UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

LUIZ RICARDO SIQUEIRA RODRIGUES NASCIMENTO

UM ESTUDO DO IMPACTO DA APLICAÇÃO DAS MELHORES PRÁTICAS DE CONFIGURAÇÃO EM SISTEMAS BASEADOS EM KUBERNETES USANDO KUBE-BENCH

CAMPINA GRANDE - PB

2021

LUIZ RICARDO SIQUEIRA RODRIGUES NASCIMENTO

UM ESTUDO DO IMPACTO DA APLICAÇÃO DAS MELHORES PRÁTICAS DE CONFIGURAÇÃO EM SISTEMAS BASEADOS EM KUBERNETES USANDO KUBE-BENCH

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Professor Dr. Reinaldo Cézar de Morais Gomes.

CAMPINA GRANDE - PB

2021

N244e Nascimento, Luiz Ricardo Siqueira Rodrigues.
Um estudo do impacto da aplicação das melhores práticas de configuração em sistemas baseados em Kubernetes usando Kube-bench. / Luiz Ricardo Siqueira Rodrigues Nascimento. – 2021.

10 f.

Orientador: Prof. Dr. Reinaldo Cézar de Morais Gomes.
Trabalho de Conclusão de Curso - Artigo (Curso de Bacharelado em Ciência da Computação) - Universidade Federal de Campina Grande; Centro de Engenharia Elétrica e Informática.

1. Container runtimes. 2. Configuração em sistemas. 3. Kubernetes. 4. Kube-bench – ferramenta e código aberto. 5. Contêineres. 6. Orquestradores de contêineres. I. Gomes, Reinaldo Cézar de Morais. II. Título.

CDU:004(045)

Elaboração da Ficha Catalográfica:

Johnny Rodrigues Barbosa
Bibliotecário-Documentalista
CRB-15/626

LUIZ RICARDO SIQUEIRA RODRIGUES NASCIMENTO

UM ESTUDO DO IMPACTO DA APLICAÇÃO DAS MELHORES PRÁTICAS DE CONFIGURAÇÃO EM SISTEMAS BASEADOS EM KUBERNETES USANDO KUBE-BENCH

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

BANCA EXAMINADORA:

Professor Dr. Reinaldo Cézar de Morais Gomes

Orientador – UASC/CEEI/UFCG

Professora Dra. Lívia Maria Rodrigues Sampaio Campos

Examinador – UASC/CEEI/UFCG

Professor Tiago Lima Massoni

Professor da Disciplina TCC – UASC/CEEI/UFCG

Trabalho aprovado em: 20 de Outubro de 2021.

CAMPINA GRANDE - PB

ABSTRACT

The portability and ease to deploy and manage software and services while employing containers, cause this kind of platform to be broadly utilized in the software development and system administration communities. The default settings of container runtimes on container orchestrators are not ideal for the security requirements of the systems to be deployed, but are in many cases left as they came for practical reasons or lack of attention to detail from the system administrator on the occasion of the deployment. This increases in a huge scale the risk to security issues on the environment, such as that in many occasions it's the path that is used to actively violate the data and services hosted on cloud environments or in any other way connected to the network. Thus, a proposal of a set of basic settings to be implemented on production environments was built, based mainly on the recommended security settings indicated on the Kubernetes documentation, through tests run over an environment simulating a production one. The open source application Kube-bench, well known and of ample use in the security and system administration communities, was employed here to analyze the impacts of changes made.

Um estudo do impacto da aplicação das melhores práticas de configuração em sistemas baseados em Kubernetes usando Kube-bench

Luiz Nascimento
luiz.nascimen@ccc.ufcg.edu.br
Universidade Federal de Campina Grande
Campina Grande, Paraíba, Brasil

Reinaldo Gomes
reinaldo@computacao.ufcg.edu.br
Universidade Federal de Campina Grande
Campina Grande, Paraíba, Brasil

RESUMO

A portabilidade e facilidade de implantação e manutenção de software e serviços quando utilizando contêineres, fazem com que esse tipo de plataforma seja amplamente aplicada nas comunidades de desenvolvimento e administração de sistemas. Definições padrão de configuração de *container runtimes* e dos orquestradores de contêineres não são ideais à segurança dos ambientes a serem implantados, mas em muitos dos casos são utilizadas por praticidade ou falta de atenção do administrador do sistema na hora da implantação. Essa prática traz grandes riscos de segurança ao ambiente, sendo que em várias ocasiões é a brecha utilizada para violar ativamente os dados e serviços hospedados em nuvem ou conectados de outra forma à rede. Desse modo, foi construída uma proposta de configurações básicas a ser seguida em um ambiente de produção, baseada nas recomendações de configuração de segurança apontadas na documentação do *Kubernetes*, a partir de testes realizados sobre um *cluster* que simula um ambiente de produção. Foi utilizada a ferramenta de código aberto *Kube-bench*, que é bem difundida nas comunidades de segurança e administração de sistemas para análise e verificação dos impactos das configurações realizadas.

1 INTRODUÇÃO

Utilizar contêineres é uma tendência crescente, já que além de tornarem o uso de recursos mais simples e eficiente, trazem a peculiaridade da facilidade de replicação e portabilidade para software e serviços fazendo uso dos *container runtimes* como plataforma de implantação[2].

Apesar dessa facilitação, nem sempre são tomadas todas as precauções possíveis para garantir o baixo nível de vulnerabilidade a falhas de segurança dos ambientes que estão sendo implantados. Um dos fatores decisivos ao nível de segurança desses ambientes é o conjunto de configurações utilizado. Uma grande parte dos eventos de ataques à brechas de segurança é baseada em más configurações dos ambientes. De acordo com o *OWASP TOP 10*[18] de 2021, organizado pela *Open Web Application Security Project (OWASP)*[17], configurações incorretas correspondem ao quinto maior tipo de incidente de segurança no campo de segurança de software, tendo subido do 6º lugar do ranking desde a última versão do estudo (de 2017).

Trazendo para o contexto do uso de *Kubernetes*[15], um grande ator no ambiente de orquestradores de contêineres, algumas dessas configurações envolvem a garantia que o sistema de arquivos tem o devido nível de privilégio de acesso, ou que as máquinas (*nodes* ou nós) podem receber o tipo certo de *pod* (a unidade básica na linha de execução do *Kubernetes*) na hora do agendamento, além de diversas outras configurações específicas que podem tornar o sistema menos vulnerável a ataques ou falhas críticas.

Como uma referência de boas práticas de configurações seguras para *Kubernetes*, o *CIS Kubernetes Benchmark*[11] é uma documentação que propõe um conjunto de configurações adequadas para implantação dos ambientes. Essa documentação é elaborada através de revisão por consenso entre diversas áreas do conhecimento, a citar como principais as áreas de segurança da informação e administração de sistemas.

A ferramenta *Kube-bench*[13] faz uso desse conjunto de regras para realizar a auditoria estática da configuração do ambiente, verificando o nível de adequação aos *benchmarks* definidos, servindo assim como uma ferramenta que o administrador do sistema pode usar para refinar o nível de segurança na implantação de microsserviços que utilizam *Kubernetes* como o orquestrador de contêineres. O *Kube-bench* é uma ferramenta de código aberto e é amplamente utilizado na comunidade de administração de sistemas e segurança de sistemas, sendo capaz de verificar de maneira automatizada grande parte das configurações recomendadas no *CIS Kubernetes Benchmarks*, além de também apontar uma parte de configurações a serem manualmente verificadas ou testadas.

Segundo o relatório publicado na página online da plataforma *StackRox*, *State of container and Kubernetes Security*[24], que realizou um estudo sobre o estado da segurança em contêineres e Kubernetes, de 2020, 90% dos correspondentes reportaram que tinham experienciado incidentes de segurança em seus ambientes *Kubernetes* e de contêineres, sendo que 67% destes teriam ocorrido por erros de configuração gerados por falha humana. Exemplos reportados de brechas de segurança derivadas de configuração incorreta de ambientes *Kubernetes* são recorrentes no mercado de software e serviços, sendo que um dos recentes foi amplamente documentado e reportado[22]. Caso em que, por uma configuração incorreta no sistema de firewall de aplicação web, um atacante teve acesso a dados pessoais de 106 milhões de clientes.

A proposta dessa experimentação é trazer um conjunto de recomendações de configuração na implantação de ambientes baseados em *Kubernetes*, obedecendo um requisito mínimo de aplicação de configurações seguras, com o principal intuito de abordar o problema apresentado. Dessa forma, o impulso sobre a segurança desses ambientes que se tornam cada vez mais comuns pode ser ampliada, gerando um impacto significativo no âmbito de falhas e brechas de segurança. *

2 FUNDAMENTAÇÃO TEÓRICA

Por causa das vantagens oferecidas pelo uso de contêineres e orquestradores os tornando atrativos no mercado, esse tipo de ambiente também se torna foco da atenção de organizações maliciosas e hackers mal intencionados, buscando as vulnerabilidades de segurança oferecidas e sua capacidade de exploração[1]. A utilização de boas práticas de configuração é uma forma de dificultar a atuação desse gênero de crime e exploração, aumentando a segurança dos ambientes e oferecendo formas de os opor[9].

Com o intuito de realizar a utilização de configurações seguras, pode ser empregada uma análise de adequação a *benchmarks* de segurança[8]. Esse tipo de análise é usado para aproximar os sistemas da configuração mais segura enquanto oferecendo a menor quantidade de concessões possível a aplicações e elementos que os compõem.

Além dessas noções sobre segurança, é necessário também entender alguns componentes que foram fundamentais aos testes aplicados.

2.1 Contêineres e *Container Runtimes*

Contêineres são unidades padrão de *software* formando um pacote contendo código e suas dependências, para que tenham funcionalidade confiável e tenham uma portabilidade entre ambientes distintos, agilizando a sua implantação. Os contêineres fazem uso da plataforma do *kernel* do sistema operacional, potencialmente juntamente com outros contêineres, tornando-os uma opção menos dependente de recursos que máquinas virtuais[6].

Container runtimes são *softwares* que rodam os contêineres e gerenciam as imagens de contêineres a serem implantadas[7]. No contexto do experimento, temos como *container runtimes Docker*, *CRI-O* e *containerd*. Vamos nos ater ao *Docker* por ser o que foi utilizado durante todo o processo.

2.1.1 Docker. *Docker* é o *container runtime* mais utilizado no cenário de microsserviços, sendo uma plataforma aberta para desenvolvimento, distribuição e execução de aplicações[5]. A utilização de *Docker*, assim como de outros *container runtimes*, habilita ao administrador do sistema ou usuário das imagens de aplicações a manutenção da infraestrutura, assim como do *software*, de maneira que o usuário consegue realizar ajustes específicos ao seu ambiente.

2.2 Orquestradores de Contêineres

Orquestradores de contêineres são plataformas que realizam agendamento e manutenção de atividade dos contêineres através de *pods* e sobre os nós do *cluster* (referindo-se ao ambiente de funcionamento dos microsserviços).

Dadas as necessidades do ambiente, o orquestrador é capaz de realizar implantação, execução e manutenção de aplicativos no *cluster*, possibilitando ao administrador do sistema os ajustes sobre controle de limite de recursos, agendamento, balanceamento de carga, verificação de integridade, tolerância a falhas e escalonamento automático[3].

2.2.1 Kubernetes. *Kubernetes* é um orquestrador de contêineres, sendo uma plataforma de código aberto, extensiva e portável para gerenciamento de microsserviços[15]. Possibilita a utilização de diversos *container runtimes* distintos, tendo também uma ampla adaptabilidade enquanto tratando de integrações[10]. Como líder de segmento, também está entre os líderes do mercado, sendo que *Open Shift* e *Rancher*, dois outros grandes atuantes, são construídos usando *Kubernetes*[2].

3 METODOLOGIA

A prática foi realizada na forma de testes sobre o ambiente de um *cluster*, implantado utilizando *Kubeadm*[14], visando aproximar da melhor forma possível a um ambiente de produção. Para tanto, foram utilizados três nós, sendo um desses um nó *master* e outros dois nós *worker*, distribuídos em três máquinas virtuais diferentes em uma mesma rede.

Foram utilizados os sistemas operacionais *Ubuntu Desktop* no nó *master* e *Ubuntu Server* nos nós *worker* e foram seguidas as recomendações mínimas de configuração definidas pelo *Kubeadm* na construção da infraestrutura dessas máquinas. Todas as máquinas virtuais foram inicializadas sobre uma só máquina hospedeira na virtualização do ambiente do *cluster*.

A princípio foram testadas as configurações padrão de diferentes *container runtimes*, sendo estes *Docker*, *containerd* e *CRI-O* (rodando o *Kube-bench* uma vez em cada na forma de um *job*, ou trabalho de *cluster*). A partir dessa análise inicial, constatou-se que não haveria alteração relevante aos testes entre as utilizações dos diferentes *runtimes*, já que essas saídas apresentadas pelo *Kube-bench* não apontavam mudanças, estando todos na mesma configuração padrão. A partir daí foram iniciados os testes utilizando o *Docker* como *runtime* base, já que é o *container runtime* mais comumente utilizado, além de ser a escolha padrão, priorizado sobre os demais em um cenário em que vários desses estejam instalados e configurados no ambiente.

Nos testes foi realizada uma iteração sobre as configurações recomendadas na própria documentação do *Kubernetes* na seção *Securing a Cluster*[19]. A partir de cada configuração ajustada, foi executado o *Kube-bench* e verificadas as alterações da saída proveniente dos *logs* do trabalho de *cluster*.

Algumas das configurações apontadas na documentação não foram aplicadas, já que não se adequavam aos testes ou necessitavam de recursos mais robustos ou da interação de outros serviços na implantação, tornando-se inviável à situação dos testes. Esses aspectos serão abordados com mais detalhes na próxima seção.

Em específico, dentre os subtópicos em *Controlling access to the Kubernetes API*, apenas *API Authorization* foi coberto nos testes, mas em uma etapa posterior, onde foi construído o controle de privilégio utilizando *Pod Security Policies*, que também envolveu a utilização de *Role-based access control (RBAC)*[21]. Das demais subseções, o *Kubeadm* já realiza a autenticação por meio de *tokens* e constrói a estrutura de *Transport layer Security (TLS)*[23] durante a inicialização do orquestrador, também fazendo uso dos *tokens* na autenticação do *Kubelet* (relacionado com a seção seguinte na documentação, *Controlling access to the Kubelet*).

A seguir estão listadas as demais seções e subseções da documentação de *Securing a Cluster* e uma descrição da importância

exercida por cada uma na segurança do ambiente, em específico, relacionada ao teste.

3.1 Controlando as capacidades de uma *workload* ou usuário em tempo de execução

Esta seção trata principalmente do funcionamento interno do *cluster*, envolvendo tanto o orquestrador quanto os elementos internos dele (*pods*, *nodes*, contêineres e recursos do ambiente).

A administração correta dos recursos pode ser utilizada como um fator de segurança, estabelecendo limites e cotas para os *pods* a serem implantados no ambiente. Essa especificação, no entanto, requer um nível mais amplo de análise, sendo fora do escopo do *Kube-bench*.

É possível também realizar controles mais finos de módulos do sistema e privilégios de execução dos elementos do *cluster*, utilizando respectivamente filtros do *modprobe* (nativo de ambientes *linux*) e *Pod Security Policies* (componente de configuração do *Kubernetes*) para limitar execução de módulos que são falhos ou não recebem atualizações e dificultar uma brecha de escalação de privilégios.

Em relação a recursos de rede e acesso, podem ser configuradas restrições através da utilização de *Network Policy*[16], limitando os acessos dos *pods*, que caso contrário utilizam sua configuração padrão, aceitando tráfego de qualquer fonte.

O agendamento automático do *Kubernetes* faz com que qualquer nó que não seja, por padrão, marcado como nó *master* (através de um *taint*[20]), estejam aptos a receber qualquer *pod*. Isso pode ser limitado para que haja mais controle sobre o funcionamento do *cluster* utilizando controles de admissão ou os próprios *taints* pareados com o uso de *tolerations* na implantação dos *pods*.

Fazem parte desta seção:

- *Limiting resource usage on a cluster*
- *Controlling what privileges containers run with*
- *Preventing containers from loading unwanted kernel modules*
- *Restricting network access*
- *Restricting cloud metadata API access*
- *Controlling which nodes pods may access*

3.2 Protegendo componentes de *cluster* contra comprometimento

Os tópicos desta seção são focados principalmente na estrutura do sistema hospedeiro e nos componentes dele relacionados ao *cluster*, estabelecendo o controle sobre os arquivos e dados presentes e sua utilização.

O diretório do *"etcd"* é o equivalente ao diretório de armazenamento de todo o *cluster*, sendo um ambiente de dados críticos. Limitar esse acesso protege o *cluster* de uma escalação de privilégio indevida e de acessos inadequados aos dados, que caso contrário poderiam estar vulneráveis. Já o registro de auditoria promove a construção de um histórico das ações da *Kubernetes API*. Esse recurso possibilita uma análise dos eventos do cluster, sendo uma ferramenta útil no aperfeiçoamento da segurança e de verificação e remediação de falhas.

Há então os seguintes pontos:

- *Restrict access to etcd*
- *Enable audit logging*

3.2.1 Tópicos não cobertos. Desses, os quatro primeiros tópicos são subjetivos ao ambiente e às escolhas do administrador do sistema, não sendo úteis ao escopo dos testes realizados. Já o último, requer a utilização de uma aplicação externa ao *cluster* (o ETCD[12]), que necessita de uma estrutura de recursos mais robusta, tornando-se inviável no âmbito do estudo realizado:

- *Restrict access to alpha or beta features*
- *Rotate infrastructure credentials frequently*
- *Review third party integrations before enabling them*
- *Receiving alerts for security updates and reporting vulnerabilities*
- *Encrypt secrets at rest*

4 RESULTADOS

A partir das recomendações seguidas puderam ser verificadas algumas alterações nos resultados do *Kube-bench*, possibilitando a síntese de um conjunto de modificações que se mostram efetivas ao ambiente do *cluster* por meio desse recurso de testes.

A tabela 1 mostra o conjunto de configurações aplicadas que promoveram alterações na saída da ferramenta e os pontos referentes ao *CIS Kubernetes Benchmarks* alterados. Apesar de algumas das configurações realizadas não apresentarem alteração na saída dos *logs*, não quer dizer que sejam menos importantes ao reforço de segurança do ambiente, já que há tópicos destacados no *Kube-bench* como verificações manuais, em conformidade com os itens do *CIS Kubernetes Benchmarks*, não havendo comandos para inspecionar de maneira automática a configuração.

4.1 Aplicação de *Pod Security Policies*

A construção e implantação de uma *Pod Security Policy* alterou a saída relacionada com a ativação do uso desse recurso, como já se esperava:

```
[WARN] 1.2.13 Ensure that the admission control plugin
SecurityContextDeny is set if PodSecurityPolicy is not
used (Manual)
...
[FAIL] 1.2.16 Ensure that the admission control plugin
PodSecurityPolicy is set (Automated)
...
== Summary total ==
70 checks PASS
11 checks FAIL
41 checks WARN
0 checks INFO
```

E após a correção:

```
[PASS] 1.2.13 Ensure that the admission control plugin
SecurityContextDeny is set if PodSecurityPolicy is not
used (Manual)
...
[PASS] 1.2.16 Ensure that the admission control plugin
PodSecurityPolicy is set (Automated)
...
```

```
== Summary total ==
72 checks PASS
10 checks FAIL
40 checks WARN
0 checks INFO
```

A partir dessas configurações, houve o aumento do controle sobre os *pods* do ambiente, sendo que somente *pods* que estavam configurados de acordo com a política estabelecida funcionavam dentro do *cluster*. Dessa maneira, acessos irrestritos ao sistema de arquivos do hospedeiro ou a escalação de privilégios não intencional não puderam mais ser realizados pelos componentes do *cluster*.

4.2 Ativação de um registro de auditoria

A utilização de auditoria para gerir os processos e eventos do *cluster*, trouxe a alteração de vários resultados:

```
[FAIL] 1.2.22 Ensure that the --audit-log-path
argument is set (Automated)
[FAIL] 1.2.23 Ensure that the --audit-log-maxage
argument is set to 30 or as appropriate (Automated)
[FAIL] 1.2.24 Ensure that the --audit-log-maxbackup
argument is set to 10 or as appropriate (Automated)
[FAIL] 1.2.25 Ensure that the --audit-log-maxsize
argument is set to 100 or as appropriate (Automated)
...
== Summary total ==
72 checks PASS
10 checks FAIL
40 checks WARN
0 checks INFO
```

Após a aplicação:

```
[PASS] 1.2.22 Ensure that the --audit-log-path
argument is set (Automated)
[PASS] 1.2.23 Ensure that the --audit-log-maxage
argument is set to 30 or as appropriate (Automated)
[PASS] 1.2.24 Ensure that the --audit-log-maxbackup
argument is set to 10 or as appropriate (Automated)
[PASS] 1.2.25 Ensure that the --audit-log-maxsize
argument is set to 100 or as appropriate (Automated)
...
== Summary total ==
76 checks PASS
6 checks FAIL
40 checks WARN
0 checks INFO
```

A partir dessas configurações, foi criado um arquivo de *logs* para que pudesse ser feita a verificação dos eventos do *cluster*, facilitando a manutenção, tanto do ambiente, quanto da segurança[4].

4.3 Aplicação de filtro de módulos e proteger os padrões do *kernel*

A aplicação de filtros sobre os módulos que podem ser executados pelo *Kubernetes* não resultou em mudanças na saída do *Kube-bench*, mas aplicar a proteção aos padrões do *kernel* fez com que um ponto ficasse em conformidade:

```
[FAIL] 4.2.6 Ensure that the --protect-kernel-defaults
argument is set to true (Automated)
...
== Summary total ==
76 checks PASS
6 checks FAIL
40 checks WARN
0 checks INFO
```

Após a aplicação da configuração:

```
[PASS] 4.2.6 Ensure that the --protect-kernel-defaults
argument is set to true (Automated)
...
== Summary total ==
77 checks PASS
5 checks FAIL
40 checks WARN
0 checks INFO
```

Vale salientar que essa mudança também tornou o sistema instável após a inicialização, mas o tempo e recurso dos testes não foram suficientes para realizar averiguação do motivo.

A aplicação de filtro sobre módulos do *Kernel* garantiu que os controles de segurança do sistema hospedeiro fossem mantidos. A partir disso, o ambiente da máquina virtual pôde ser utilizado como mais uma camada de segurança para o *cluster*.

4.4 Demais configurações e abordando os outros *fails*

Além dessas configurações, nas quais eram esperadas as alterações ocorridas, apenas a de restrição de acesso ao *"etcd"* não resultou em mudanças na saída. A verificação através do comando foi realizada e o retorno estava nos conformes.

```
$ stat -c %U:%G /var/lib/etcd
etcd:etcd
```

O ocorrido tem precedentes na comunidade do *Kube-bench*. Pelo processo do *"etcd"* estar rodando em um contêiner, o ambiente da máquina hospedeira não possui um usuário relacionado, sendo necessária a criação de um e atribuição do diretório a este. Mesmo com esse ajuste, não houve alteração no resultado, tendo sido mantido o status de *"[FAIL]"* no teste.

Após o processo de adequação às recomendações da documentação em *Securing a Cluster*[19], foram endereçados os demais problemas reportados pelo *Kube-bench*. Em específico, a configuração que desativa o perfilamento dos componentes do *cluster*:

```
[FAIL] 1.2.21 Ensure that the --profiling argument
is set to false (Automated)
...
1.2.21 Edit the API server pod specification file
/etc/kubernetes/manifests/kube-apiserver.yaml
on the master node and set the below parameter.
--profiling=false

[FAIL] 1.3.2 Ensure that the --profiling argument
is set to false (Automated)
...
1.3.2 Edit the Controller Manager pod specification file
/etc/kubernetes/manifests/kube-controller-manager.yaml
on the master node and set the below parameter.
--profiling=false

[FAIL] 1.4.1 Ensure that the --profiling argument
is set to false (Automated)
...
1.4.1 Edit the Scheduler pod specification file
/etc/kubernetes/manifests/kube-scheduler.yaml file
on the master node and set the below parameter.
--profiling=false

== Summary total ==
77 checks PASS
5 checks FAIL
40 checks WARN
0 checks INFO
```

Após a correção:

```
[PASS] 1.2.21 Ensure that the --profiling argument
is set to false (Automated)
...
[PASS] 1.3.2 Ensure that the --profiling argument
is set to false (Automated)
...
[PASS] 1.4.1 Ensure that the --profiling argument
is set to false (Automated)
...
== Summary total ==
80 checks PASS
2 checks FAIL
40 checks WARN
0 checks INFO
```

Esse ajuste é mais relacionado ao nível preventivo, sendo que a partir do momento em que o perfilamento é habilitado, podem ser avaliados gargalos e problemas de desempenho do sistema do *cluster*. Essa configuração pode dar pistas de falhas exploráveis a atacantes que possam realizar o perfilamento do sistema, enquanto está ativa[11]. Por isso, há a sugestão de desabilitá-la.

A última falha apresentada (à parte da falha relacionada ao *"etcd"*) é relacionada ao uso de *TLS*[23], que não foi coberto no conjunto de testes realizado.

5 CONCLUSÃO

A aplicação das configurações recomendadas trouxe os resultados esperados, aumentando o controle sobre o *cluster* e a habilidade de resposta sobre erros e falhas de segurança do administrador do sistema.

Em um ambiente de produção a utilização de todas as configurações pode se tornar inviável, mas a aplicação de ao menos uma parte dessas configurações é de grande importância à manutenção da segurança do ambiente, sendo que as aqui destacadas irão reforçar consideravelmente a segurança do ambiente.

Sempre que possível, trabalhar com a aplicação de uma *Pod Security Policy*, pode dar controle sobre os níveis de acesso que cada *pod* vai alcançar, especificamente funcionando como uma plataforma de controle de segurança dos *pods* do *cluster*, já que um dos principais meios de exploração de brechas do segmento é utilizar a escalação de privilégios. Essa função requer a aplicação de diversos níveis de configurações, se tornando complexa em casos que *pods* necessitam de muita especificidade de acesso. Em um caso mínimo, aplicar níveis distintos para tipos distintos de aplicações é o suficiente, dando mais controle de privilégio somente para os *pods* que realmente fazem uso disso e limitando o máximo possível os outros. Em uma pequena quantidade de objetos de política de segurança é possível realizar um aumento razoável de segurança do ambiente de produção.

A utilização de um registro de auditoria é sempre útil, apesar de se tornar uma medida posterior a falhas e situações de risco, a capacidade de visualizar os eventos de um *cluster* dessa forma aumenta as chances de elevar a segurança do ambiente. No pior dos casos, a utilização do módulo de auditoria de *Kubernetes* vai aumentar o uso de memória secundária do ambiente e talvez requer um pouco de recursos a mais.

Construir um documento de política de auditoria (*audit poilicy*) bem elaborado pode manter esses custos baixos e ainda assim oferecer um bom nível de informações sobre os eventos do ambiente.

A utilização da opção de proteção dos padrões de *kernel* preestabelecidos faz com que o *cluster* proteja a configuração construída sobre o sistema hospedeiro, trabalhando assim com o que foi definido pelo administrador do sistema. Subentende-se que o ambiente foi ajustado de maneira segura e que deve ser mantido daquela forma, assim é uma configuração de uso especificamente importante. A utilização dessa configuração, no entanto, requer uma atenção maior sobre as configurações também do *cluster*, sendo que ambas devem estar sintonizadas para que não haja inconsistência e instabilidade.

Cabe também ao administrador do sistema se manter informado sobre possíveis ameaças e necessidades específicas do seu sistema. Isso promove uma atenção aos detalhes relacionados às demais configurações, principalmente as subjetivas ao ambiente. Ter uma métrica de utilização dos recursos e manter o ambiente sempre atualizado vai diminuir consideravelmente a chance de uma falha de segurança no sistema. Além disso fazer análises e testes sobre

Tabela 1: Resultados da aplicação das configurações sobre o retorno de execução do *Kube-bench*

Seção	Itens alterados(*CIS Kubernetes Benchmarks*)
Controlling what privileges containers run with	1.2.13, 1.2.16
Preventing containers from loading unwanted kernel modules	4.2.16
Enable audit logging	1.2.22, 1.2.23, 1.2.24, 1.2.25

o sistema com ferramentas como o *Kube-bench*, vai permitir que problemas específicos do ambiente sejam abordados.

Vale também mencionar que pelos testes terem sido realizados em um ambiente que progrediu da configuração padrão até uma configuração inicial aceitável, não quer dizer que está completa. Como mencionado, cada situação deverá ser abordada de acordo com as necessidades do ambiente implantado, visando principalmente atender a expectativa do administrador do sistema e dos usuários do serviço ou aplicação. Portanto, utilizar esse texto e o que aqui foi exposto como uma linha de base para uma boa atividade de configurações seguras sobre ambientes que utilizam *Kubernetes* como orquestrador de contêineres, é razoável.

5.1 Sugestão de pesquisa

Por causa de recursos e tempo, as aplicações dos testes foram limitadas e houve necessidade de um controle maior sobre o ambiente para que os testes pudessem ser completados.

Sugere-se que sejam feitos estudos sobre ambientes em atividade utilizando o *Kube-bench* ou outras ferramentas com objetivos semelhantes, de forma que sejam adquiridos dados estatísticos sobre a segurança aplicada a ambientes reais de produção e assim possa ser construída uma visão amplificada do tema.

REFERÊNCIAS

[1] Linetskyi Artem, Babenko Tetiana, Myrutenko Larysa, and Vialkova Vira. Eliminating privilage escalation to root in containers running on kubernetes. *Scientific and practical cyber security journal*, 2020.

[2] E Carter. Sysdig 2019 container usage report: New kubernetes and security insights. *Sysdig Inc*, 29, 2019.

[3] Emiliano Casalicchio and Stefano Iannucci. The state-of-the-art in container technologies: Application, orchestration and security. *Concurrency and Computation: Practice and Experience*, 32(17):e5668, 2020.

[4] Datadog. How to monitor kubernetes audit logs, acessado em outubro 2021. https://www.datadoghq.com/blog/monitor-kubernetes-audit-logs/.

[5] Docker. Docker overview, acessado em outubro 2021. https://docs.docker.com/get-started/overview/.

[6] Docker. Use containers to build, share and run your applications, acessado em outubro 2021. https://www.docker.com/resources/what-container.

[7] Lennart Espe, Anshul Jindal, Vladimir Podolskiy, and Michael Gerndt. Performance evaluation of container runtimes. In *CLOSER*, pages 273–281, 2020.

[8] Jéssica Pereira Ferreira. *Hardening de Sistemas com CIS Benchmark*. PhD thesis, 2021.

[9] Sergio Loureiro. Security misconfigurations and how to prevent them. *Network Security*, 2021(5):13–16, 2021.

[10] Lubos Mercl and Jakub Pavlik. The comparison of container orchestrators. In *Third International Congress on Information and Communication Technology*, pages 677–685. Springer, 2019.

[11] [n.d.]. Cis kubernetes benchmarks, acessado em outubro 2021. https://www.cisecurity.org/benchmark/kubernetes/.

[12] [n.d.]. Etcd, a distributed, reliable key-value store for the most critical data of a distributed system, acessado em outubro 2021. https://etcd.io/.

[13] [n.d.]. Kube-bench, acessado em outubro 2021. https://github.com/aquasecurity/kube-bench.

[14] [n.d.]. Kubeadm, acessado em outubro 2021. https://kubernetes.io/docs/reference/setup-tools/kubeadm/.

[15] [n.d.]. Kubernetes, acessado em outubro 2021. https://kubernetes.io/.

[16] [n.d.]. Network policies, acessado em outubro 2021. https://kubernetes.io/docs/concepts/services-networking/network-policies/.

[17] [n.d.]. Owasp, acessado em outubro 2021. https://owasp.org/.

[18] [n.d.]. Owasp top 10 : 2021, acessado em outubro 2021. https://owasp.org/Top10/.

[19] [n.d.]. Securing a cluster, acessado em outubro 2021. https://kubernetes.io/docs/tasks/administer-cluster/securing-a-cluster/.

[20] [n.d.]. Taints and tolerations, acessado em outubro 2021. https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/.

[21] [n.d.]. Using rbac authorization, acessado em outubro 2021. https://kubernetes.io/docs/reference/access-authn-authz/rbac/.

[22] Nelson Novaes Neto, Stuart Madnick, Moraes G de Paula, Natasha Malara Borges, et al. A case study of the capital one data breach. *Stuart E. and Moraes G. de Paula, Anchises and Malara Borges, Natasha, A Case Study of the Capital One Data Breach (January 1, 2020)*, 2020.

[23] Eric Rescorla and Tim Dierks. The transport layer security (tls) protocol version 1.3. 2018.

[24] StackRox. The state of kubernetes security 2021, acessado em outubro 2021. https://security.stackrox.com/state-of-kubernetes-security-2021.html?Source=Website&LSource=Website.