Linux Foundation CKS参考書勉強: Certified Kubernetes Security Specialist (CKS) - It-Passports最高の供給資格認証攻略

さらに、It-Passports CKSダンプの一部が現在無料で提供されています：https://drive.google.com/open?id=1ureVAYuIRpdE7lFSitd0aBnQU_zfNjv0

弊社It-Passportsの資料を使用すると、最短でCertified Kubernetes Security Specialist (CKS)の最高の質問トレントを習得し、他のことを完了するための時間とエネルギーを節約できます。 最も重要なのは、Linux FoundationのCertified Kubernetes Security Specialist (CKS)学習資料を安全にCKSダウンロード、CKSインストール、使用できることです。 製品にウイルスがないことを保証できます。 それだけでなく、最高のサービスと最高のLinux FoundationのCertified Kubernetes Security Specialist (CKS)試験トレントを提供し、製品の品質が良好であることを保証できます。 そのため、購入後はお気軽にご利用ください。お金を無駄にさせません。

CKS 試験は、Kubernetes セキュリティに関連する実世界の問題を解決する候補者の能力をテストする実践的でパフォーマンスベースの試験です。候補者は、所定の時間内に Kubernetes クラスターとアプリケーションのセキュリティに関連するタスクを実行する必要があります。試験はオンラインで実施され、候補者は世界中から受験できます。

>> CKS参考書勉強 <<

CKS試験の準備方法｜権威のあるCKS参考書勉強試験｜一番優秀なCertified Kubernetes Security Specialist (CKS)資格認証攻略

It-PassportsのLinux FoundationのCKS試験トレーニング資料を手に入れるなら、君が他の人の一半の努力で、同じLinux FoundationのCKS認定試験を簡単に合格できます。あなたはIt-PassportsのLinux FoundationのCKS問題集を購入した後、私たちは一年間で無料更新サービスを提供することができます。もしうちのLinux FoundationのCKS問題集は問題があれば、或いは試験に不合格になる場合は、全額返金することを保証いたします。

Linux Foundation Certified Kubernetes Security Specialist (CKS) 認定 CKS 試験問題 (Q24-Q29):

質問 # 24
Secrets stored in the etcd is not secure at rest, you can use the etcdctl command utility to find the secret value for e.g- ETCDCTL_API=3 etcdctl get /registry/secrets/default/cks-secret --cacert="ca.crt" --cert="server.crt" --key="server.key" Output
Using the Encryption Configuration, Create the manifest, which secures the resource secrets using the provider AES-CBC and identity, to encrypt the secret-data at rest and ensure all secrets are encrypted with the new configuration.

正解：

解説：
ETCD secret encryption can be verified with the help of etcdctl command line utility.
ETCD secrets are stored at the path /registry/secrets/$namespace/$secret on the master node.

The below command can be used to verify if the particular ETCD secret is encrypted or not.
ETCDCTL_API=3 etcdctl get /registry/secrets/default/secret1 [...] | hexdump -C

質問 # 25
You are building a microservice architecture on Kubernetes- You are using Docker images from a public registry for your applications. One of the microservices is responsible for managing sensitive user data. To minimize the base image footprint and enhance security, you need to create a custom base image that iS as minimal as possible while still containing the required dependencies for your service.
What are the steps you would take to create a custom base image for this microservice? How would you ensure the custom base image is secure, and how would you incorporate it into your deployment process? Provide a step-by-step guide with code examples.

正解：

解説：
Solution (Step by Step) :
1. Choose a Minimal Base Image:
- Select a base image like Alpine Linux, which is known for its small size and security features.
- Use a multi-stage build to minimize the size of the final image.
- Example:
docket-file
FROM alpine:3.16 as builder
Install required dependencies
RUN apk update && apk add --no-cache python3 python3-dev build-base
2. Security Best Practices: - Use a non-root user inside the container. - Enable security options in your Dockerfile like '-no-cache' to minimize potential vulnerabilities. - Harden the base image: - Remove unnecessary packages and services. - Disable unnecessary pons and protocols. - Set appropriate permissions for files and directories. - Example: dockeflle FROM alpine:3.16 as builder USER nonrootuser RUN apk update && apk add 0--no-cache python3 python3-dev build-base # ... rest of the Dockerfile 3. Deployment Process: - Build the custom base image. - Push the base image to a private registry. - Update the deployment YAML file to use the new base image. - Example:
4. Testing and Monitoring: - Regularly scan the base image for vulnerabilities. - Monitor the container for suspicious activity - Employ security tools like Falco and Clair-

質問 # 26
You need to implement a secure network policy that allows communication only between specific pods within a namespace. For example, you want to allow communication between pods that have the label 'app=frontend' and pods that have the label 'app=backend', but block all other communication within the namespace.

正解：

解説：
Solution (Step by Step) :
1. Create a NetworkPolicy:
- Define a NetworkP01icy that allows communication between 'frontend' and 'backend' pods, but blocks other communication within the namespace.
2. Create a Frontend Pod: - Create a Pod with the label 'app=frontend'.
3. Create a Backend Pod: - Create a Pod With the label 'app=backend'.
4. Apply the YAML files: - Apply the created YAML files using ' kubectl apply -f 5. Verify the Network Policy: - Try to connect from the 'frontend-pod' to the 'backend-pod' (e.g., using ' kubectl exec -it frontend-pod bash' and 'curl backend-pod:80')- It should succeed. - Try to connect from the 'frontend-pod' to another pod in the namespace that doesn't have the Sapp-backend' label. This connection should be blocked.

質問 # 27
You are tasked with securing a Kubernetes cluster that runs sensitive workloads. You need to implement a mechanism to enforce least privilege access for all pods in the cluster.

正解：

解説：
Solution (Step by Step) :
1. Create a Service Account with Limited Permissions:
- Create a new ServiceAccount with minimal permissions:
2. Create a Role with Limited Permissions: - Create a Role that only grants the necessary permissions for the pods:
3. Bind the Role to the Service Account: - Bind the Role to the ServiceAccount:
4. Configure PodS to use the Service Account - Update your Deployment YAML to use the ServiceAccount:

質問 # 28
You are responsible for securing the Kubernetes clusters supply chain. Your organization utilizes a private Docker registry to host container images. Currently, images are built and pushed to this registry without any validation or signing. How can you implement a policy to ensure that only signed and verified images are deployed to the cluster?

正解：

解説：
Solution (Step by Step) :
1. Set Up a Signing Authority:
- Choose a trusted entity (e.g., a dedicated server or a dedicated user account) to act as the signing authority.
- Generate a private and public key pair using tools like 'openssr or 'gpg'
- Store the private key securely and ensure only authorized individuals have access.
2. Configure Image Signing:
- Create a script or integrate signing into your image build process.
- when building an image, use the private key from the signing authority to sign the image.
- The signing process embeds a digital signature within the image manifest.
3. Integrate Image Verification
- Configure the Kubernetes cluster to enforce image signature verification.
- Utilize tools like 'admission webhookS to inspect incoming images.
- The webh00k will check if the image has a valid signature from the trusted authority.
- If the signature is invalid or missing, the deployment will be blocked.
4. Example Implementation (using 'cosign'):
-
5. Integrate with CI/CD pipelines: - Integrate image signing and verification into your automated CI/CD pipelines. - This ensures consistency and prevents accidental deployment of unsigned images.

質問 # 29
.....

だからってバカにしやがって！

Linux Foundation CKS参考書勉強: Certified Kubernetes Security Specialist (CKS) - It-Passports 最も信頼できるウェブサイト

一部の人々は、一部のWebサイトCertified Kubernetes Security Specialist (CKS)で製品を購入した後、匿名のSMS広告やCKSテレマーケティングに悩まされることがよくあります、お客様に最高のサービスを提供するというコンセプトに沿って、当社は専任のサービスチームと成熟した思慮深いサービスシステムを構築しました。

当社は、教材を担当しています、信じられないなら、我々のサイトで無料なサンプルを利用してみることができます、It-PassportsはLinux FoundationのCKS「Certified Kubernetes Security Specialist (CKS)」の認証試験の合格率を高めるのウエブサイトで、It-Passports中のIT業界の専門家が研究を通じてLinux FoundationのCKSの認証試験について問題集を研究し続けています。

ちなみに、It-Passports CKSの一部をクラウドストレージからダウンロードできます：https://drive.google.com/open?id=1ureVAYuIRpdE7lFSitd0aBnQU_zfNjv0