CONTAINER SECURITY WORKSHOP

ABOUT US

- Rory McCune
  - Security Advocate at Datadog
  - Ex-Pentester/Infosec person
- Iain Smart
  - Containerisation Practice Lead at NCC Group
  - SecDevOps Hacker Person

HIGH LEVEL COURSE OBJECTIVES

- Getting familiar with Containers
- Securing and breaking into containerized workflows
- Introduction to Kubernetes and container clustering

COURSE LOGISTICS

- Ground Rules
- Materials
  - Slides
  - Handouts
- Questions? **Just Ask**

AGENDA

- Container Basics
- Docker Security
- Kubernetes Fundamentals
- Kubernetes Security
- Kubernetes Networking
- Kubernetes Distributions
- Container Security Problems/Challenges

ACCESSING THE LABS

- Course content available at https://workshop.steelcon.container.farm
  - We aren't frontend people...
  - Yes, shared SSH key
- SSH to `ubuntu@studentx.steelcon.container.farm` where `x` is your student number
- Alternatively, pretty web editor:
  - https://studentx.steelcon.container.farm
  - Password is `containersarecool123!`

CONTAINER BASICS

WHAT IS A CONTAINER?

Jorge Silva
@thejsj

Follow

CONTAINERS ARE NOT A REAL THING!!! @jessfraz talking containers #GoogleNext17

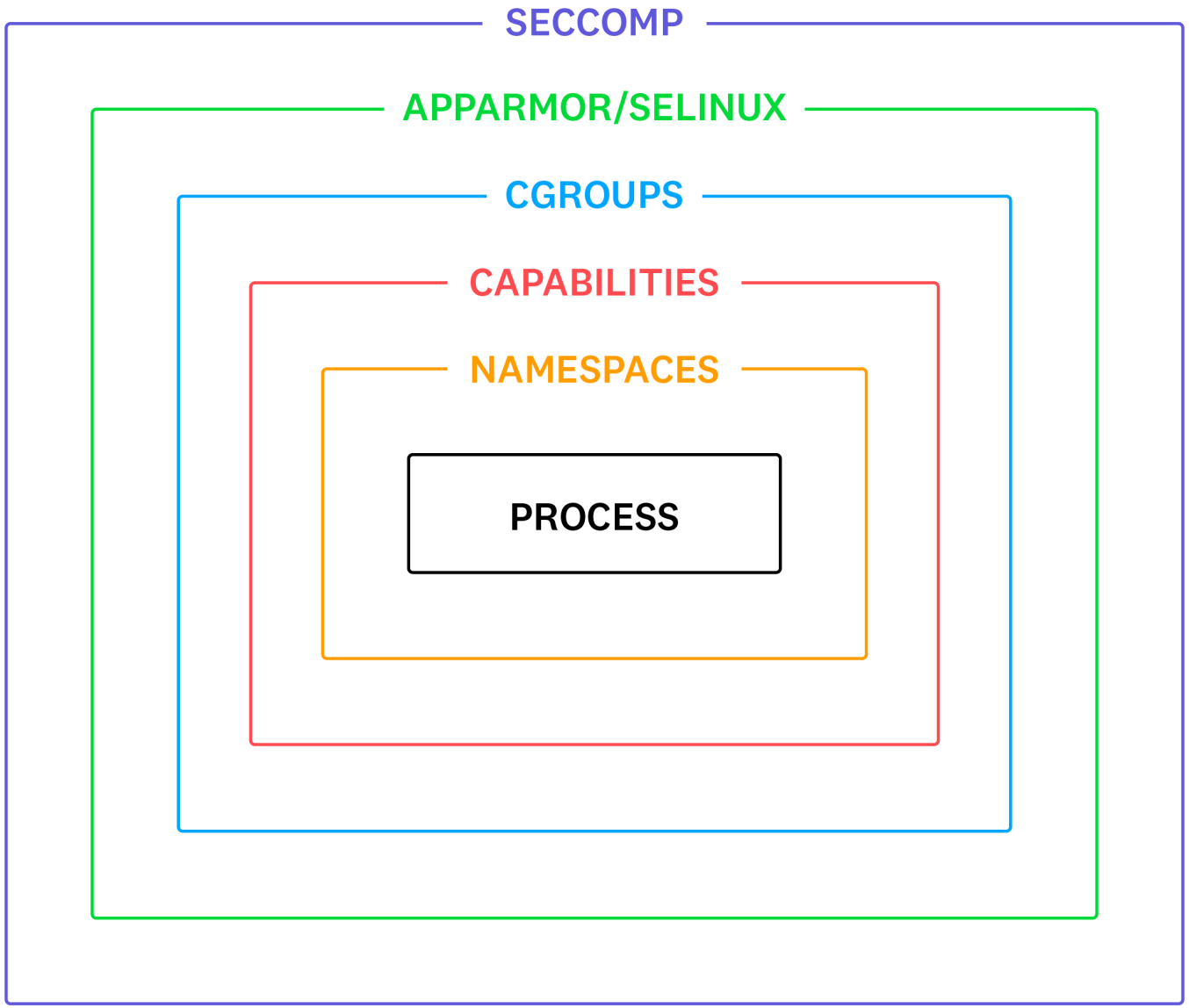
8:16 PM - 10 Mar 2017

23 76

SO WHAT IS A CONTAINER?

- It depends!
  - Linux containers are usually just processes
  - Some Linux containers use VM isolation
  - Windows containers are either Job Objects or Hyper-V VMs

CONTAINER ISOLATION



RUNNING CONTAINERS - LINUX

- Docker daemon + CLI
  - Install from package manager
  - Install from Docker
- Podman
- LXC/LXD

RUNNING LINUX CONTAINERS - WINDOWS/MAC

- Docker Desktop
- Rancher Desktop
- Podman Desktop
- Docker CLI + VM

EXERCISE 1 - RUNNING CONTAINERS IN DOCKER

```
docker run hello-world
```

- Please shout if this doesn't work. If it doesn't, none of our labs will

EXERCISE 2 - SINGLE COMMAND CONTAINERS

```
docker run raesene/ubuntu-nettools ip addr
```

EXERCISE 3 - INTERACTIVE CONTAINERS

```
docker run -it ubuntu:22.04 /bin/bash
```

EXERCISE 4 - INTERACTIVE CONTAINERS (2)

```
CTRL-PQ
```

```
docker ps
```

```
docker attach <id>
```

EXERCISE 5 - BACKGROUND CONTAINERS

```
docker run -d nginx
```

```
docker ps
```

```
docker stop <nginx_id_here>
```

EXERCISE 6 - DOCKER CONTAINERS ARE JUST PROCESSES

```
ps -fC nginx
```

```
docker run -d --name webserver nginx
```

```
ps -fC nginx
```

```
docker exec webserver touch /my-file
```

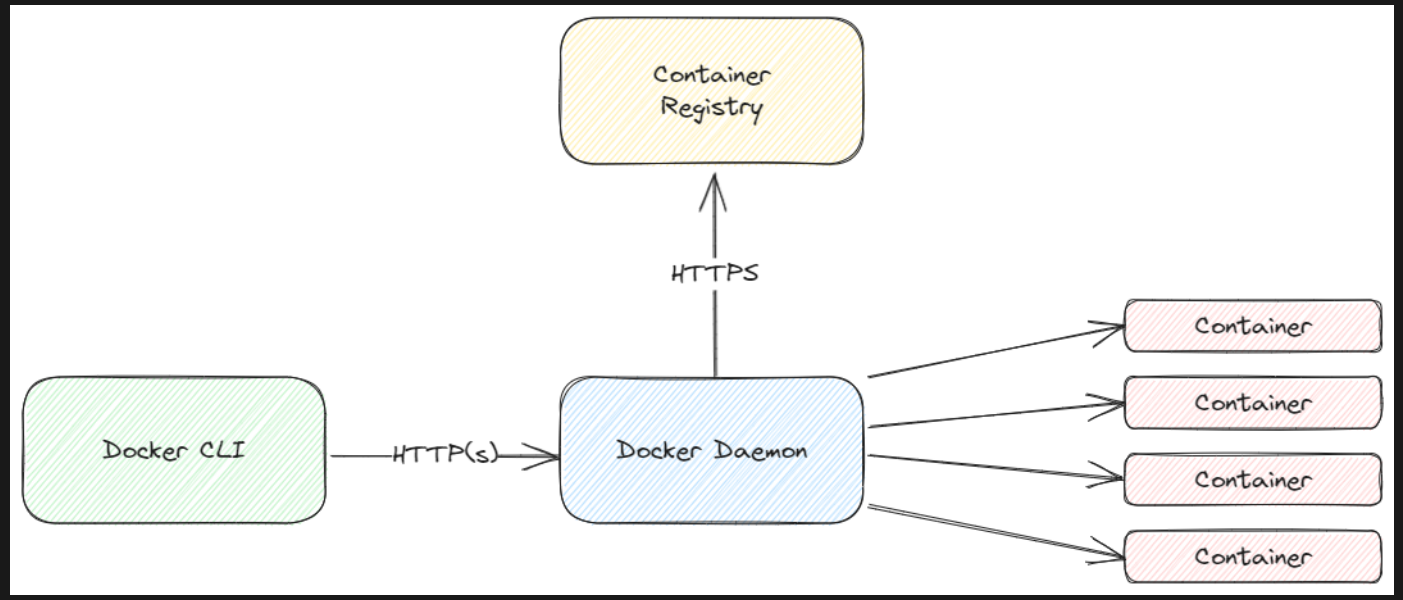
```
sudo ls /proc/[pid]/root
```

MODULE CONCLUSION

- Containers are just processes
- There's a number of ways we can run containers using Docker or other tools

DOCKER SECURITY

DOCKER ARCHITECTURE



DOCKER ATTACK SURFACE

- Docker daemon
  - Listen on a socket (/var/run/docker.sock) **This is the default**
  - Listen on a TCP port (2375/TCP) unauthenticated.
  - Listen on a TCP port (2376/TCP) authenticated.

DOCKER DAEMON AUTHENTICATION

- Docker daemon can be configured to listen on a TCP port with TLS authentication.
- Authentication is based on client certificates.
  - client credentials stored in `~/.docker` by default

EXERCISE 1 - VIEWING DOCKER DAEMON TRAFFIC

```
sudo socat -v UNIX-LISTEN:/tmp/tempdock.sock,fork UNIX-CONNECT:/var/run/docker.sock
```

- In another terminal:

```
sudo docker -H unix:///tmp/tempdock.sock images
```

LOCAL ATTACK SURFACE

- Docker Socket
  - `/var/run/docker.sock`
  - Default permissions are `660` (root:docker)
- Containerd Socket
  - `/run/containerd/containerd.sock`
  - Default permissions are `600` (root:root)

DOCKER SECURITY MODEL

- Relatively simple. If you have Docker access, you have root.
- All of the layers of isolation that containers provide can be removed by anyone with docker access.

PRIVESC WITH DOCKER SOCKET ACCESS

- Once you have access to the Docker socket on a host getting `root` should be trivial
- There's a number of different ways of doing it but the easiest is "The Most Pointless Docker Command Ever"

EXERCISE 2 - PRIVESC WITH DOCKER SOCKET ACCESS

```
docker run -ti --privileged --net=host --pid=host --ipc=host --volume /:/host busybox chroot /host
```

CONTAINER BREAKOUT

- From inside a container, there's a number of ways you might be able to break out
  - mounted docker socket (use the pointless docker command)
  - Other "sensitive" mounts (e.g. `/etc/shadow`)
  - `privileged` containers
  - kernel exploits

EXERCISE 3 - CONTAINER BREAKOUT FROM A PRIVILEGED CONTAINER

```
docker run -ti --privileged ubuntu:22.04 /bin/bash
```

```
mount
```

```
mkdir /host
```

```
mount /dev/nvme0n1p1 /host
```

- Edit files as needed (e.g. /host/etc/shadow)

CONTAINER IMAGE SECURITY

- Containers run as root by default!
  - An important first step when using them is trying to run them as non-root.
- Container Images are essentially mini-linux distributions (in most cases)
  - OS Libs and language Libs need patching just like any other OS.

CONTAINER SECURITY SCANNERS

- Wide range of options available
  - Trivy
  - Grype
  - ...
- Can scan images for vulnerabilities
- Some can also scan for mis-configurations

EXERCISE 4 - SCANNING AN IMAGE WITH TRIVY

```
trivy image ubuntu:22.04
```

```
trivy image --ignore-unfixed ubuntu:22.04
```

```
trivy image --image-config-scanners config ubuntu:22.04
```

MODULE CONCLUSION

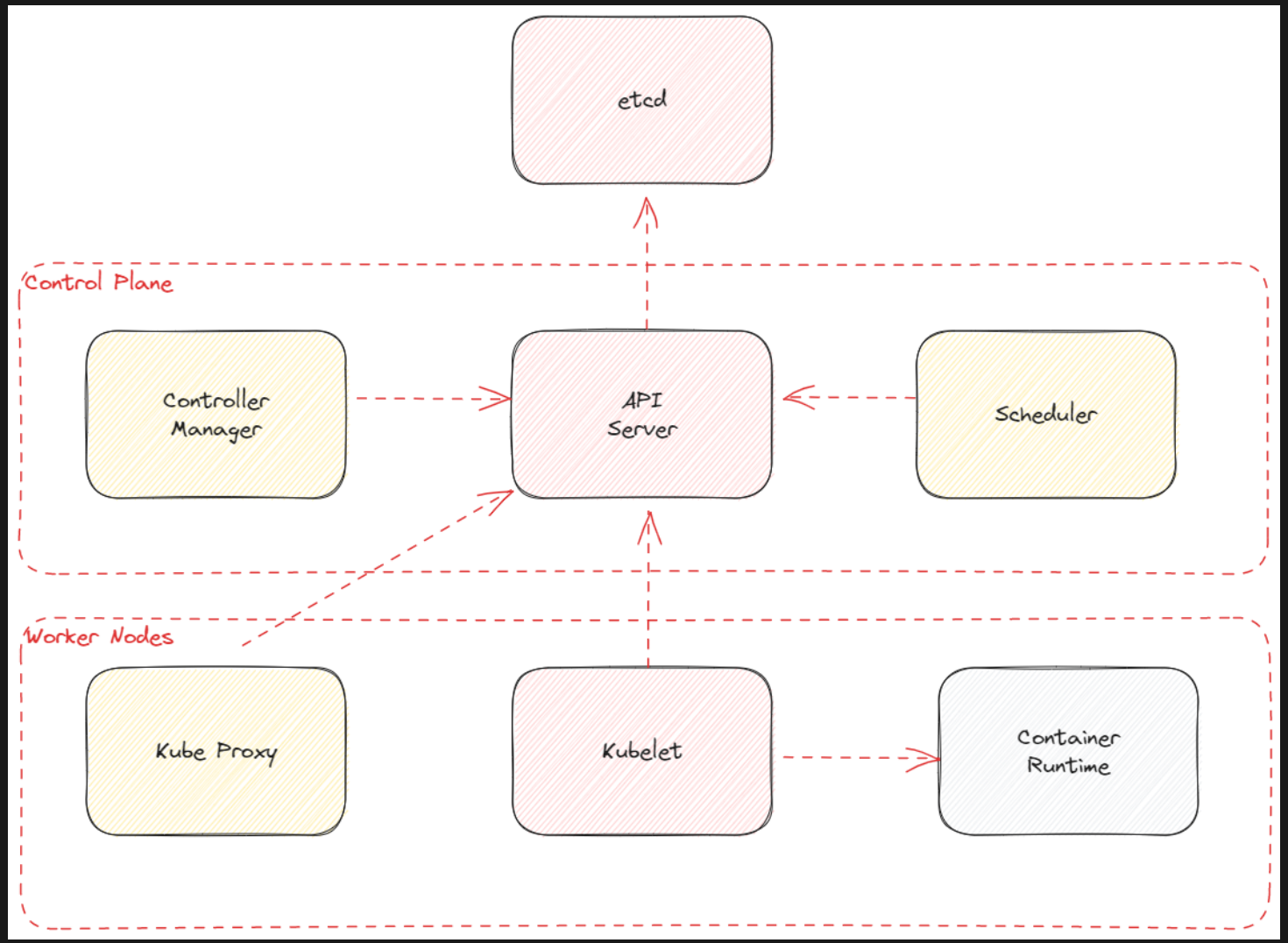
- Docker has a relatively simple security model.
- Users who have Docker rights will be able to gain root
- "The most pointless docker container" is actually the most useful one.

KUBERNETES FUNDAMENTALS

WHAT IS KUBERNETES?

- Container orchestration platform
- Started by Google now managed by the CNCF
- Not the only way to orchestrate containers, but the main one.

KUBERNETES ARCHITECTURE



KUBERNETES RESOURCES

- Base Kubernetes has 50+ resources types.
- Fortunately you don't need to know about, or use, most of them

```
kubectl api-resources
```

KUBERNETES COMPONENTS - API SERVER

- Core of a Kubernetes cluster
- Manages communication with all other components
- Presents an HTTP API for interaction
  - 443/TCP usually
  - Sometimes 6443/TCP or 8443/TCP

KUBERNETES COMPONENTS - SCHEDULER

- Lives in the control plane
- Handles deployment of pods to nodes
- All communications via the API server
- Typically listens on 10259/TCP

KUBERNETES COMPONENTS - CONTROLLER MANAGER

- Lives on the control plane
- Actually a collection of different controllers
- Works via the API Server
- Typically listens on 10257/TCP

KUBERNETES COMPONENTS - ETCD

- Key/value store
- Can be either a single instance or a cluster of it’s own
- Responsible for storing cluster state
- 2379/TCP – client communication
- 2380/TCP – inter-cluster communications.
- Technically can be used by not-Kubernetes projects, but rarely is.

KUBERNETES COMPONENTS - KUBELET

- Lives on most/all nodes
- listens on 10250/TCP
- Manages the Container runtime
  - Containerd, CRI-O, Docker, or others...

KUBERNETES COMPONENTS - KUBE-PROXY

- Network Proxy*
- Runs on each node
- Handles the mapping of services to pods
- Forwards traffic to containers in the cluster
- "Healthz" port typically 10256/TCP

WHAT KUBERNETES DOESN'T DO OUT OF THE BOX

- In some areas the Kubernetes designers took the position that they would delegate an area to external software
- For each of these an interface was designed so that a consistent API would be available.
- Main ones are:
  - CRI - Container Runtime Interface.
  - CNI - Container Network Interface.
  - CSI - Container Storage Interface.

KUBECTL

- This is the main tool used to manage and interact with clusters
- At least somewhat modelled after the Docker client.
- has a wide range of commands for container lifecycle management
- Help system is pretty good. `--help` is your friend!

ACCESSING CLUSTERS - KUBECONFIG

- Kubeconfig is the main way to access clusters
- A file that, by default, lives in `~/.kube/config`
- Contains definitions of one or more clusters and one or more users
- Sometimes has embedded credentials, sometimes references external credentials

INSPECTING A KUBECONFIG

- 3 sections plus metadata
  - Cluster definitions handle the network bits
  - User data is your identity and authentication
  - Contexts pair users to clusters
- You should all have a rancher kubeconfig on your machines

VIEW YOUR KUBECONFIG

```
cat ~/.kube/config
```

INTRODUCTION TO RANCHER

- Rancher is a managed Kubernetes distribution
- No affiliation, it's just shiny
- Accessible at https://rancher.steelcon.container.farm
- studentx::Changeme123!

EXERCISE 1 - RUNNING COMMANDS IN A CLUSTER

```
kubectl get pods
```

EXERCISE 2 - RUNNING A POD

```
kubectl run --image nginx {yourinitials}-nginx
```

CONCLUSION

- Kubernetes is a relatively complex system compared to Docker
- It's important to understand the components and how they interact
- It's important to understand how to access the cluster

EXERCISE 1 - SETTING UP A KIND CLUSTER

```
kind create cluster --config=kind_configs/kind-config.yaml
```

```
kubectl get po -A
```

EXERCISE 2 - LOOKING AT THE KUBERNETES PROCESSES

```
docker exec -it kind-control-plane /bin/bash
```

```
ps -ef
```

```
ss -ltnp
```

KUBERNETES SECURITY

KUBERNETES ATTACKS

- Three Threat Models
  - External Attacker
  - Compromised Container
  - Malicious User

EXTERNAL ATTACKER

- Attack Surface - Cloud Hosted
  - Likely just the API Server
- Attack Surface - On-Premises - A range of potential ports
  - API Server
  - Kubelet
  - etcd

FINDING KUBERNETES CLUSTERS ONLINE

- Shodan, Censys, BinaryEdge, etc.
- Censys coverage is likely the best at the moment

EXERCISE 1 - FINDING KUBERNETES CLUSTERS ONLINE

- Go to https://search.censys.io

```
services.kubernetes.version_info.git_version="*"
```

```
services.kubernetes.pod_names="*"
```

API SERVER ACCESS

- Usually authenticated, but not always
- On older clusters the `insecure-port` can be enabled
- On newer clusters it's possible to bind rights to the `system:anonymous` user to allow unauthenticated access.

EXERCISE 2 - SETTING UP A KIND CLUSTER

```
kind create cluster --name=insecurecluster --image=kindest/node:v1.19.16 --config ~/kind_configs/insecurecluster.yaml
```

- Kubernetes in Docker (KinD) does what it says on the tin
- Excellent for security testing and trialling things
- Great for deploying older vulnerable clusters

EXERCISE 3 - API SERVER ACCESS

```
curl http://localhost:8080/
```

```
curl http://localhost:8080/api/v1/namespaces/kube-system/pods | jq
```

KUBELET API

- Listens on port 10250 by default
- Controls access to containers on a given host
- Can be used to run commands in a container
- Largely undocumented

EXERCISE 4 - KUBELET API

```
curl -k https://localhost:10250/
```

```
curl -k https://localhost:10250/pods | jq
```

```
curl -k https://rancher.steelcon.container.farm:10250/
```

KUBELET API - HANDY FOR ATTACKERS

- Direct access to the Kubelet bypasses admission control
- Also bypasses audit logging
- Service Accounts with `node/proxy` rights can access the API directly

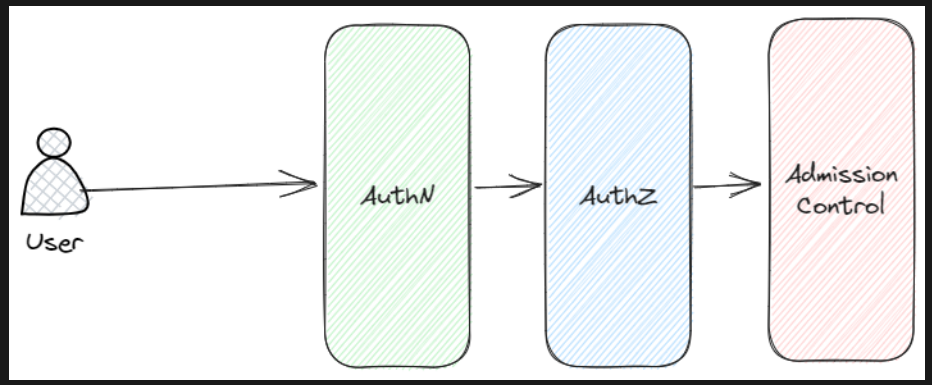
ETCD

- Runs on port 2379 by default
- Stores all cluster state
- Accessible over gRPC and HTTP
- Not accessible without authentication

CONCLUSION

- Kubernetes has a number of APIs that can be accessible.
- In modern clusters it *shouldn't* be possible to access them without authentication.
- Worth checking though, just in case!

KUBERNETES AUTHENTICATION


User
AuthN
AuthZ
Admission Control

KUBERNETES AUTHENTICATION

- Inbuilt Authentication Options
  - Static Token (Token Auth)
  - Client Certificates
  - Service Account Tokens
- Authentication Options that require outside systems
  - OpenID Connect Tokens
  - Webhook Tokens
  - Authenticating Proxy
- N.B. Kubernetes does not in any circumstances actually have a user database… (well apart from the one they don’t talk about)

STATIC TOKEN AUTHENTICATION

- Static file on disk containing credentials
- Requires a restart of the API server to make changes
- Credentials are held in the clear on disk

CLIENT CERTIFICATE AUTHENTICATION

- Client certificates signed by the main Kubernetes CA
- User and group information encoded into the certificate
- Encoded in to Kubeconfig files for user authentication

EXERCISE 1 - CLIENT CERTIFICATE AUTHENTICATION

```
kind create cluster --config=kind_configs/kind-config.yaml
```

```
kubectl get po -A
```

EXERCISE 1 - CLIENT CERTIFICATE AUTHENTICATION - 2

```
docker exec -it kind-control-plane bash
```

```
cd /certs
```

```
openssl genrsa -out user1.key 2048
```

```
openssl req -new -key user1.key -out user1.csr -subj "/CN=user1/O=group1"
```

```
openssl x509 -req -in user1.csr -CA /etc/kubernetes/pki/ca.crt -CAkey\
 /etc/kubernetes/pki/ca.key -CAcreateserial -out user1.crt
```

```
chmod 777 *
```

```
exit
```

EXERCISE 1 - CLIENT CERTIFICATE AUTHENTICATION - 3

```
kubectl config set-credentials user1\
 --client-certificate=/home/ubuntu/certs/user1.crt\
 --client-key=/home/ubuntu/certs/user1.key --embed-certs=true
```

```
kubectl config set-context user1@kind\
 --cluster=kind-kind --user=user1
```

```
kubectx user1@kind
```

```
kubectl get po
```

AUTOMATING CLIENT CERT CREATION

- We can use Teisteanas to automate the creation of client certificates
- https://github.com/raesene/teisteanas

```
teisteanas --username user2
```

- Creates a file called `user2.config` in the current directory
- then :-

```
kubectl --kubeconfig user2.config get pods
```

USING CLIENT CERTIFICATES WITH CURL

- Also possible to use client certificate authentication with curl e.g.

```
cd /home/ubuntu/certs
```

```
curl -sk --cert user1.crt --key user1.key https://127.0.0.1:40000/api/
```

```
curl -sk --cert user1.crt --key user1.key https://127.0.0.1:40000/api/v1/pods
```

BRIEF ASIDE - CERTIFICATE MANAGEMENT

- One of the generally unsolved problems in Kubernetes security
- Default setups store the certificate authority private key in the clear on disk on the API server
- Access to this file provides a persistent cluster backdoor
  - Default CA certificate lifetime is 2-10 years
- Certificate authentication is required for operation
  - Component to component authentication.
- Protecting the key is very important.

USER TOKENS

- Look at your Kubeconfig file
- There's a token that Rancher generates and uses
- This is Rancher-specific, but similar batters are common

SERVICE ACCOUNT TOKENS

- Intended for use by applications running in the cluster
- In older versions of k8s, these were non-expiring static tokens stored as secret objects
- In newer version of k8s, they are short lived tokens generated by the API server

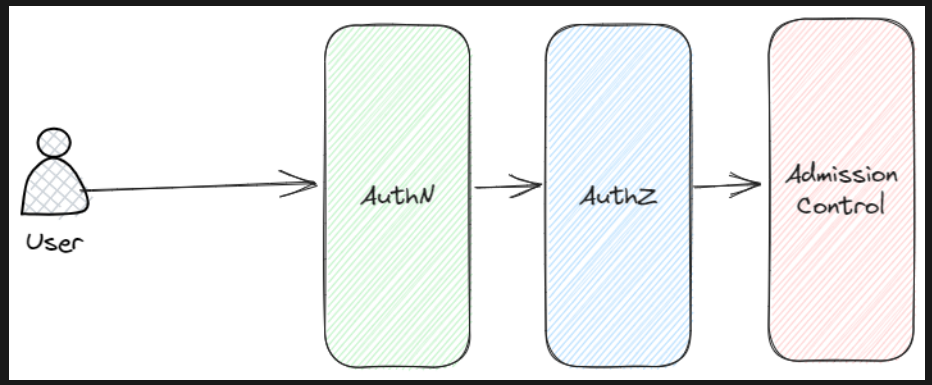
ATTACKING AUTHENTICATION

- Typically the best way to do this is stealing Kubeconfig files
- Depending on the authentication method used it may be static credentials
- Temporary access to admin creds allows for new users to be created
  - Notable exception of EKS where this is not possible

MODULE CONCLUSION

- Kubernetes has a number of authentication options
- In-built options are *not* suitable for production
- For external use, options tend to vary per distribution

KUBERNETES AUTHORIZATION


User
AuthN
AuthZ
Admission Control

INTRODUCTION

- Once we've authenticated the user, we need to sort authorization
- Again a number of options
  - AlwaysAllow (this is bad)
  - RBAC (Role Based Access Control) - Current main option used inside Kubernetes
  - Webhook - Allows delegation of AuthZ decisions to an external service.

KUBERNETES RBAC

- Makes use of roles which describe a set of permissions to a resource and rolebindings which bind a role to a set of subjects
- subjects can be of three types
  - Users
  - Service accounts
  - Groups

KUBERNETES RBAC - SCOPE

- Resources are scoped in one of two ways
  - Specific namespace
  - Cluster-wide resources

KUBERNETES RBAC - BUILT-IN ROLES

- There are a number of built-in clusterroles
- Used to provide rights to service accounts
- Also provide some generic roles (e.g. `cluster-admin`)

RBAC - ASSIGNING RIGHTS

- ClusterRoleBinding --> ClusterRole == Rights assigned at cluster level
- RoleBinding --> Role == Rights assigned to one namespace
- RoleBinding --> ClusterRole == Rights assigned to one namespace

KUBERNETES RBAC - DEFAULT ROLES

```
❯ kubectl get clusterroles
NAME                                                                   CREATED AT
admin                                                                  2023-06-13T09:27:08Z
cluster-admin                                                          2023-06-13T09:27:08Z
edit                                                                   2023-06-13T09:27:08Z
kindnet                                                                2023-06-13T09:27:12Z
kubeadm:get-nodes                                                      2023-06-13T09:27:10Z
local-path-provisioner-role                                            2023-06-13T09:27:12Z
system:aggregate-to-admin                                              2023-06-13T09:27:08Z
system:aggregate-to-edit                                               2023-06-13T09:27:08Z
system:aggregate-to-view                                               2023-06-13T09:27:08Z
system:auth-delegator                                                  2023-06-13T09:27:08Z
system:basic-user                                                      2023-06-13T09:27:08Z
system:certificates.k8s.io:certificatesigningrequests:nodeclient       2023-06-13T09:27:08Z
system:certificates.k8s.io:certificatesigningrequests:selfnodeclient   2023-06-13T09:27:08Z
system:certificates.k8s.io:kube-apiserver-client-approver              2023-06-13T09:27:08Z
system:certificates.k8s.io:kube-apiserver-client-kubelet-approver      2023-06-13T09:27:08Z
system:certificates.k8s.io:kubelet-serving-approver                    2023-06-13T09:27:08Z
system:certificates.k8s.io:legacy-unknown-approver                     2023-06-13T09:27:08Z
system:controller:attachdetach-controller                              2023-06-13T09:27:08Z
system:controller:certificate-controller                               2023-06-13T09:27:08Z
system:controller:clusterrole-aggregation-controller                   2023-06-13T09:27:08Z
system:controller:cronjob-controller                                   2023-06-13T09:27:08Z
system:controller:daemon-set-controller                                2023-06-13T09:27:08Z
```

EXERCISE 1 - RBAC CLUSTER ROLES AND BINDINGS

```
kubectx kind-kind
```

```
kubectl get clusterroles -o yaml
```

```
kubectl get clusterrolebindings -o yaml
```

EXERCISE 1 - USER RIGHTS

```
kubectx user1@kind
```

```
kubectl get po
```

EXERCISE 1 - ASSIGNING RIGHTS TO USERS

```
kubectx kind-kind
```

```
kubectl create clusterrolebinding group1-binding --clusterrole=cluster-admin --group=group1
```

EXERCISE 1 - CHECKING NEW USER RIGHTS

```
kubectx user1@kind
```

```
kubectl get po -n kube-system
```

RBAC GOTCHAS

- Read-only access can be dangerous (specifically for secrets)
- Allowing Pod creation leads to privesc through a variety of routes (even with PSP enabled)
- Allowing impersonation rights
- K8s docs on privilege escalation https://kubernetes.io/docs/concepts/security/rbac-good-practices/

RBAC GOTCHA - THIRD PARTY INSTALLS

- Always be careful before applying RBAC rights to clusters as part of product installation
- They may do something you don't want like bind the default service account to `cluster-admin`
- https://github.com/spekt8/spekt8 has an example with https://raw.githubusercontent.com/spekt8/spekt8/master/fabric8-rbac.yaml

EXERCISE 2 - KUBERNETES PERMISSION AUDITING - MANUAL

```
kubectl get clusterrole cluster-admin -o yaml
```

```
kubectl get clusterrolebinding cluster-admin -o yaml
```

- Note that if you review the clusterrolebindings you won’t see any mention of user1

EXERCISE 2 - KUBERNETES PERMISSION AUDITING - KUBECTL

```
kubectl auth can-i get pods
```

```
kubectl auth can-i --list
```

```
kubectl auth can-i --as ServiceAccount:kube-system:node-controller get pods
```

RBAC AUDITING - TOOLS!

- Good range of tools to help assess RBAC rights
- Quite a few unmaintained (surprise :P)
- rbac-tool is a good one
  - https://github.com/alcideio/rbac-tool

EXERCISE 3 - KUBERNETES PERMISSION AUDITING - RBAC-TOOL

```
rbac-tool who-can get secrets
```

```
rbac-tool analysis
```

ATTACKING AUTHORIZATION

- Depending on your rights, you can escalate privileges
- Often users will have create pod rights
- Opportunities for privesc.

EXCERCISE 2 - GETTING ROOT ON A NODE

```
kubectx kind-kind
```

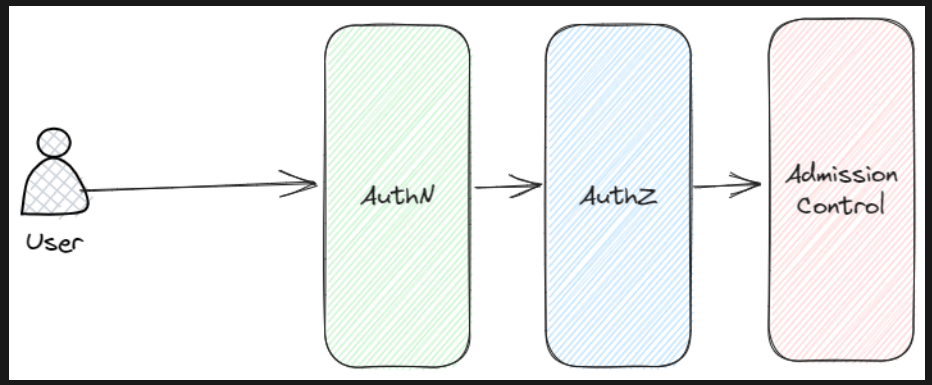
```
kubectl create -f /home/ubuntu/manifests/noderootpod.yaml
```

```
kubectl exec -it noderootpod -- chroot /host
```

CONCLUSION

- RBAC is the main authorization mechanism in Kubernetes
- Some risks for cluster operators of privilege escalation
- Without more controls most users can compromise clusters using standard Kubernetes functionality

KUBERNETES ADMISSION CONTROL


User
AuthN
AuthZ
Admission Control

ADMISSION CONTROLLERS

- Once Authentication and Authorization gates are passed, there is one more step before a resource is deployed to a cluster, Admission controllers.
- Admission controllers can modify workloads before they launch or block their them.

TWO TYPES OF ADMISSION CONTROLLERS

- Mutating Admission Controllers
  - Modify the resource being created
- Validating Admission Controllers
  - Validate the resource being created

POD SECURITY WITH ADMISSION CONTROLLERS

- One of the main roles of admission control is to enforce security on pods
- Helps to stop the attack we used in the authorization section
- In modern clusters there are a couple of main ways of doing this.
  - Pod Security Admission
  - External Admission Controllers

POD SECURITY ADMISSION

- Works by applying one of three levels of security to each namespace in a cluster
  - Privileged
  - Restricted
  - Baseline
- Not massively flexible but built-in to Kubernetes

EXTERNAL ADMISSION CONTROLLERS

- More flexible than Pod Security Admission
- Can be used to enforce a wide range of constraints on resources in a cluster
- Some popular examples:
  - Open Policy Agent Gatekeeper
  - Kyverno

EXERCISE - POD SECURITY ADMISSION

```
kubectx local
```

```
kubectl create -f /home/ubuntu/manifests/noderootpod.yaml
```

MODULE CONCLUSION

- Admission controllers are a key part of the Kubernetes security model.
- The in-built options are easy to use but not very flexible.
- External admission controllers are more flexible but require more work to setup and maintain.

KUBERNETES NETWORKING

OVERVIEW

- Kubernetes has a couple of network features that are "interesting"
- Typically the networking is built off of Linux features
- Bridges + iptables
- Also all cluster nodes are routers, so if you're on the same LAN as them you can route traffic via them :)

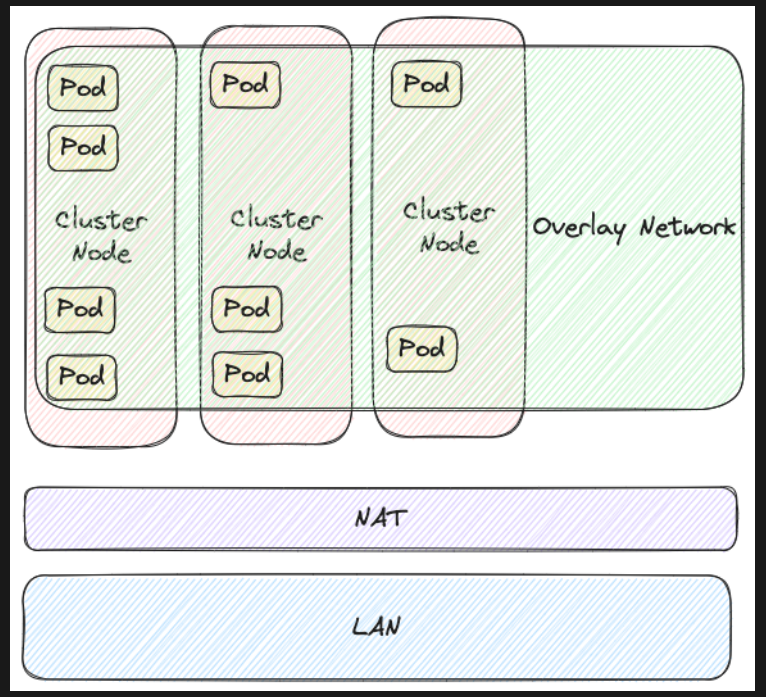
CNI

- Kubernetes does not provide networking itself
- Users use Container Network Interface Plugin(s) with each cluster
- Exactly how networking works, will depend on the plugin(s) used.

CNI OPTIONS

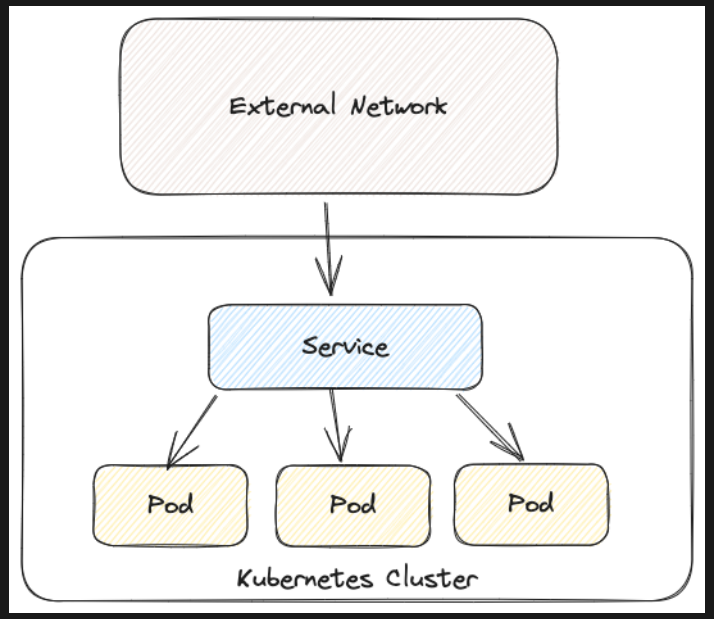
- Calico
- Cilium
- (Azure/AWS/GCP) networking
- ...

GENERAL NETWORKING MODEL



SERVICE NETWORKING

- Pods come and go, so we need another object for persistent networking
- this is the `service` object in Kubernetes
- All they actually are is .... iptables rules!
- redirect traffic to one of the currently running pods.


External Network
Service
Pod
Pod
Pod
Kubernetes Cluster

EXERCISE 1 - KUBERNETES IP ADDRESSES

```
kubectl get pods -o wide
```

```
kubectl get svc
```

```
docker exec kind-control-plane ip addr
```

KUBERNETES & DNS

- Another networking service that gets heavy use in Kubernetes is DNS
- Used for service discovery
- Predictable naming is also useful for enumeration!

EXERCISE 2 - ENUMERATING KUBERNETES RESOURCES WITH DNS

```
kubectx kind-insecurecluster
```

```
kubectl run -it dnstest --image=raesene/alpine-containertools -- /bin/bash
```

```
/scripts/k8s-dns-enum.rb
```

EXERCISE 3 - KUBERNETES IPTABLES

```
docker exec -it kind-control-plane bash
```

```
iptables -L -n -t nat
```

- Notice the Kubernetes service IP addresses and ports

RESTRICTING ACCESS IN A CLUSTER

- By default all pods in a cluster can communicate with each other
- If we want to restrict this, we can use Network Policies
- Essentially act a bit like Firewall ACLs but we can use Kubernetes names for source and destination

EXAMPLE NETWORK POLICY

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: web-deny-all
spec:
  podSelector:
    matchLabels:
      app: web
  ingress: []
```

EXERCISE 4 - NETWORK POLICIES - SETUP CLUSTER

```
kind create cluster --name=netpol --config=/home/ubuntu/kind_configs/kind-netpol-config.yaml
```

```
cilium install
```

EXERCISE 4 - NETWORK POLICIES - DEPLOYING THE APP

```
kubectl run web --image=nginx --labels app=web --expose --port 80
```

```
kubectl run -it netpoltest --image raesene/alpine-containertools  /bin/bash
```

EXERCISE 4 - NETWORK POLICIES - TESTING THE APP

```
curl http://web
```

```
exit
```

EXERCISE 4 - NETWORK POLICIES - APPLY A NETPOL

```
kubectl apply -f ~/netpol/deny-web.yaml
```

```
kubectl attach -it netpoltest
```

```
curl http://web
```

NETWORK POLICY GOTCHAS

- Use of "hostNetwork: true" will bypass network policies
- Network policies are not enforced by default
- Network policies are enforced by the CNI plugin, so the exact behaviour will depend on the plugin used.

CONCLUSION

- Kubernetes networking can be a bit complex due to the number of options
- At base a lot of it is just Linux network features
- Network policies are needed unless you like that old-school Flat LAN feel

KUBERNETES DISTRIBUTIONS

OVERVIEW

- Very few people run base Kubernetes in production
- Most times a distribution is used
- There are ... a lot ... of them

TYPES OF KUBERNETES DISTRIBUTIONS

- Managed Kubernetes
  - AWS EKS, Azure AKS, Google GKE
- Unmanaged Kubernetes
  - Kops, Kubespray, Kubeadm
- "Platforms"
  - OpenShift
  - Rancher
  - Tanzu

MANAGED KUBERNETES

- No access to the control plane nodes
- Provider chooses the configuration of the control plane
- Some options are exposed in the providers UI
  - This depends on the provider!

MANAGED KUBERNETES - DEFAULTS

- Defaults vary by provider
- Not always the most secure
- The big 3 all put the API server on a public IP by default
- Auditing may or may not be enabled
- ...

HONOURABLE MENTION - OPENSHIFT

- Red Hat OpenShift needs it's own slide
- Large platform built on top of Kubernetes
- highest level of variance from "base" Kubernetes
- Different security primitives
  - SCCs
- Lots and Lots of operators

CONCLUSION

- It's important to know that there are different distributions
- The implementations vary quite a bit
- Defaults are often different

CONTAINER SECURITY CHALLENGES

OUT OF THE BOX SECURITY

- By default it's optimized for ease of use not security.
- Hardening is needed at the Docker and Kubernetes level
- RCE as a service!

THREAT MODEL DIFFERENCES

- Some open source tools do not have the same threat model as enterprise software
  - places where having no auth. is not considered a problem
  - no support for non-repudiation
- Always consider how your threat model does/doesn't match up

CONTAINERS ARE EPHEMERAL

- We've seen that containers start and stop quickly and often leave no traces
- causes problems with logging
- causes problems with forensics

CONTAINER TECH IS (RELATIVELY) NEW

- This is still new tech. to a lot of companies
- problems with fitting it in to existing architectures

THERE'S A LOT OF VARIETY

- over 100 different distributions
- many different versions with different settings
- lots of plugin and software variety

WHO BUILT YOUR CONTAINER?

- Containers are just someone else's code on the internet
- It's important to know where your images come from
- Insert rant about image signing?

COMPLIANCE!

- There are now some standards
  - CIS benchmark
  - NSA Hardening guide
  - PCI Guidance
- They *cannot* cover all the scenarios
- This causes problems if compliance is dogmatic

CONCLUSION

- Containers introduce new security challenges
- The variety of the ecosystem definitely presents some problems
- The fast moving nature of the tech. also can be challenging.

CONCLUSION

THINGS TO REMEMBER

- A lot of what containers do is just linux (well apart from Windows containers)
- once you've got a handle on the core technologies it's easier to get started with new ones

MORE INFORMATION!

- container-security.site
- talks.container-security.site
- #SIG-Security & #kubernetes-security on Kubernetes slack
- #TAG-Security on CNCF slack

THANKS!

- Feedback always welcome
  - rorym@mccune.org.uk
  - @raesene@infosec.exchange
  - iain@iainsmart.co.uk
  - @smarticu5{@infosec.exchange}

LEARNING RESOURCES

BOOKS

- https://hello-tanzu.vmware.com/kubernetes-up-and-running-3/ - Free Kubernetes up and running book.
- Container Security - O'Reilly
- Hacking Kubernetes - O'Reilly
- N.B. The challenge of books is that they go out of date quickly

ONLINE RESOURCES

- Madhu Akula's Attacking and Auditing Docker Containers and Kubernetes Clusters
  - https://madhuakula.com/content/attacking-and-auditing-docker-containers-and-kubernetes-clusters/
- Kubernetes Goat - Interactive training env.
  - https://madhuakula.com/kubernetes-goat/
- Kube Security Lab - Set of KinD based challenges
  - https://github.com/raesene/kube_security_lab
- KillrCoda - interactive lab env
  - https://killercoda.com/

WEBSITES

- https://www.container-security.site/ - Container Security Site
  - https://www.container-security.site/general_information/reading_list.html - Reading List
- https://talks.container-security.site/ - Container Security Talks (300+!)