# Attacking Kubernetes Clusters

- An attack on Kubernetes generally starts from the perspective of a compromised pod.
- The threat actor reaches this point via a scenario similar to these:
  - Actor compromised the application running in one container in the pod.
  - Actor phished/compromised a person who had access to the pod.
  - Actor was authorized and wants to escalate their privileges.

- As a defender, once you can handle the compromised pod scenario, it's time to gain the ability to handle a compromised node.
  - Nodes are compromised either directly, through phishing/social engineering attacks, or through container breakouts.

## Attacks from within a Compromised Pod

An attacker in a pod may, among other things:

- Use the access provided by the pod to access other services`
- Attack other containers in their pod
- Make requests to the API server or a Kubelet to:
    - Run commands (possibly interactively) in a different pod
    - Start a new pod with privilege and node filesystem/resource access
    - Gather secrets that Kubernetes provides to pods
- Connect to the Kubernetes dashboard to perform actions
- Interact with the etcd server to change the cluster state
- Interact with the cloud service provider using a cluster account.

# Microsoft's Threat Matrix for Kubernetes

| 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 | |

## Defense: Overarching Note

You must upgrade your Kubernetes cluster.

Kubernetes development is active and moves very quickly.

The Kubernetes project supports only the last year's worth of releases. If a cluster is more than 12 months old, it may very well no longer have security patches available.

Before the third quarter of 2020, Kubernetes only supported 9 months of releases.

Additionally, the Kubernetes security defaults and capabilities continue to improve.

## Defense: RBAC and Authorization (Authz)

- Role-based Access Control (RBAC)
- Removing default service account permissions

## Role-Based Access Control

- You can place restrictions on the API server via RBAC.
- RBAC defines what PRINCIPALS can perform what ACTIONS.
- Principals are users or service accounts.
  - Example: [ jay in group system:authenticated ]
- Actions are VERBS combined with OBJECT types:
  - Example: [ create namespace ]
  - Example: [ in a specific namespace, create apps deployment]

## Role-Based Access Control: Roles

- You provide the ability to do these things by creating:
  - Role – specifying a list of actions
  - Role Binding – allowing a principal to use a role (list of actions).
- Roles have a many-to-many relationship with principals.
- Roles and Role Bindings are scoped to a namespace.
  - To scope globally, use Cluster Roles and Cluster Role Bindings.

# Create a Role and RoleBinding

```
kind: Role
apiVersion: …
metadata:
   name: ing-pod-reader
   namespace: inguardians-ns
rules:
- verbs: ["get","list"]
  apiGroups: [""]
  resources: ["pods"]
```

```
kind: RoleBinding
apiVersion: …
metadata:
   name: frontend-pod-reader
   namespace: inguardians-ns
roleRef:
  kind: Role
  apiGroup: …
  name: ing-pod-reader
Subjects:
- kind: ServiceAccount
  apiGroup: …
  name: frontend
```

## Creating Custom Roles Automatically

Jordan Liggitt wrote a tool called Audit2RBAC, similar to Audit2Allow for SELinux.

https://github.com/liggitt/audit2rbac/

Watch this in action via this video:

https://www.youtube.com/watch?v=n2cD20moYe8&feature=youtu.be

## Default Service Account Permissions

Once you have custom service accounts defined and working, remove permissions on the default service accounts.

Reference:

https://kubernetes.io/docs/tasks/configure-pod-container/configure-service-account/

# Exercise: Kubernetes Own the Nodes

We're going to do our first Kubernetes exercise now.

Please:

Open the Firefox browser on the class machine to:
http://localhost:10000/exercises/kubernetes-own-the-nodes

You may need to replace your daemonset file. Please find a replacement in the repo.

## Replacement daemonset

```
containers:
  - name: attack-root
  image: k8s.gcr.io/redis:e2e
  resources:
    limits:
      memory: 100Mi
      cpu: 1m
    requests:
      cpu: 1m
      memory: 100Mi
```