|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Topic** | **What to test** | **How to test (command)** | **Expected output & how to check** | **Risk if vulnerable** |
| cap\_sys\_admin (SYS\_ADMIN) | Presence of SYS\_ADMIN capability in pods; ability to mount host FS or bypass LSMs | kubectl exec -n <ns> <pod> -- capsh --print  kubectl get pod <pod> -n <ns> -o yaml | yq '.spec.containers[].securityContext.capabilities' | capsh output shows 'Capabilities: SYS\_ADMIN' or manifest adds SYS\_ADMIN. Vulnerable if SYS\_ADMIN present - can mount host, bypass AppArmor/SELinux. | Host compromise, mount host filesystem, bypass LSMs, near-root access. |
| cap\_sys\_ptrace (SYS\_PTRACE) | Presence of SYS\_PTRACE; ability to ptrace other processes and dump memory | kubectl exec -n <ns> <pod> -- capsh --print  kubectl exec -n <ns> <pod> -- gcore <PID> (if shared PID namespace) | capsh shows SYS\_PTRACE. Vulnerable if present and PID namespace shared—able to dump process memory and steal secrets. | Credentials and secrets leakage, process memory disclosure. |
| cap\_dac\_override (DAC\_OVERRIDE) | Capability that bypasses file permission checks allowing reading protected files | kubectl exec -n <ns> <pod> -- capsh --print  kubectl exec -n <ns> <pod> -- cat /host/root/.ssh/id\_rsa (if host mount exists) | capsh shows DAC\_OVERRIDE. Vulnerable if present and host mounts accessible — can read files regardless of perms. | Disclosure of private keys, sensitive files; lateral movement and persistence. |
| cap\_sys\_module (SYS\_MODULE) | Ability to load kernel modules from inside container (rare but critical) | kubectl exec -n <ns> <pod> -- capsh --print  # destructive: attempt insmod rootkit.ko (do NOT run without permission) | capsh shows SYS\_MODULE. Vulnerable if host kernel allows module loading — attacker can load rootkits or tamper kernel. | Kernel compromise, persistent rootkits, complete host takeover. |
| **Topic** | **What to test** | **How to test (commands)** | **Expected output & how to check** | **Risk if vulnerable** |
| Namespaces | Namespace isolation; cross-namespace access; RBAC & NetworkPolicies | kubectl get ns  kubectl auth can-i --list --namespace <ns> --as system:serviceaccount:<ns>:<sa>  kubectl get networkpolicies -A | Namespaces list; 'auth can-i' shows allowed verbs. Vulnerable if SAs perform cross-namespace actions or NetworkPolicies missing/allow all. | Lateral movement, data access across workloads, privilege escalation. |
| Docker & Container Runtime Exploits | Docker socket access; insecure capabilities; outdated runtimes | ls -l /var/run/docker.sock (node)  kubectl get pods -A -o jsonpath='{range .items[\*]}{.metadata.name} {.spec.containers[\*].securityContext.privileged}{"\n"}{end}'  ps aux | grep containerd | docker.sock exists and is writable;  privileged containers true.  Vulnerable if docker.sock writable or privileged containers present. | Host compromise, container escape, full cluster takeover. |
| CVEs (images & components) | Known vulnerabilities in images and cluster components | trivy image <image> OR grype <image>  kubectl get nodes -o wide (check versions) | Lists CVEs with severity. Vulnerable if HIGH/CRITICAL CVEs affect running images/components. | Remote code execution, privilege escalation, data exfiltration. |
| Unrestricted Docker Pull (image policy) | Cluster can pull untrusted images from public registries | kubectl run testimg --image=alpine --restart=Never --rm -it --command -- sh -c "echo ok" | Pod pulls image from external registry. Vulnerable if arbitrary images can be pulled without admission controls. | Supply chain compromise — malicious images executed in cluster. |
| Kubernetes API Server | Authentication/authorization; exposed endpoints | kubectl cluster-info  curl -k https://<apiserver>:6443/version  kubectl auth can-i --list | Accessible API endpoints or anonymous access shown. Vulnerable if API exposed publicly or RBAC too permissive. | Full cluster control via API, data leakage, resource abuse. |
| Kubernetes etcd | etcd accessibility; plaintext secrets in etcd | ss -lntp | grep 2379 (control plane)  curl <http://127.0.0.1:2379/v2/keys/> --max-time 5 (if unsecured) | etcd listens; curl returns keys. Vulnerable if etcd accessible without TLS/auth or stores plaintext secrets. | Secrets disclosure, control plane compromise, persistent state access. |
| Pods running as root user | Pods/containers running as UID 0 | kubectl get pods -A -o jsonpath='{range .items[\*]}{.metadata.namespace} {.metadata.name} {.spec.containers[\*].securityContext.runAsNonRoot} {.spec.containers[\*].securityContext.runAsUser}{"\n"}{end}' | runAsNonRoot false or runAsUser 0. Vulnerable if many pods run as root or enforcement missing. | Escapes more dangerous; potential host-level access. |
| **Topic** | **What to test** | **How to test (commands)** | **Expected output & how to check** | **Risk if vulnerable** |
| Containers running as root user | Image defaults or runtime set to root | kubectl describe pod <pod> -n <ns> | grep -i SecurityContext -A3  # or inspect image locally with docker inspect <image> | SecurityContext shows root UID or no non-root user. Vulnerable if image default USER is root. | Privilege actions inside container, lateral movement, host mount abuse. |
| Network Policies | Pod segmentation and egress restrictions | kubectl get networkpolicies --all-namespaces  kubectl exec -n <ns> <pod> -- curl -sS http://<other-pod>:<port> | No NetworkPolicies or policies with podSelector: {} allow all. Vulnerable if internal traffic unrestricted. | East-west movement, data exfiltration, internal service abuse. |
| Secrets Management | Hardcoded secrets; secrets in configmaps; etcd encryption | kubectl get secrets -A -o jsonpath='{range .items[\*]}{.metadata.namespace} {.metadata.name} {.type}{"\n"}{end}'  kubectl get configmap -A -o yaml | grep -i password | Many secrets or base64 data decodable; passwords in configmaps. Vulnerable if secrets in configmaps or etcd unencrypted. | Credential theft, lateral access to services/cloud providers. |
| Admission Controllers | Presence/enforcement of PodSecurity/OPA/Gatekeeper | kubectl api-versions | grep admissionregistration  kubectl get validatingwebhookconfigurations | No validating/mutating webhooks or misconfigured policies. Vulnerable if no admission controls. | Unrestricted deployment of privileged or malicious workloads. |
| Logging & Monitoring | Audit logs, alerting, detection coverage | kubectl get pods -n kube-system | grep fluentd\|elasticsearch\|prometheus  # Check control plane audit-policy.yaml | Missing audit logs or low retention. Vulnerable if no auditable logs or alerts. | Delayed breach detection, poor incident response. |
| Image Vulnerability Scanning (registry) | Auto-scan images on push; registry access controls | curl -s <https://registry.example/v2/_catalog>  trivy image <registry>/<image>:<tag> | Unscanned/public images; scanner reports CVEs. Vulnerable if registry allows anonymous push or unscanned images. | Deploying vulnerable/malicious images; supply chain attacks. |
| Kubelet / Node Port & Metrics | Kubelet unauthenticated endpoints; NodePort exposure | curl -k http://<node-ip>:10255/  kubectl get svc --all-namespaces | grep NodePort | kubelet responds or NodePorts open externally. Vulnerable if kubelet unauthenticated/NodePorts expose services. | Remote command execution on nodes, info disclosure, service exposure. |
| ServiceAccounts & RBAC | Overly-permissive service accounts; wildcard roles | kubectl get clusterrolebindings,rolebindings -A -o yaml | grep -E "(\\*:|resources: \[\"\\*\"\])" -n  kubectl auth can-i create pods --as system:serviceaccount:<ns>:<sa> | Bindings granting cluster-admin or wildcards. Vulnerable if SA can escalate or access secrets. | Privilege escalation, full cluster compromise via SA tokens. |
| Launch privileged container |  | docker run -it --privileged --name my\_container ubuntu bash  docker run -it ghcr.io/myusername/myapp:latest |  |  |
| Launch pod as root user |  | kubectl apply -f root-pod.yaml  kubectl exec -it root-pod -- id |  |  |
| Malicious docker |  | git clone <https://github.com/medoix/malicious-docker> && cd malicious-docker docker build -t malicious-docker:latest . docker run malicious-docker:latest |  |  |