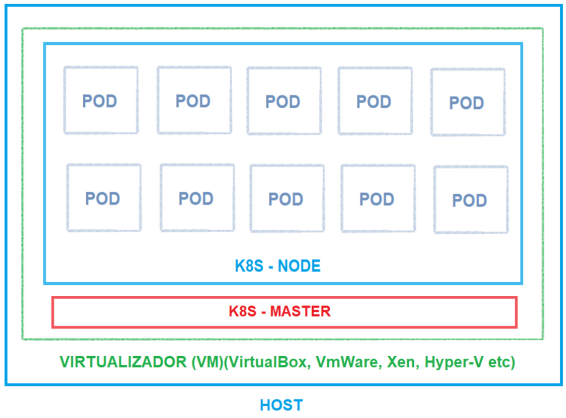

Figura 3. Arquitetura kubernetes

2.1. Arquitetura do kubernetes e do SARIK

Na Figura 3, é apresentada arquitetura padrão do kubernetes ou mais conhecido K8S. Os principais componentes que compõe o SARIK que estão associados ao *cluster* kubernetes são:

- ***Clusters***: é um grupo de instâncias que executam o Kubernetes.
- ***Master***: o kubernetes ao ser iniciado em um *cluster* que tenha mais de um nó ou mais de uma máquina, é necessário que um nó seja o *manager* (*master*), este é encarregado em verificar qual o status de saúde dos PODs, ou seja, será o gerenciador de todo o cluster.
- **Nodes**: é um nome dado para cada *host* do *cluster*.
- **Kubelet**: agente que executa nos *hosts* do *cluster*.
- **POD**: é a menor unidade de um *cluster*, nada mais é que um *container* ou um conjunto de *containers* com a mesma função.
- ***Namespace***: é a segmentação do *cluster*, em outras palavras, através desse componente é possível fazer a divisão do *cluster* em dois ou mais ambientes isolados, dessa forma, limitando os recursos computacionais de cada *cluster*.
- ***Services***: caso algum POD seja finalizado por qualquer problema, o *replication controler* vai ser responsável em subir uma nova versão do POD, nesse contexto, o *services* é responsável por monitorar essa ação.

A proposta da arquitetura do SARIK é ser totalmente isolado dos componentes do K8S. Em outras palavras, o SARIK acessa cada POD presente nos nodes do *cluster* e através de regras de *iptables* vai executando uma lista de comandos já padronizadas para cada sistema operacional do *container*. Como pode ser visualiado na Figura 3 o SARIK somente acessa os PODs e faz essa execução de comandos em cada um dos *containers*.

3. Roteiro de Demonstração

Nesta seção é apresentada a demonstração do uso do SARIK em um ambiente controlado com uma aplicação de votação incluída no pacote do código fonte do sistema. O código fonte [2], site [3], manual [4] e vídeo [5] de demonstração estão disponíveis em

[2]Código fonte: https://github.com/jonathamgg/sarik
[3]Website: https://sarik.org/
[4]Manual: https://sarik.org/manual/
[5]Vídeo: https://sarik.org/videos/

https://github.com/jonathamgg/sarik. O roteiro de uso do *framework* é dividido em duas partes: (1) criação do cenário de simulação e (2) simulação do *framework* configurando as regras de iptables nos PODs.

Para realização da demonstração do *framework* SARIK, foi utilizado um notebook pessoal com as seguintes configurações, que atua como um *cluster* kubernetes nos nodes/PODs presentes nessa infraestrutura: **Processador:** Intel Core i5-3360M CPU @ 2.80 GHz; **Memória:** 16GB DDR3; **Interface de rede:** Intel 82579LM GigabitEthernet; **Armazenamento:** SSD 256GB; **Sistema Operacional:** Ubuntu Mint

3.1. Criação do cenário da simulação

Para realizar a codificação do SARIK foi criado um cenário semelhante ao mundo real, com a seguinte estrutura para um sistema de aplicação de votação, contendo: *Deployements* (responsável pela criação dos PODS), *Namespaces* (responsável pela criação da rede interna de comunicação) e *Services* (responsável pela liberação das portas dos serviços do *cluster*).

Em cada pasta da aplicação foram configuradas com arquivo .yaml e criada uma estrutura da aplicação de votação. Por exemplo, na pasta de *deployements* temos cinco arquivos de configuração que são responsáveis por gerar os PODS responsáveis em oferecer os serviço que cada POD irá disponibilizar. Ou seja, para cada POD este disponibiliza o serviço de banco de dados ou até aplicação em si. No *deployements* foi pensado em usar N sistemas operacionais diferentes, para que o SARIK tenha uma quantidade maior de versões de sistema operacional para configurar automaticamente as regras de iptables. Para esta pesquisa foram escolhidas as seguintes distribuições do Linux, Alpine e Debian.

Na pasta *namespaces* é responsável pela criação da rede de comunicação interna dos PODs, ou seja, os PODs não se comunicam por meio de endereço IP, mas pelo nome do *namespace* atribuído no arquivo vote.yaml. Convenientemente, a escolha da rede é vote.

A pasta *services* é responsável pela liberação das portas de comunicação das aplicações, ou seja, foram configurados quatro serviços de liberação, por exemplo, a porta 5432 para banco de dados, a porta 6379 para o redis, as portas da aplicação de votação: 5000 e 31000 e a porta da aplicação do resultado: 5001 e 31001.

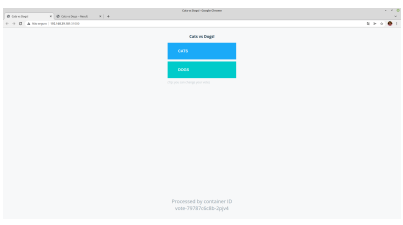

Figura 4. Sistema de votação

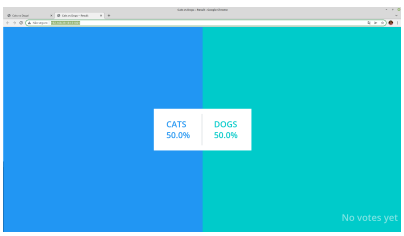

Figura 5. Sistema de resultado

As Figuras 4 e 5 apresentam o funcionamento do sistema de votação criado. Na Figura 4 é a interface do usuário e na Figura 5 é a interface do administrador, após a etapa de criação o sistema passará pela etapa de configuração de iptables nos PODs.

3.2. Simulação do framework

O *framework* SARIK, inicialmente faz uma checagem no *cluster* buscando os *softwares* docker, minikube ou kubernetes e caso estes estejam instalados o *script* continua com a execução do programa. Caso algum desses softwares não esteja instalado o script é interrompido e a seguinte mensagem é apresentada caso o docker não esteja instalado ”Precisa instalar o docker, por favor, instale.”ou minikube/kubernetes não esteja ”Precisa instalar o minikube ou kubectl, por favor, instale.”.

Para que o SARIK consiga encontrar os PODs no *cluster*, uma série de condições devem ser satisfeitas. Primeiro, realiza-se uma contagem de node no *cluster* armazenando esse valor em uma variável. Segundo, faz-se uma varredura para encontrar qual *namespace* é definido na estrutura de rede do *cluster*. Terceiro, com o nome do *namespace* encontrado, podemos listar os PODs atrelados a esse *namespace* e armazenando-os numa lista de valores em um arquivo, após isso, realiza-se o armazenamento dessa lista em um *array*. Quarto, para cada POD é atribuído um endereço IP. Com o nome do POD e seu *namespace* é realizada a captura dessas informações e inserida em um *array*, a mesma coisa é feita com as portas e protocolos de serviços de cada POD. Em cada uma dessas condições são feitas expressões regulares que fazem o trabalho de limpeza das saídas dos comandos, deixando somente o valor buscado.

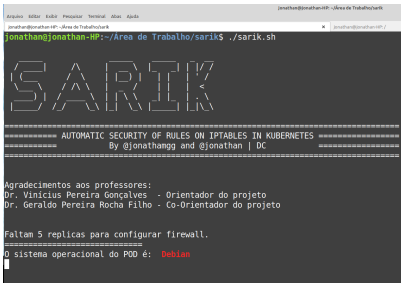

Figura 6. Apresentação SARIK

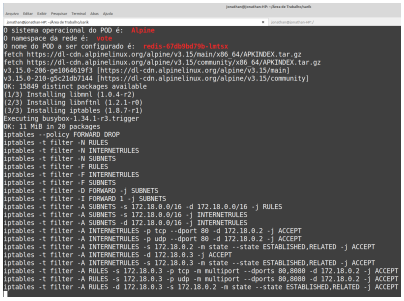

Figura 7. Configuração de iptables nos PODs

Como apresentado na Figura 6 o SARIK realiza os passos descritos anteriormente e em seguida utiliza um *loop* que vai decrementando o contador até zero, ou seja, o contador recebe o número de PODs contidos no *cluster* e realiza a configuração em cada POD, importante informar que é feita uma checagem em cada POD para ver qual o sistema operacional aquele POD foi instanciado, assim, o SARIK utiliza de um *CASE* e compara qual o sistema operacional é do *container* e executa um *script* específico de *iptables* para aquele POD que se encontra na pasta RuleSO, para esse trabalho temos dois *script*s: alpine.cf e debian.cf que contém uma série de regras personalizadas para o sistema operacional do POD. Na Figura 7, percebemos que a configuração das regras de *iptables* é feita e o *framework* mostra o nome do POD, *namespace* e qual o sistema operacional desse POD. Ao final da(s) etapa(s) o sistema imprime uma mensagem de configuração concluída com sucesso.

4. Discussão

A proposta do SARIK é configurar automaticamente todos os PODs pertencentes a uma rede de *cluster* kubernetes com regras de *iptables* personalizadas aferindo o comportamento das conexões existentes e realizando uma série de expressões regulares para então

configurá-los nesse ambiente. Através do *framework* proposto, desenvolvedores podem se concentrar nas partes fundamentais: aplicação, rede e serviços. Por exemplo, imagine um DEV (desenvolvedor) acessando um POD, atualizando o sistema, instalando e configurando cada linha de regra no *iptables* e encerrando a conexão no POD. Tal ação deve ser feita por PODs em um *node* e dependendo do tamanho da estrutura a tarefa será longa e há chance de erro humano nessa operação. O SARIK realiza essas ações de forma rápida, simples e eficiente em ambientes de *cluster* que tenham grandes quantidades de PODs, evidenciando as vantagens de escolhe-lo como solução para configuração de *clusters*.

5. Conclusão e Trabalhos Futuros

Nesse trabalho foi desenvolvido o SARIK, um *framework* para configuração automática de regras de *iptables* em ambientes de orquestração de *container* kubernetes. A motivação para o desenvolvimento do SARIK foi proteger a camada POD que abriga os diversos *containers* a partir de um *cluster* kubernetes, dessa forma, o *framework* infere o comportamento das conexões e cria regras de *firewalls* personalizadas para cada POD.

O SARIK foi apresentado em um ambiente controlado com minikube e testada também na *Google Cloud Platform* e na Microsoft *Azure*. A segurança promovida pelo *framework* aumenta consideravelmente a segurança na camada POD, portanto, podemos concluir que o SARIK apresentado nesse trabalho consegue minimizar ataques entre os PODs adjacentes em um *cluster*. Como trabalhos futuros, listamos o seguinte ponto que precisa ser averiguado e abordado no futuro: Em um ambiente padrão kubernetes, após a exclusão de um POD, o *framework* SARIK não é capaz de verificar se o novo POD está configurado com as regras de *iptables*, portanto, um mecanismo de detecção de exclusão de POD é interessante de ser desenvolvido para que o SARIK seja mais autônomo em verificar o status dos PODs.

Referências

[Alley 2020] Alley, A. (2020). Cloud providers see ”aggressive”growth amidst covid-19 pandemic.

[Balabanian and Henriques 2019] Balabanian, F. and Henriques, M. (2019). Tocker: framework para a segurança de containers docker. In *Anais Estendidos do XIX Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais*, pages 145–154. SBC.

[Burns et al. 2016] Burns, B., Grant, B., Oppenheimer, D., Brewer, E., and Wilkes, J. (2016). Borg, omega, and kubernetes. *Communications of the ACM*, 59(5):50–57.

[da Costa Cordovil et al. 2020] da Costa Cordovil, M. G., Farias, F. N. N., and Abelém, A. J. G. (2020). vsdnemul: Emulando de redes definidas por softwares através de contêineres docker. In *Anais Estendidos do XXXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 49–56. SBC.

[Kulathunga 2021] Kulathunga, R. (2021). *Dynamic security model for container orchestration platform*. PhD thesis.

[Silva 2018] Silva, F. (2018). Docker: como hackers estão explorando conteinerização.

[Simone 2019] Simone, S. D. (2019). Cenário de segurança do ecossistema docker e melhores práticas.

[Souza et al. 2016] Souza, J., Santos, A., Bandini, M., Klôh, H., and Schulze, B. (2016). Rufus: Ferramenta para o gerenciamento de infraestrutura para a execuç ao de aplicaç oes em containers.

[Vermee and Henry 2019] Vermee, B. and Henry, W. (2019). Shifting docker security left.