



Assessing IAM Exposure in GCP

Colin Estep
cestep@netskope.com
Netskope Threat Labs
November 30, 2020

Contents

Contents	2
Introduction	3
Identity and Access Management	3
Least Privilege	4
IAM Exposure Risk	4
IAM in GCP	5
Types of Roles	6
Service Accounts and Impersonation	7
IAM Exposure in GCP	7
Service Account Users	8
Risks Associated with Service Account Impersonation	8
The Solution	10
Direct Permissions	10
Basic roles	10
Set IAM Policy	11
Service Account Impersonation	11
Indirect Permissions	11
Creating the Graph and Generating Results	11
Determining the Scope of Service Account Access	14
Determining Indirect Permissions	14
Report for john@siftsec.com	14
Findings	15
Permission Mining Tool	16
Conclusion	16
Appendix A	17
Least Privilege Monitoring in GCP	17
GCP Access Scopes	17

Introduction

We all see the media regularly publish stories about accidental data exposure from storage buckets hosted by public cloud providers, which usually exposes sensitive files due to some misconfiguration of the bucket or file access controls. Another concern that has not received the same level of attention is the misconfiguration of Identity and Access Management (IAM) controls, which increases the risk of activity (either malicious or accidental) that could damage the environment. CodeSpaces went completely out of business in 2014 after an attacker took advantage of an IAM misconfiguration to delete all of their AWS infrastructure. While best practices have evolved since then to help prevent a total loss, there are still holes in the IAM auditing capabilities that we must address.

In this paper, we will discuss what the risk is and why it is significant. We will focus on Google Cloud Platform (GCP) in particular, although the risk applies to other cloud providers as well. We will illustrate the problem and outline a successful approach to continuously monitor this risk. The approach discussed in this document centers around the ability to answer this question: What is the full scope of permissions granted to all of the identities in my cloud environment?

Answering this question means that we will have a full view of the privilege escalation and lateral movement potential in our cloud environments. This view of the environment will not only tell you what permissions your identities have directly applied to them, but also what permissions they can access indirectly. The indirect permissions are obtained through impersonation or by using credentials associated with cloud resources. The most egregious example is that of so-called 'shadow admins', users who are indirectly granted who have full admin privileges. The full view of potential permissions will reveal who is able to obtain an unintended level of privilege and give you the chance to refine the permissions before there's an incident.

Identity and Access Management

In public cloud providers, administrators must configure authorization for the identities that will interact with cloud services. For example, if you have users that need to launch and configure virtual machines, you will grant those permissions to those users via Identity and Access Management (IAM) policies. The IAM service will verify that any action taken by an identity has been authorized by a policy before it's allowed to take effect. In a real environment, determining which permissions are appropriate can become complex. It may require multiple attempts to arrive at a configuration that will allow the identity to accomplish its tasks without giving the identity more access than required.

Least Privilege

One important concept that is often discussed when it comes to cloud access is the principle of least privilege. This principle states that identities should only be given the permissions required to do their jobs, and nothing more. If identities are given more than what they require, this could pose undue risk to the environment, either through accidental changes, or if their credentials were ever compromised. The concern here is that someone who doesn't need administrator level privileges could maliciously (or accidentally) expose data, revoke other people's access, or cause an outage.

While this principle is often mentioned in documentation and discussed by cloud administrators, it is difficult to determine how to implement it in a real environment. Since the responsibilities are assigned differently in each organization, this means that each cloud environment is different. Cloud providers can't really provide a one-size fits all set of templates for every IAM policy.

Another factor that leads to difficulty in managing IAM permissions is the dynamic nature of cloud environments. As teams grow and change, permissions may need to change, and the changes are prone to error. It is often easier to assign broad permissions with the intent to refine them later to avoid blocking other teams. In addition, as new services are released, and resources are created or destroyed, the scope of any given identity's capabilities may change. These changes mean that administrators must monitor the permissions directly assigned to identities and modify them as needed.

Cloud providers are releasing services to help assess least privilege, such as IAM Recommender in GCP, which are meant to continuously provide feedback on what permissions are not being used. The services look at the last 30+ days to determine if the user is using all the privileges granted, and will show administrators which permissions are not being used. While this is useful, it will still allow the permissions to be assigned to the identity for as many as 90 days before even suggesting a change. In addition, these services are not really doing anything to help the assignment of permissions going forward. If someone has taken advantage of their overly liberal permissions by using them, then no change will be suggested to the administrator.

IAM Exposure Risk

As discussed in the previous section, limiting the permissions directly assigned to identities to a level of least privilege can be challenging. What is even more challenging is to assess the permissions that are indirectly accessible to identities.

By design, users can escalate their privileges in cloud environments. Much like the 'sudo' utility in Unix-based systems, this is a good practice for being able to conduct sensitive operations before returning to a less privileged state. However, in cloud environments there are multiple identities that could be impersonated by a user, which may or may not have privileges intended

for that user. In cloud environments, it can be a complex task to determine all of the identities a user is able to impersonate and even harder to determine the totality of permissions available to the user as a result.

Cloud providers also give us the ability to assign credentials to resources (such as virtual machines or applications) to allow them to interact with cloud services. This means that users may have access to the resources and can make use of the resource's credentials to escalate their privileges in a way we, the administrators, did not intend. At the time of writing, the cloud providers provide no services that allow administrators to easily discover or assess indirect permissions. The open-source tools generally focus on permissions directly applied to identities, so we are missing the full view of indirect permissions.

IAM in GCP

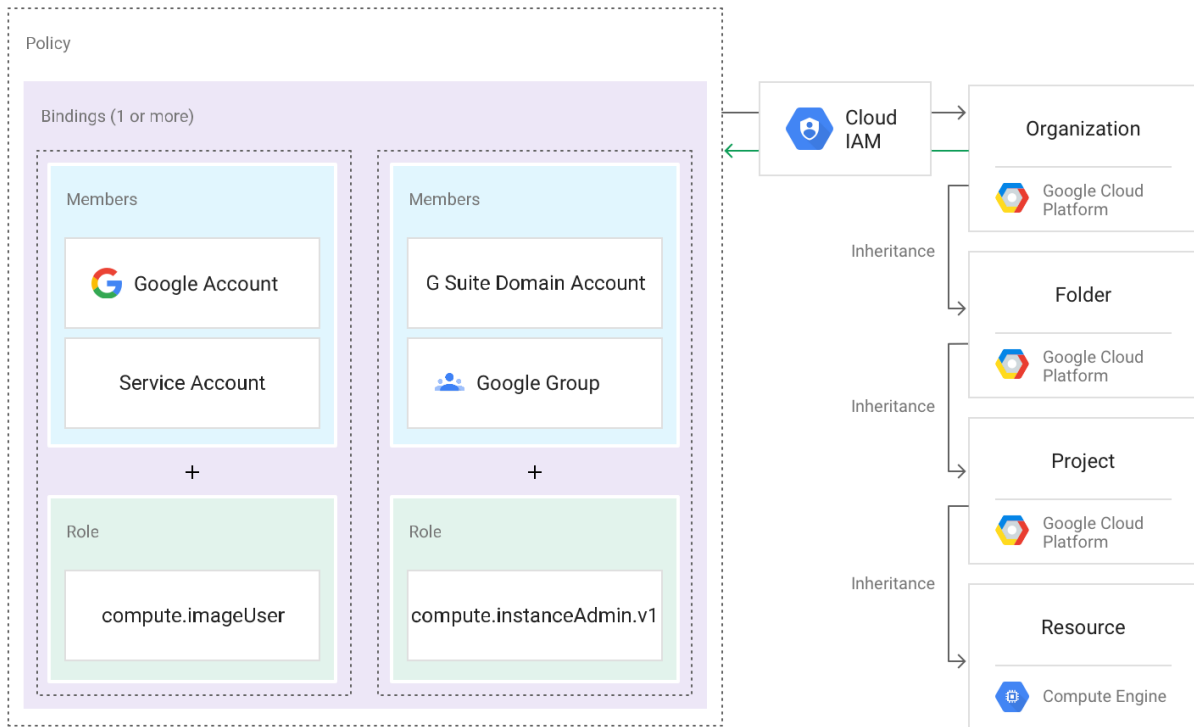
Having covered some general IAM concepts, this section will introduce IAM in GCP. It is necessary to understand the concepts in this section in order to understand the exposure risk in GCP, as well as our solution.

There are 4 different types of identities in GCP, which are called members:

1. User
2. Group
3. Domain
4. Service Account

They are all assigned permissions in GCP the same way, by assigning roles to them at various levels of the GCP organization structure. A role is just a collection of permitted API calls. When roles are assigned to an identity, it's called an IAM binding. As you can see in the diagram below¹, bindings can be applied at every point of the hierarchy, and the permissions are inherited down:

¹ Sourced from Google's documentation: <https://cloud.google.com/iam/docs/overview#cloud-iam-policy>



Bindings in GCP only grant permissions. There is no ability to explicitly deny access to something in an IAM policy, so once permission is granted at a high level, it cannot be negated. This means that administrators must understand this model and be very deliberate about which levels are used for assigning permissions. There are other types of controls that may mitigate some of these permissions, such as organization policies, but we will not be covering those since this document is focused on IAM.

Types of Roles

GCP contains the following types of roles:

- Basic
- Predefined
- Custom

Basic roles are provided by Google, but are not recommended for use², as they are fairly broad. Basic roles can have permissions to conduct thousands of different API calls, some of which have administrative capabilities. In particular, the “Owner” and “Editor” basic roles should be used carefully.

² Sourced from Google’s documentation: <https://cloud.google.com/iam/docs/understanding-roles>

Predefined roles are provided and managed by Google, and are meant to more specifically line-up with job roles. They focus on a particular service. For example, there is a role for the “Compute Instance Admin”.

If an administrator does not find a role that contains the permissions they would like to apply to their members, then they can create a custom role. These are generated and maintained by the administrator of the GCP environment.

Service Accounts and Impersonation

A service account in GCP is an identity that is provided to allow applications to authenticate and make calls to the GCP APIs. It does not use passwords, but uses RSA keys to authenticate. If you are familiar with IAM roles in AWS, this is the same concept for GCP. When you assign a service account to a resource, such as a virtual machine, Google generates and rotates the RSA keys automatically. This increases your security hygiene by making it easy to avoid storing static credentials in your code. However, GCP does allow you to manually create keys for a service account and use them to authenticate as the service account.

One important distinction to keep in mind for GCP is that a service account must be created in a project. However, a service account could be given bindings at a level higher than a project. The permissions could be granted at the Folder or even Organization levels. These are dangerous, because the permissions at higher levels could give the service account access to sensitive security controls that exist at the Organization level, and all the projects under the structure will inherit the permissions.

Resources, applications, users, and even service accounts can all impersonate a service account. Impersonating a service account means that you are able to authenticate as the service account, and now have whatever permissions were granted to it. API audit logs will show any actions as being performed by the service account.

IAM Exposure in GCP

It is important to have a well-architected design for your GCP environment up front with IAM inheritance in mind, so that you can grant the correct permissions to your members at the lowest possible level of the organizational structure. For example, Google recommends using Folders as isolation boundaries within your Organization³. In addition, administrators should strive to maintain least privilege by granting roles that contain the least amount of permissions required to complete the required tasks. However, a problem that still remains is how to effectively monitor the indirect permissions that each member can attain through service account

³ Sourced from Google documentation:
<https://cloud.google.com/resource-manager/docs/cloud-platform-resource-hierarchy#folders>

impersonation. This is a bigger challenge for most enterprises, because it's not only the least understood risk, but also the most difficult to discover.

Service Account Users

What are the permissions required to be able to use or impersonate a service account? According to Google, these are the main roles to be concerned with⁴:

- Service Account User - Can assign a service account to a resource, such as a virtual machine.
- Service Account Token Creator - Can impersonate a service account directly, including the ability to create OAuth2 tokens, signing blobs, or signing JSON Web Tokens (JWT) on its behalf.
- Service Account Key Admin - Can create and rotate RSA keys for a service account.
- Workload Identity User - Can impersonate service accounts used by Google Kubernetes Engine (GKE) workloads.

I would also add these:

- Editor - Includes everything from the Service Account User role
- Owner - Includes everything from the Editor role

The CIS benchmarks recommend assigning the Service Account User role to a user for a specific service account rather than assigning the role to a user at project level⁵. This means that the user can only impersonate a specific service account. However, if these permissions are granted at the project level or higher, then the user can enumerate and impersonate all service accounts under that structure. This would increase the likelihood of a member finding a way to escalate privileges, and it is a scenario that is missed by the IAM recommender service⁶. For example, if a member is granted service account token creator permissions at the project level, and that member only uses it for a single service account, the recommendation service won't alert you to this over-provisioning of permissions. This happens because the API calls are being used, and the binding is just applied at a wider scope than what's necessary.

Risks Associated with Service Account Impersonation

An administrator may assign bindings to a member intending to keep that member's permissions contained within a small part of the environment. However, a service account may have bindings elsewhere, which could allow the member to gain access outside of the intended area.

For example, I have a group of users that should have full access to the development resources, but not to our production resources. So, I assign the group a basic role of "Editor" at

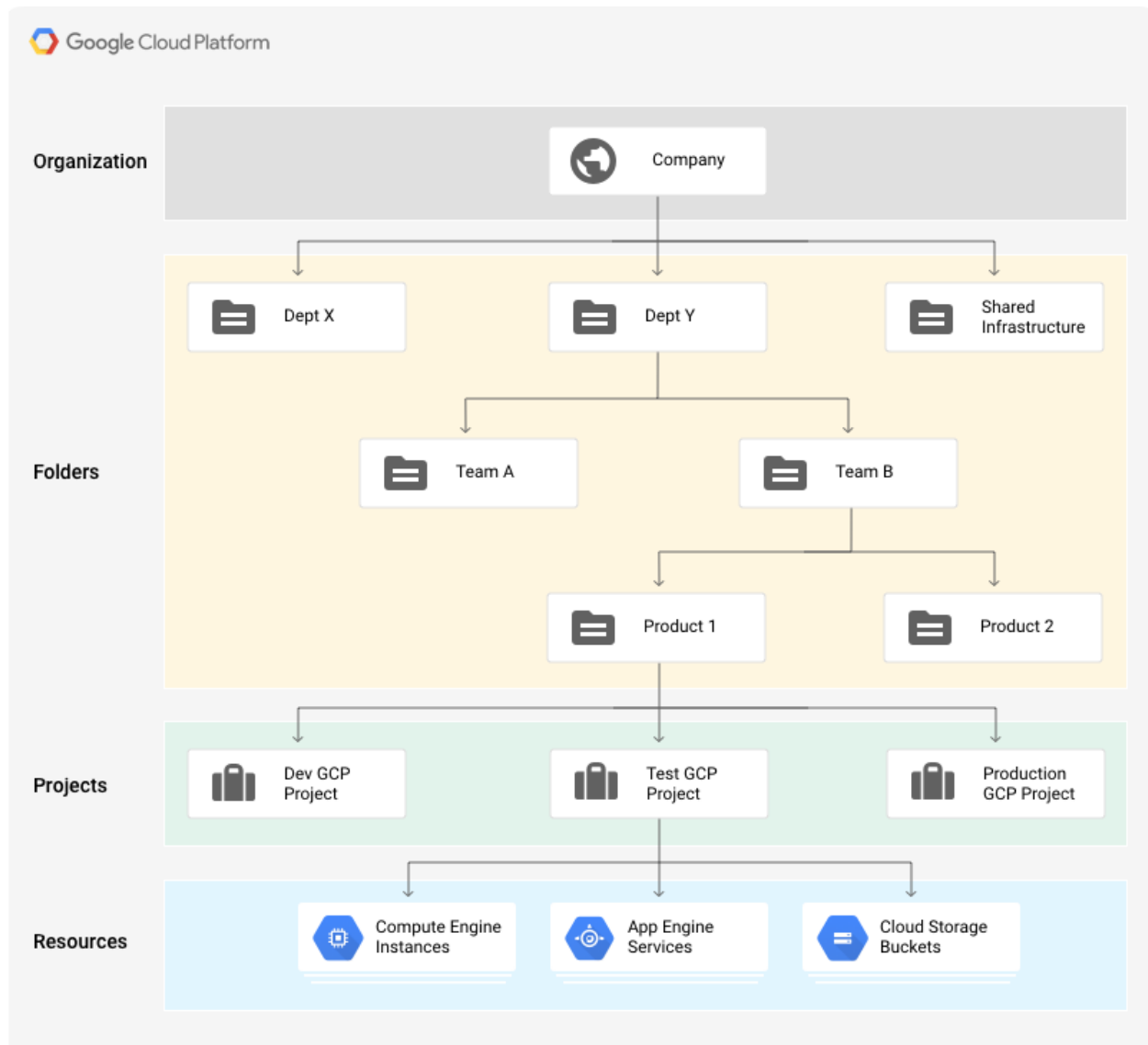
⁴ Sourced from Google's documentation:

<https://cloud.google.com/iam/docs/understanding-roles#service-accounts-roles>

⁵ Reference: [CIS Benchmark for GCP](#)

⁶ More information on IAM recommender available in the [Appendix](#).

the level of “Dev GCP Project” in the diagram below⁷. I expect that this group can only access resources within the project. However, that project contains a service account with bindings in the “Production GCP Project”. Therefore, any user in the group can effectively gain some level of access to the production project by impersonating the service account.



Another risk associated with impersonation includes privilege escalation. This means that a member was given limited permissions, but may be able to gain additional permissions by impersonating a service account that had more capabilities.

The final risk is actually a combination of the two previous ones, where a service account may have more privileges, as well as bindings that are in another part of the environment (or may

⁷ Sourced from Google documentation:
<https://cloud.google.com/resource-manager/docs/cloud-platform-resource-hierarchy#inheritance>

even be assigned at the Organization level). This means that a member with limited, granular access may be able to become a full administrator in the environment.

The Solution

While GCP and their recommended open-source tools address some IAM misconfiguration use cases, they do not provide full visibility into the potential for privilege escalation or lateral movement. The solution we designed provides the missing visibility to identify indirect permissions.

The solution uses API calls only and is intended to be repeatable and automated, so it can easily monitor the risk of the environment over time. Using the responses from the API calls, our solution maps-out the hierarchy of the overall environment. It also compiles an inventory of the service accounts and the IAM policies. The result is a compilation of all of the permissions assigned and where the bindings are located in the hierarchy.

Once all of the data is retrieved, we can perform detection in two main areas:

1. **Direct Permissions:** Identify members with sensitive permissions applied directly to them, which are described in the section above. This is a prerequisite for finding the indirect permissions and is straightforward. GCP and open source tools provide detections at this level as well.
2. **Indirect Permissions:** For members that have service account impersonation capabilities, compile the aggregate of the service accounts' permissions. This is the more difficult step that requires the right approach to do at scale. This phase is missing from GCP and open source tools.

What is deemed to be sensitive permissions should be tuned for each environment. However, there are some permissions we built into our solution that should be tracked in every environment.

Direct Permissions

Below are the types of permissions we deem sensitive to any environment, which our solution will look for in the permissions directly assigned to the GCP members.

Basic roles

Owner and Editor roles are able to change resources, manipulate service accounts, and more. Having either of these roles assigned above the project level is especially concerning, because they both have such broad capabilities.

An Editor can create and modify resources, enumerate all of the service accounts, and assign any of the service accounts to resources. An Owner can do everything that the Editor can do, as

well as set IAM policies (change permissions). This means that a member could change the permissions of a service account before assigning it to a resource, or grant another member access to the environment.

Set IAM Policy

A role that allows a member to set IAM policy is concerning, as this means that the member can add, change, or delete permissions for members in the environment. If this is assigned above the resource level, then the member can apply broader changes across your environment, which increases risk.

Service Account Impersonation

Service Account users can assign service accounts to resources, and then use the resources to take advantage of the service account's permissions. However, there are also roles that allow a member to directly impersonate a service account without using a resource. Both of these use cases should be carefully tracked to ensure that the full scope of risk is monitored. The identities that match this use case will be further evaluated, which is described in the Extended Permissions section.

Indirect Permissions

Once we have a full inventory of bindings and service accounts that exist, the extended permissions detection can evaluate the overall permissions the members have through impersonation. The solution examines the service account user / admin binding to determine at what level of the organization the binding exists. Once this is found, it needs to enumerate which service accounts exist at or below that level to determine what service accounts the member can access. Once the solution has a complete list of service accounts that are in scope, it uses the bindings for those service accounts to determine the effective permissions of each member.

To do this efficiently, the solution constructs a graph that contains only the members that have the ability to impersonate service accounts, as well as the service accounts, projects, and project ancestors. The member bindings are represented as edges between the members and the organization / folder / project / resource.

The solution uses the graph to traverse organizational relationships and discover which service accounts the members can access. Once we have a list of service accounts for the member, we can compute the total permissions available to the member.

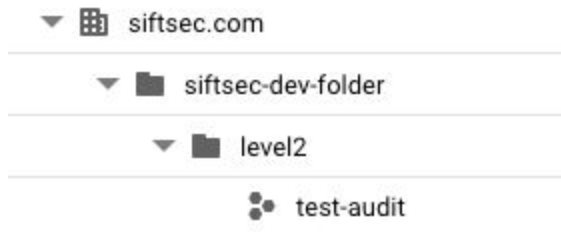
Creating the Graph and Generating Results

The graph creation happens in two main phases:

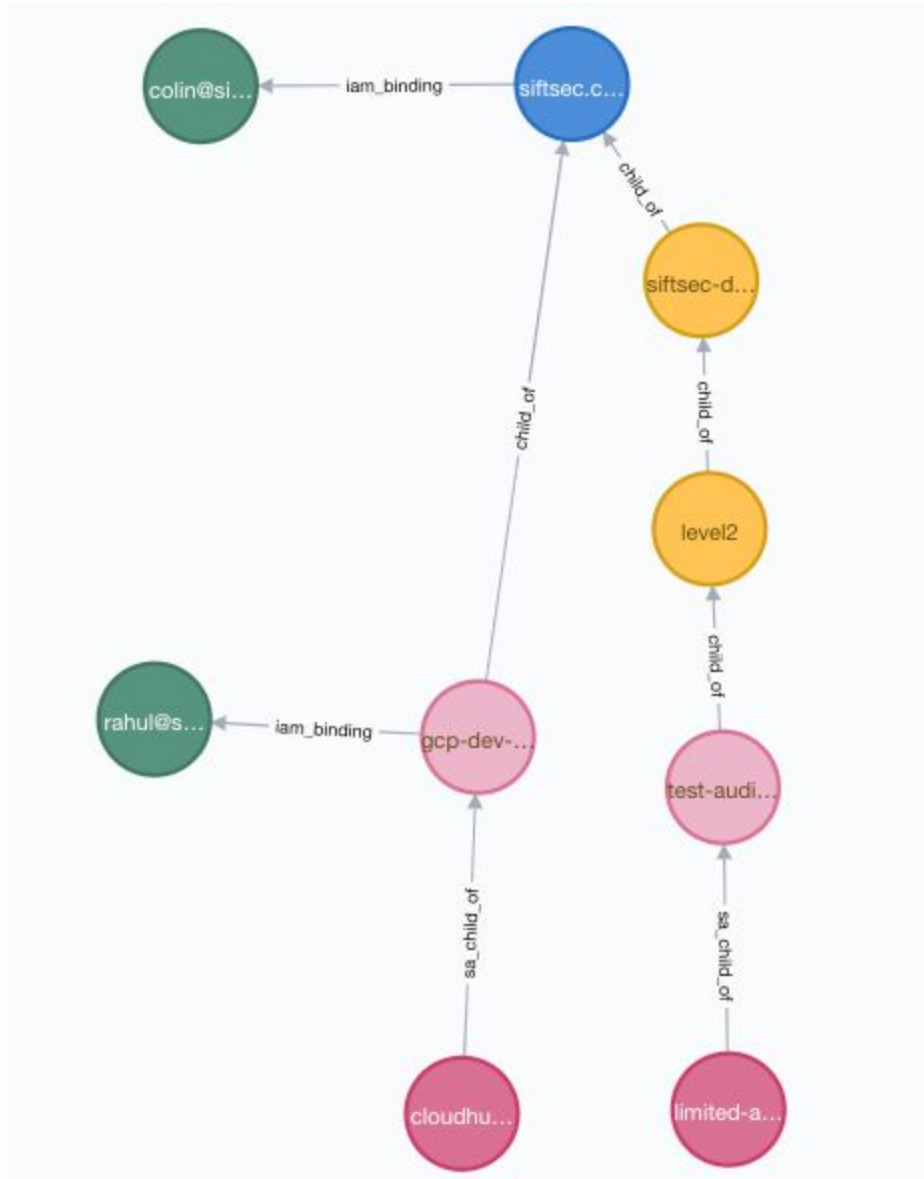
1. Add nodes and edges for organizational structure elements, which reflects the environment's hierarchy:
 - a. Organization

- b. Folder
 - c. Project
 - d. Service Accounts
2. Add nodes and edges to reflect the members and their IAM bindings at the pertinent levels of the environment's hierarchy.

The following image, taken from the GCP console, shows the structure in our GCP environment for the "test-audit" project:



A graph constructed by our solution for the same environment is shown in the image below.



Node Legend:

- Blue node - The GCP Organization
- Yellow nodes - GCP Folders (contains other folders or projects)
- Pink nodes - GCP Projects
- Green nodes - GCP members that have permission to impersonate a service account
- Red nodes - Service Accounts (as resources)

Edge Legend:

- “child_of” - Hierarchical relationship that results in IAM inheritance
- “sa_child_of” - Connects service accounts to the projects they are within
- “iam_binding” - Connects members with the hierarchical location of their bindings (only for bindings related to service account impersonation).

Determining the Scope of Service Account Access

Once the graph has been populated as shown above, we can easily traverse the graph to find all of the service accounts that each member can impersonate. After it has gathered all of the service accounts in scope, it will aggregate the full permissions set by examining all permissions assigned to those service accounts.

In this example, you can see the green node with the name “colin” has a binding at the organization level. This means that the “colin” member will have access to all the service accounts in this organization. The service accounts in this example are “cloudhunter” and “limited-access”. The “colin” member will also have access to any new service accounts created, regardless of the project they are created within. The “rahul” member in this example has a binding at the “gcp-dev” project level. Therefore, “rahul” only has access to impersonate the “cloudhunter” service account, he cannot impersonate “limited-access”. However, if other service accounts are created in the “gcp-dev” project, then he will have access to those.

Since “colin” can act as any service account, he may be able to escalate his privileges this way, and becomes a risk to the environment. We can quickly see that the binding at the organization level may be an unacceptable risk for our environment. The administrator may be able to substitute this for a binding further down the hierarchy, which will give “colin” more granular permissions.

Determining Indirect Permissions

After the graph is generated, the solution also computes the direct and indirect permissions for all members. The report produced shows the overall permissions for each member. The example below is completely separate from the example used in the previous section.

Report for john@siftsec.com

Direct Bindings

Binding Level	Name of Entity	Roles Granted
organization	'siftsec.com'	'roles/resourcemanager.organizationViewer', 'roles/viewer'
folder	-	-
project	-	-
service_account	'projects/siftsec-gcp-dev/serviceAccounts/colin-permissions-testing@siftsec-gcp-dev.iam.gserviceaccount.com'	'roles/iam.serviceAccountTokenCreator', 'roles/iam.serviceAccountUser'

compute_instance	'colin-instance-1'	'roles/compute.admin'
------------------	--------------------	-----------------------

Indirect Bindings

Binding Level	Name of Entity	Roles Granted
organization	'siftsec.com'	'roles/pubsub.editor', 'roles/iam.securityReviewer', 'roles/storage.objectViewer', 'roles/datastore.viewer', 'roles/browser'
folder	'siftsec-dev-folder'	'roles/pubsub.admin'
project	'colin-child-project'	'roles/compute.admin'
project	'siftsec-gcp-dev'	'roles/owner'
service_account	-	-
compute_instance	-	-

In the example above, the direct bindings only give john@siftsec.com the ability to do the following:

- Browse all resources in the organization
- Administer the 'colin-instance-1' virtual machine
- Impersonate the 'colin-permissions-testing@siftsec-gcp-dev.iam.gserviceaccount.com' service account

Based on the service account's bindings, john@siftsec.com has the potential to do a lot more, including:

- View all data stored in our storage buckets for the entire organization.
- Edit pub/sub topics in the entire organization
- Owner permissions in the 'siftsec-gcp-dev' project, which are basically administrator permissions, so he could modify permissions for other users, and create, modify, or delete any resources in that project

Findings

The solution described above has been applied to multiple environments, and enabled us to answer the following questions:

- What are the overall permissions of every member in the environment?
- Which members can impersonate service accounts?
- Which members can escape from a particular project, and make changes in another project?
- Which members are able to obtain administrator privileges in an unexpected way?

Our findings allowed us to more thoroughly apply the principle of least privilege by reducing the aggregate permissions that each member could obtain.

Permission Mining Tool

Netskope's IaaS Permission Mining Tool for GCP is freely available on [Github](#) and has been released to the open source community. Anyone with a GCP environment can download and use the tool, which makes API calls to retrieve information about members, IAM bindings, and the hierarchy of the environment. It uses the information obtained from GCP to construct a graph and discover which members have the ability to escalate their privileges via service account impersonation. The tool generates a report in JSON that contains direct and indirect bindings for members in the environment. GCP Administrators can use the JSON to easily discover sensitive permissions that should be removed.

Conclusion

In this document, we briefly reviewed Identity and Access Management in GCP. There is no straightforward way to see the full permissions (both direct and indirect) of every identity that exists in GCP. However, every enterprise using GCP should be aware of the potential for privilege escalation or lateral movement in their own environment. Understanding the permissions available to each identity allows the administrator to eliminate unwanted risk and maintain least privilege for their users.

The IaaS Permission Mining Tool for GCP was implemented by the author and used on various GCP environments to audit the permissions of all members in GCP. Our tool allows administrators to continually scan the environment for sensitive permissions and provides the information necessary to consistently manage the level of risk around their IAM configuration.

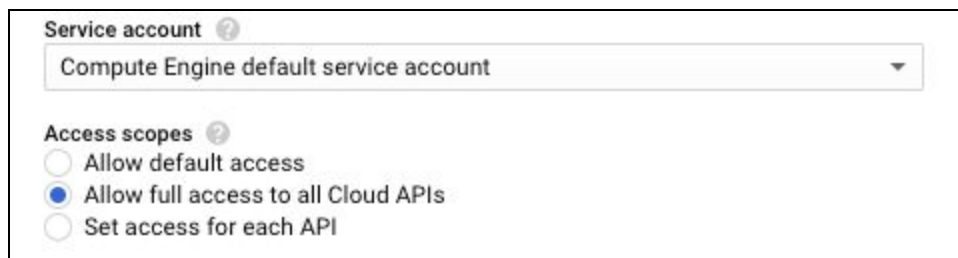
Appendix A

Least Privilege Monitoring in GCP

GCP released a service called Policy Intelligence, which includes IAM recommendations. The IAM recommendation engine is enabled in each project. It examines the previous 90 days of API calls and highlights which policies should be changed, based on the fact that permissions were not used in that period. Users can view the recommendations in the console, and can quickly apply the recommendations to reduce the permissions assigned to each member.

GCP Access Scopes

Access scopes are a legacy control for resources, such as virtual machines, that have been assigned a service account. This is a control that dictates which APIs the service account is able to access. So, if the service account is granted access to the Storage API in the IAM policy, the account still cannot access the Storage API if the access scopes do not include it. Here is an example the access scope options available in the GCP console:



The screenshot shows a GCP console interface with two sections. The first section is labeled "Service account" with a help icon. Below it is a dropdown menu showing "Compute Engine default service account". The second section is labeled "Access scopes" with a help icon. It contains three radio button options: "Allow default access", "Allow full access to all Cloud APIs" (which is selected), and "Set access for each API".

If a virtual machine has “Allow full access to all Cloud APIs” selected, then the service account can use all of its permitted API calls. This is configured per virtual machine, so your environment could have varied levels of access for its virtual machines.

Since Google recommends restricting access through IAM policies and opening access scopes to full access⁸, the solution presented in this document does not consider scopes as a mitigating control.

⁸ Sourced from Google documentation:
https://cloud.google.com/compute/docs/access/service-accounts#associating_a_service_account_to_an_instance