DeleFriend: Severe design flaw in Domain Wide Delegation could leave Google Workspace vulnerable for takeover

By Team Axon

Published: 2023-11-28 · Archived: 2026-04-05 16:41:24 UTC

By Yonatan Khen, Threat Hunting Expert at Team Axon

TL;DR:

- Hunters' Team Axon uncovered a design flaw in Google Workspace's Domain-Wide delegation feature, which results in misuse of existing delegations, potentially enabling privilege escalation and unauthorized access to Workspace APIs without Super Admin privileges. Team Axon has dubbed this design flaw as DeleFriend.
- This design flaw and research paper were responsibly disclosed to Google in advance as part of the "Bug Hunters" program in August 2023. As of this publication, the flaw remains active.
- Team Axon created a proof-of-concept tool included in this blog post to assist organizations in detecting misconfigurations and reducing DeleFriend's exploitation risks.
- This research paper delves into the Domain Wide Delegation feature, detailing its mechanism and potential for malicious use.
- The "Let's go hunting" section presents thorough threat hunting, detection techniques, and best practices for countering Domain-Wide delegation attacks.
- Read the press release here

Table of Contents:

Executive Summary

Over the past few years, there has been a notable shift towards cloud technology, with the development of cloud and SaaS tools designed to boost efficiency by integrating features that simplify work processes. One of these features is Google's Domain-Wide delegation. This feature permits a comprehensive delegation between Google Cloud Platform (GCP) identity objects and Google Workspace (GWS) applications. In other words, it enables GCP identities to execute tasks on Google SaaS applications, such as Gmail, Google Calendar, Google Drive, and more, on behalf of other Workspace users.

During 2023, Team Axon conducted extensive research on the Domain-Wide delegation feature and its potential implications for organizations. The research primarily focused on gaining a thorough understanding of the relevant attack surface, the service internals, and, most importantly, how it can be detected effectively.

This paper aims to illustrate how threat actors with varying privileges in a target GCP environment can potentially abuse Domain-Wide delegation. In addition, we'll also introduce a new proof-of-concept tool that allows for a full takeover of the Google Workspace domain using relevant GCP role permissions. With this tool, red teams, pentesters and security researchers can evaluate their security risks and improve the posture of their Workspace and GCP environments.

The final sections of the blog post offer actionable insights on digital forensics, threat-hunting methodologies, and detection engineering best practices based on telemetry logs from Google Cloud Platform and Google Workspace.

GCP <> GWS Authentication Fundamentals

Before delving into the feature internals, let's first discuss the relationship between GWS and GCP.

Domain-Wide delegation is based on the systemic connection between Google Cloud and Workspace, but aside from both being developed by Google, what is the connection between the two?

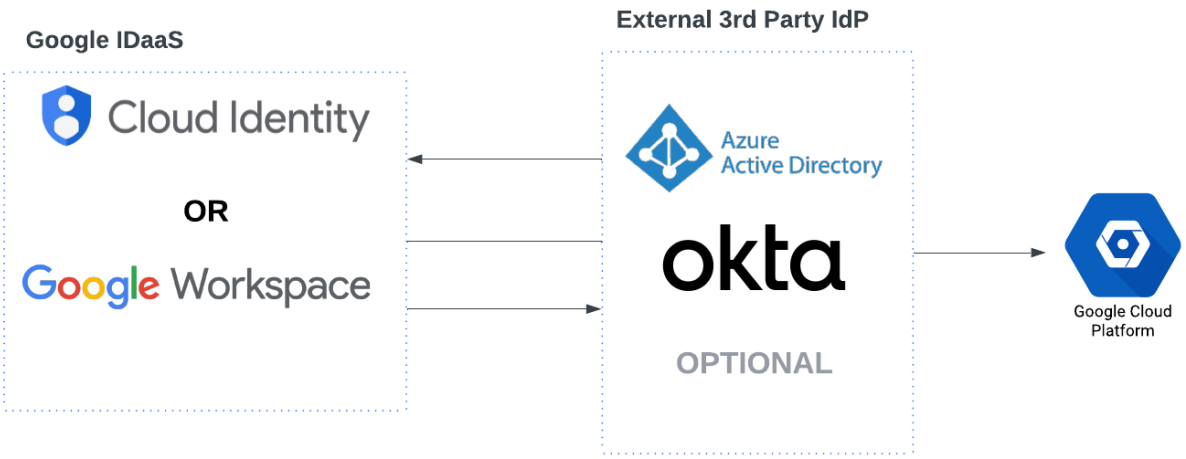
The Google ecosystem offers a robust suite of tools and services to meet the diverse needs of organizations. Two of these services - Google Cloud and Google Workspace (or Cloud Identity) - have distinctive roles but are intricately linked, particularly when we examine authentication.

The concept of integrating identity is key to setting up user authentication and access control for GCP services. While GCP IAM is capable of managing control and visibility for resources internally (for example, projects or individual compute resources), it does not dictate who can access the GCP itself.
When a business chooses to use Google Cloud, it needs a central 'hub' for managing all users, groups, settings, and data. This 'hub' (or 'directory') can either be Google Workspace or a Google Cloud Identity account. Google categorizes those with the term 'IDaaS' or identity-as-a-service.

Though Google Workspace is commonly known for its productivity tools like Gmail, it also has a crucial role in managing user identities across Google Cloud services. Google Cloud Identity plays as the cheaper option for organizations who like to use GCP, but do not necessarily need the productive applications offered by Workspace (for example organizations that are using O365).

Interestingly, even for organizations who are using third party identity providers (IdP), such as Okta or Azure AD, to manage their users and identities, Google still requires authentication through their IDaaS mechanisms. This

means organizations that want to use GCP services will still need to sync their third party IdP with either GWS or Cloud Identity directories.



Google's IDaaS concept

It's crucial to understand the interconnected nature of Google Cloud and Workspace, as this understanding underscores how the misuse of one could potentially impact the other.

Understanding Domain-Wide Delegation

Google Workspace's Domain-Wide delegation allows an identity object, either an external app from Google Workspace Marketplace or an internal GCP Service Account, to access data across the Workspace on behalf of users. This feature, which is crucial for apps interacting with Google APIs or services needing user impersonation, enhances efficiency and minimizes human error by automating tasks. Using OAuth 2.0, app developers and administrators can give these service accounts access to user data without individual user consent.

Google Workspace allows the creation of two main types of global delegated object identities:

- **GWS Applications:** Applications from the Workspace Marketplace can be set up as a delegated identity. Before being made available in the marketplace, each Workspace application undergoes a review by Google to minimize potential misuse. While this does not entirely eliminate the risk of abuse, it significantly increases the difficulty for such incidents to occur.
- **GCP Service Account:** Differently from the third party applications, Google also allows the creation of internal delegated identities in GCP, with identity-based service accounts. Those service accounts can leverage OAuth 2.0 to create temporary tokens and authorize against Google APIs, including access to Workspace REST APIs.

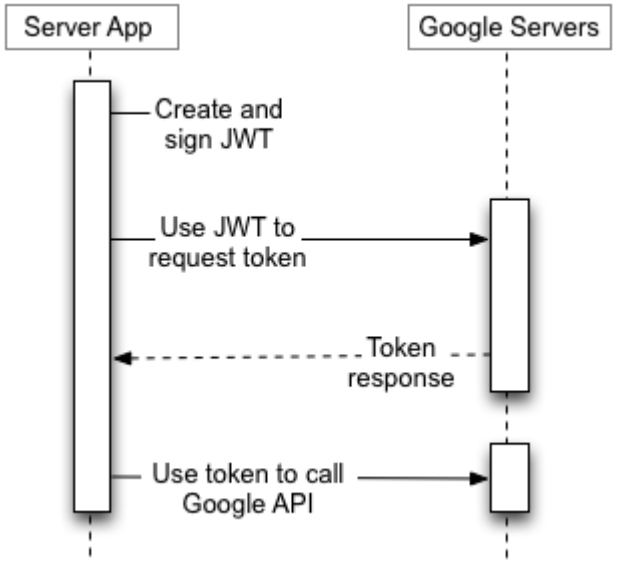
In this blog post, we will primarily focus on the misuse of delegation via GCP Service Accounts. As mentioned earlier, Marketplace applications represent a smaller attack surface due to the rigorous security audits that Google carries out before approving any new application.

Domain-Wide Delegation: Under the Hood

The next step is to understand how it works under the hood when leveraging a GCP service account private key pair to initiate a request to Google APIs on behalf of other identities in Google Workspace.

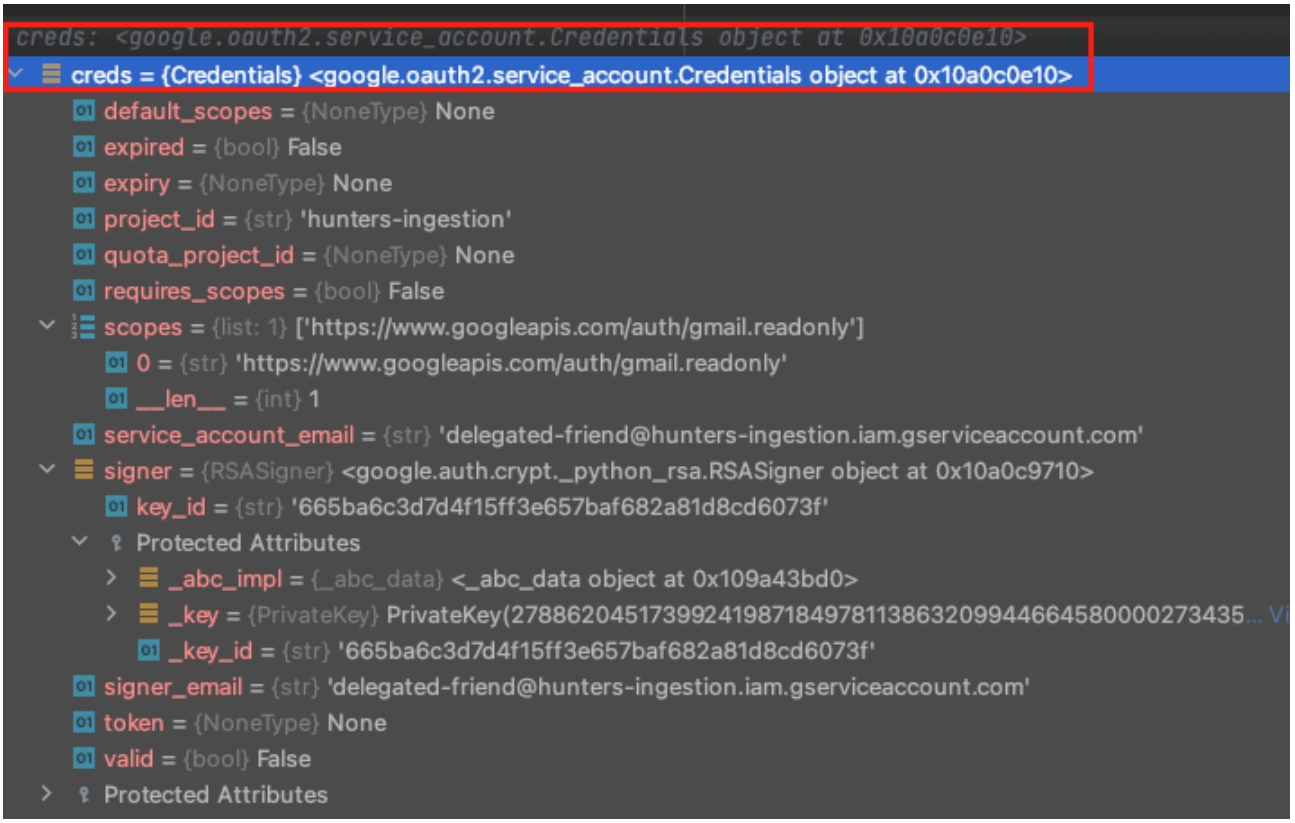
In terms of implementing a delegated authorization, Google aligns with other well-known Cloud providers by utilizing OAuth 2.0 RFC 6749. The fundamental concept here involves allowing an identity to grant permission to different Workspace REST API applications, without the need to expose his credentials.

Let’s see how it works. The following diagram illustrates the steps created by the delegated identity to access Google API.



OAuth 2.0 High-Level diagram

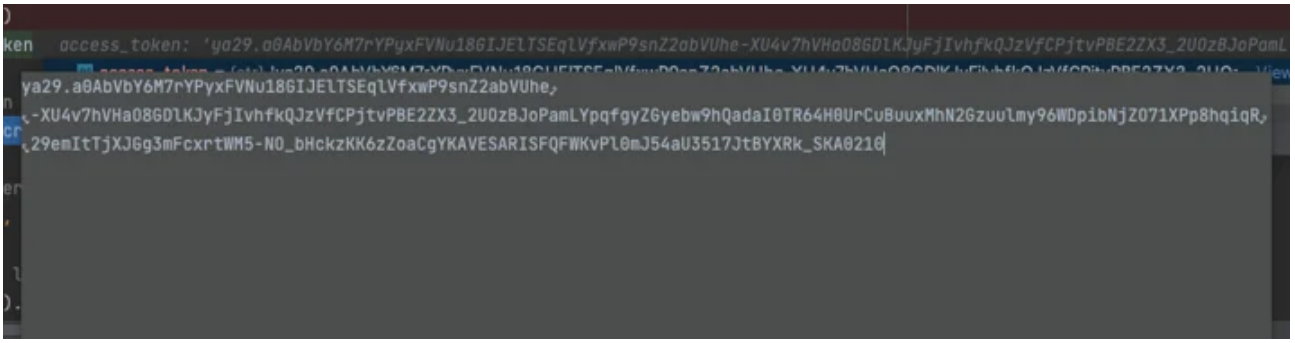
1. Identity creates a JWT: The Identity uses the service account's private key (part of the JSON key pair file) to sign a JWT. This JWT contains claims about the service account, the target user to impersonate, and the OAuth scopes of access to the REST API which is being requested.



Google Credential object that represents a JWT

2. The Identity uses the JWT to request an access token: The application/user uses the JWT to request an access token from Google's OAuth 2.0 service. The request also includes the target user to impersonate (the user's Workspace email), and the scopes for which access is requested.

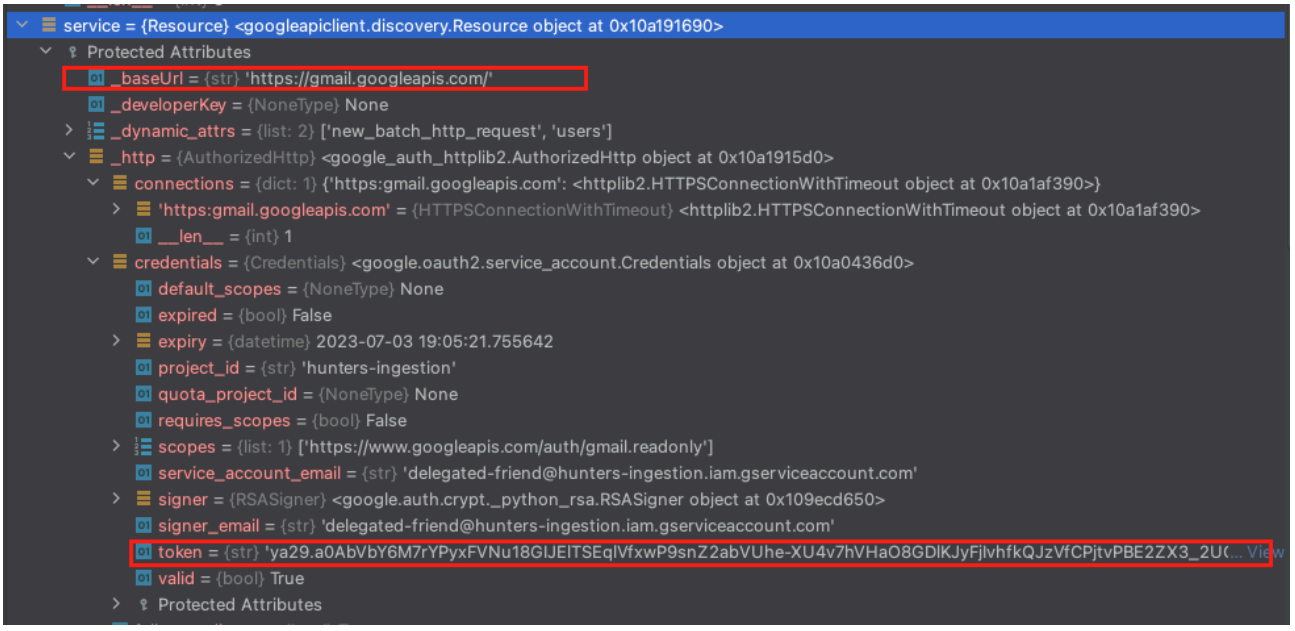
3. Google's OAuth 2.0 service returns an access token: The access token represents the service account's authority to act on behalf of the user for the specified scopes. This token is typically short-lived and must be refreshed periodically (per the application's need). It's essential to understand that the OAuth scopes specified in the JWT token have validity and impact on the resultant access token. For instance, access tokens possessing multiple scopes will hold validity for numerous REST API applications.



Example of an access token

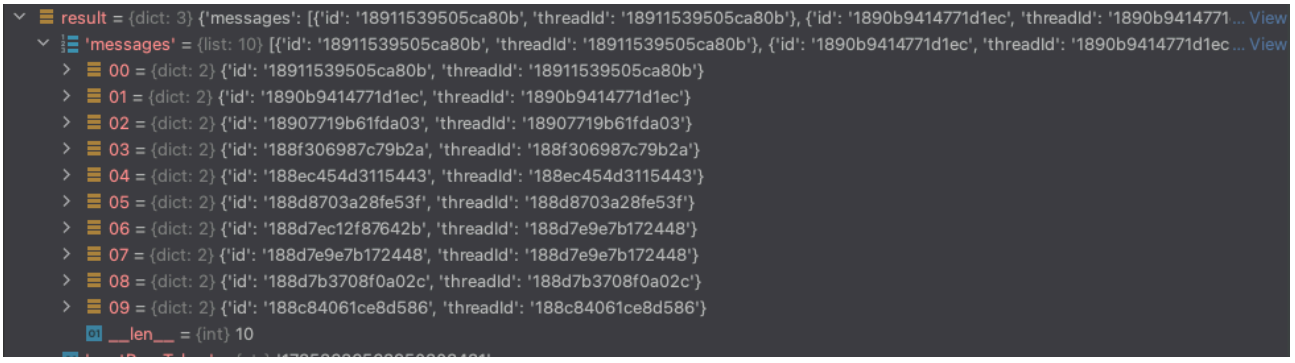
4. The Identity uses the access token to call Google APIs: Now with a relevant access token, the service can access the required REST API. The application uses this access token in the "Authorization" header of its HTTP requests

destined for Google APIs. These APIs utilize the token to verify the impersonated identity and confirm it has the necessary authorization.



API call to Google APIs using the returned access token

5. Google APIs return the requested data: If the access token is valid and the service account has appropriate authorization, the Google APIs return the requested data. For example, in the following picture, we've leveraged the *users.messages.list* method to list all the Gmail message IDs associated with a target Workspace user.



API results of users.messages.list

Abusing Domain-Wide Delegation

Up until now, we've discussed the feature internals, but have not touched on how it could potentially be abused. Let's review two scenarios that include different permissions of the target identity within GCP and Google Workspace.

The first scenario is a typical abuse of Domain-Wide delegation as an impactful post-exploitation method of creating a new delegation after gaining access to a Super Admin privilege on the target Workspace environment. This technique has been observed in the wild by Team Axon and has been known to be exploited by threat actors in recent years.

The second scenario is a new method to abuse existing delegations rather than creating a new one. Instead of requiring Super Admin privilege on the Workspace environment, the method requires less privileged access on the relevant GCP projects in order to enumerate successful combinations of service account keys and OAuth scopes. The concept will be detailed in the “DeleFriend″ section.

Scenario 1: New delegation with interactive GWS access

In the first scenario, an actor typically gains initial access to an IAM identity, with the ability to create service accounts in a GCP project. In addition, after extensive work in the target domain, he now holds a super admin privilege to GWS, and is looking for smart options to achieve strong persistence and exfiltration capabilities.

Attack steps:

1. **Generating a New Service Account and Corresponding Key Pair** The initial step starts with generating a new service account. On GCP, new service account resources can be produced either interactively via the console or programmatically using direct API calls and CLI tools. This requires the role *iam.serviceAccountAdmin* or any custom role equipped with the *iam.serviceAccounts.create permission*. Once the service account is created, we'll proceed to generate a related key pair. This key pair will consist of parameters that hold relevant information about the service account, including a private key. Basically, this key pair enables authentication of the service account. After this process, we can perform tasks using the identity of the service account, leveraging all the permissions that it's been accorded. The ability to generate a new key pair is facilitated by the *iam.serviceAccountKeys.create* permission.

Google’s Service Account (SA) private key, is a JSON file composed of the following:

- GCP Organization Metadata:
    - type: will always point to *service_account* object type.
    - project_id: the global GCP organization resource name. From our research, this attribute provides an additional level of classification and security to provide protection against impersonation attacks with the global service account ID.
    - private_key_id: unique hash identifier of the private key. This value is available in the GCP Console also after the private key creation.
    - private_key: private key value. This value won’t be available in the GCP console after the private key creation.
    - client_email: the GCP SA IAM email representation, a format of:

The default filename given by Google upon downloading the keys composed from a fixed naming format: *<gcp_project_id>-<private_key_id{12}>.json*. This format makes these files particularly attractive to red team members and potential attackers once they've gained access to a workstation or storage. We'll delve into methods for finding these files in the Threat Hunting Posture section.

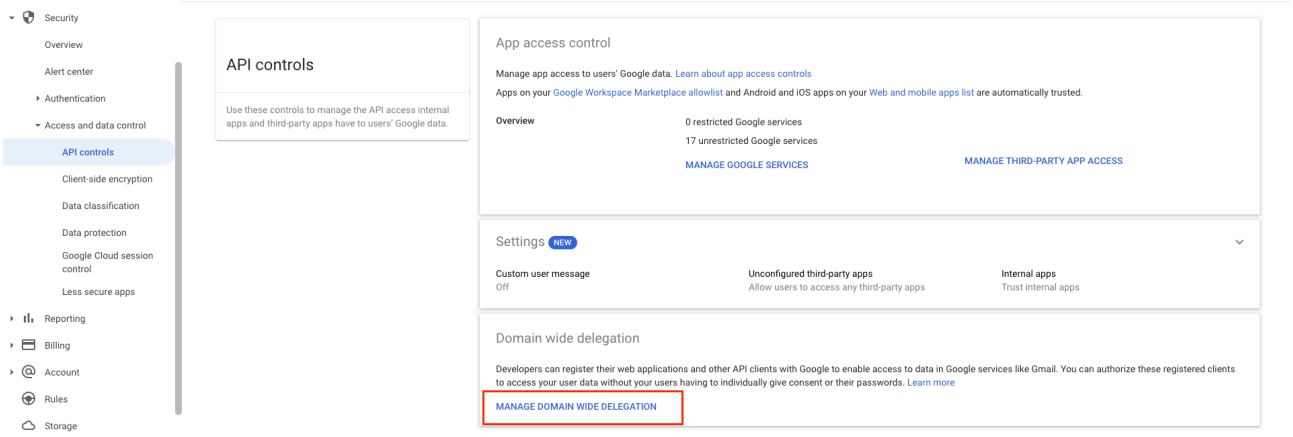
```
{
  "type": "service_account",
  "project_id": "hunters[redacted]",
  "private_key_id": "ec11ec94169050c256101ef9b7[redacted]",
  "private_key": "-----BEGIN PRIVATE KEY-----\nMIIEvgIBADANBgkqhkiG9w0BAQEFAASCBKgwggSkAgEAAoIBAQC9krQF8CyrZvx7[redacted]
  "client_email": "test-[redacted].iam.gserviceaccount.com",
  "client_id": "113933[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/t[redacted]",
  "universe_domain": "googleapis.com"
}
```


Service Account Private Key Pair JSON

2. Creation of new delegation: After having a relevant identity object and a related private key that enables authentication with Google APIs, we need to establish a new delegation rule for the service account resource within Google Workspace. This delegation rule will enable us to perform the Google APIs activity on Workspace REST API applications. It's important to understand that only the Super Admin role possesses the capability to set up global Domain-Wide delegation in Google Workspace.

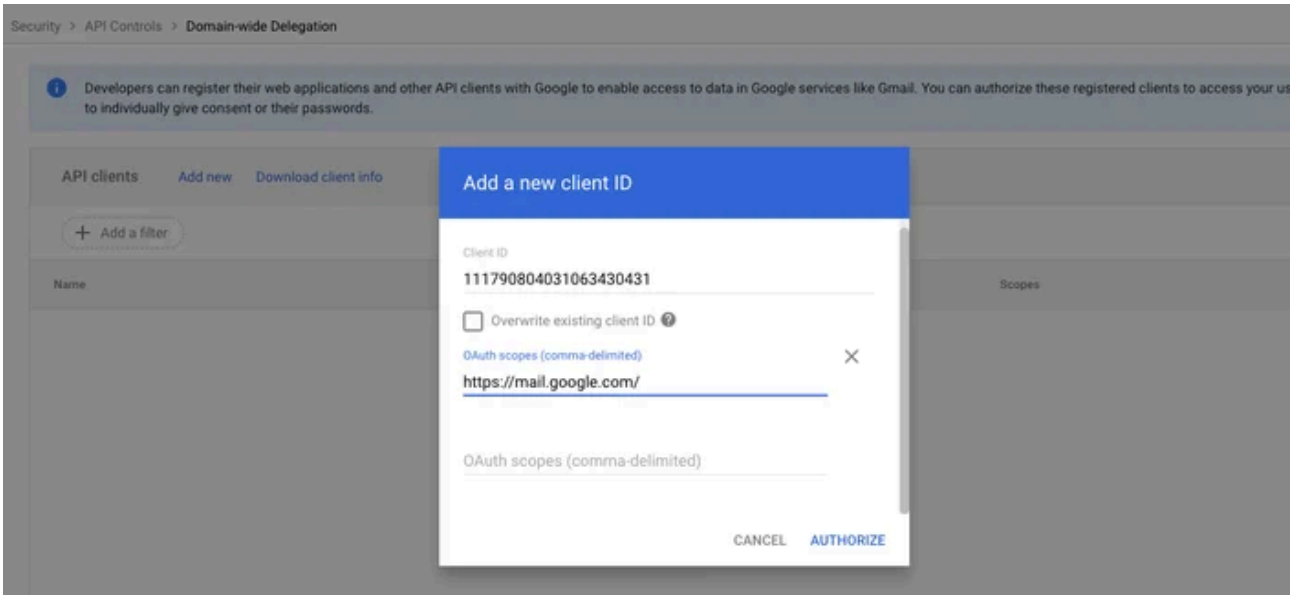
Furthermore, Domain-Wide delegation cannot be set up programmatically. It can only be created and adjusted manually through the Google Workspace console. Throughout our research, we tried both documented and undocumented methods of doing this, without any success. Interestingly, even admins who manage subdomains within Google Workspace cannot delegate permissions to applications and service accounts.

The creation of the rule can be found under the page API controls → Manage Domain-Wide delegation in Google Workspace Admin console.



API controls page in Google Workspace Admin Console

3. Attaching OAuth scopes privilege: When configuring a new delegation, Google requires only 2 parameters, the Client ID, which is the OAuth ID of the GCP Service Account resource, and OAuth scopes that define what API calls the delegation requires.



Setting domain-wide delegation configuration

OAuth scopes is a concept in Google APIs that limits an application's access to a user's account. Each scope represents a specific permission granted by the user to the application.

Scopes	
https://mail.google.com/	Read, compose, send, and permanently delete all your email from Gmail
https://www.googleapis.com/auth/gmail.addons.current.action.compose	Manage drafts and send emails when you interact with the add-on
https://www.googleapis.com/auth/gmail.addons.current.message.action	View your email messages when you interact with the add-on
https://www.googleapis.com/auth/gmail.addons.current.message.metadata	View your email message metadata when the add-on is running
https://www.googleapis.com/auth/gmail.addons.current.message.readonly	View your email messages when the add-on is running
https://www.googleapis.com/auth/gmail.compose	Manage drafts and send emails
https://www.googleapis.com/auth/gmail.insert	Add emails into your Gmail mailbox
https://www.googleapis.com/auth/gmail.labels	See and edit your email labels
https://www.googleapis.com/auth/gmail.metadata	View your email message metadata such as labels and headers, but not the email body
https://www.googleapis.com/auth/gmail.modify	Read, compose, and send emails from your Gmail account

OAuth scopes permissions in Google APIs example. The full list can be found here.

For example, if an application requests access to a user's Google Calendar data, the scope would be *https://www.googleapis.com/auth/calendar*. If the application also requested access to Google Drive, another scope *https://www.googleapis.com/auth/drive* would be needed.

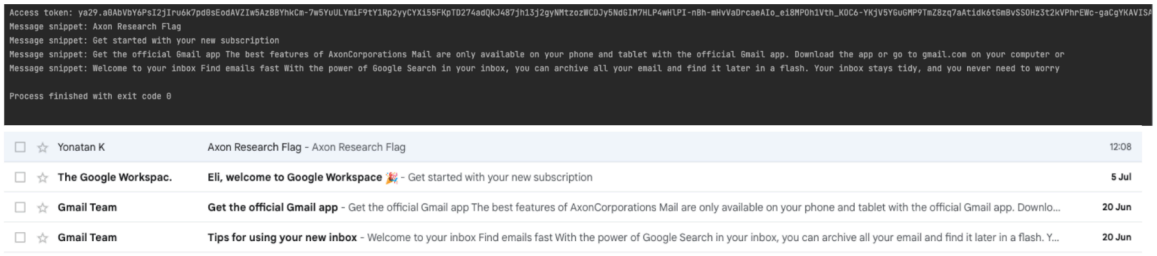
Each scope corresponds to a specific set of permissions, such as read, write, or delete. For instance, granting a read scope allows the application to view data but not modify it. Conversely, granting a write scope allows the application to modify data.

It's important to note that, for security purposes, scopes should be as limited as possible. The best practice of least privilege always should be applied, where an application only asks for the permissions it absolutely needs to function. We'll revisit that in the threat hunting section when speaking about posture best practices.

It is important to note that the delegation is attached to the service account identity itself in the form of the OAuth Client ID, and NOT for a specific key. This is important as we'll discuss how it can be abused in the next exploitation method.

4. Acting on behalf of the target identity: At this point, we have a functioning delegated object in GWS. Now, using the GCP Service Account private key, we can perform API calls (in the scope defined in the OAuth scope parameter) to trigger it and act on behalf of any identity that exists in Google Workspace. As we learned, the service account will generate access tokens per its needs and according to the permission he has to REST API applications.

In the given example, we utilize the users.messages.list API to iterate through and list the message IDs of emails in the target inbox. We then use users.messages.get to fetch the entire email content associated with the GWS Identity of a user named Eli Ohana.

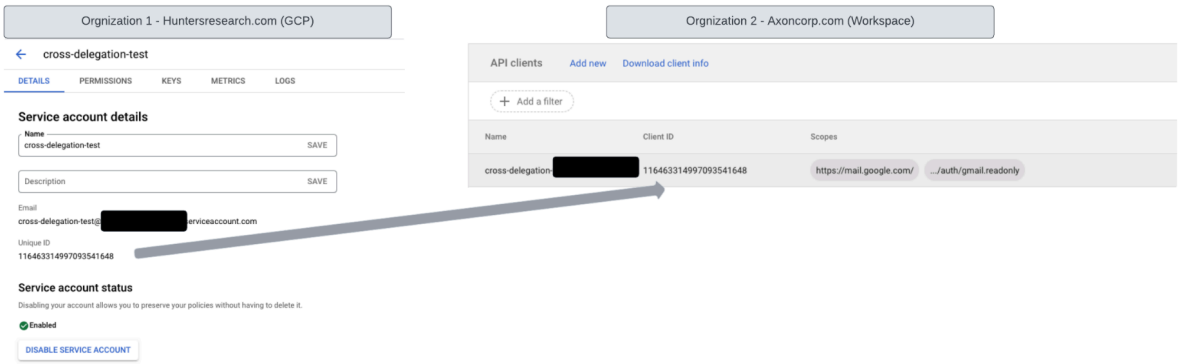


Obviously, the example demonstrates requests to the Gmail REST API, but in theory, it can be used on any other Workspace applications based on the OAuth scopes attached to the object.

Bonus: Cross-Organizational delegation

While this scenario pointed to an adversary with relevant permissions to create a new service account in GCP, our research observed that OAuth SA ID is global and can be used for cross-organizational delegation. There has been no restriction implemented to prevent cross-global delegation. In simple terms, service accounts from different GCP organizations can be used to configure domain-wide delegation on other Workspace organizations. This would result in only needing Super Admin access to Workspace, and not access to the same GCP account, as the adversary can create Service Accounts and private keys on his personally controlled GCP account.

We're uncertain if this functionality was an intentional design by Google, but it certainly elevates the risk of creating more "invisible" domain-wide delegations if an adversary creates the service accounts outside of the target organization's visibility.



Scenario 2: DeleFriend - Compromise existing delegation

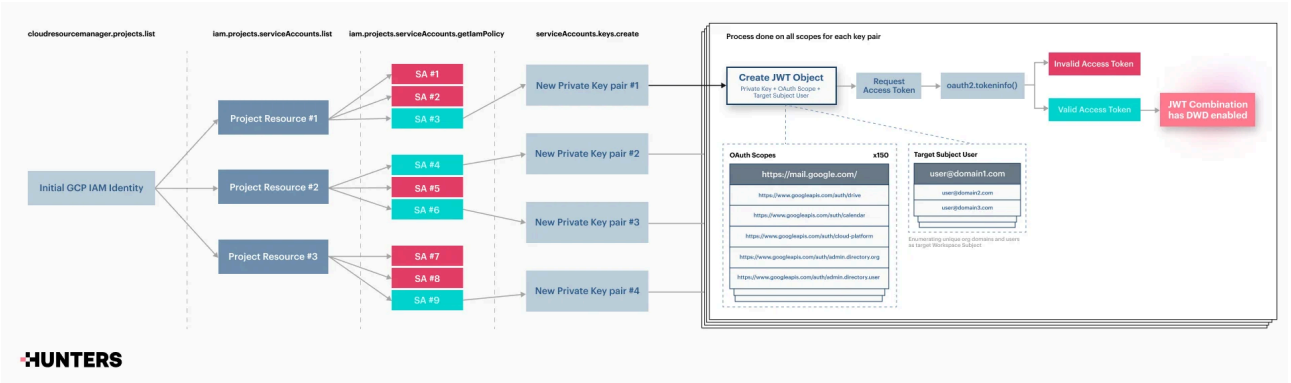
In the first scenario, we discussed a situation where the actor already gained privileged access to the Super Admin account in GWS. While this isn't uncommon in breaches, we researched further possibilities for abusing GWS delegation objects using lower privileges. We tried to look for ways to pivot from limited GCP permissions to Workspace, without necessarily holding a Super Admin privilege.

We discovered a way to do exactly this.

There is one questionable decision Google made when designing the feature. When the delegation configuration is created, it is defined by the service account resource identification (the OAuth ID) **and not by the specific private key/s attached to the identity**. This means that in case we have access to a relevant GCP service account resource with **existing Domain-Wide delegation** within the IAM Policy, nothing stops us from creating a new key, enumerating all the existing JWT possibilities and using that existing delegation to perform API calls to Google Workspace on behalf of other identities in the domain.

Now, understanding the theory behind this is one thing, but we want to give you a little bit more. That's why we have written a proof-of-concept tool that can help the security community increase awareness around Domain-Wide delegation attacks, and improve the security posture of their environments.

Introducing **DeleFriend POC** - a new tool to automate, find, and abuse existing delegation between GCP and GWS.



Let's see how the POC tool works:

1. Enumerate GCP Projects using Resource Manager API.
2. Iterate on each project resource, and enumerate GCP Service account resources to which the initial IAM user has access using *GetIAMPolicy*.
    1. GCP IAM Policies are attached directly to resources (including service accounts). As a result, GCP doesn't maintain a centralized location for IAM permissions, we're leveraging the *GetIAMPolicy* method in order to check whether the target IAM user (which is the provided access token) has a role on the enumerated service account.
3. Iterate on each service account role, and find built-in, basic, and custom roles with *serviceAccountKeys.create* permission on the target service account resource. It should be noted that the Editor role inherently possesses this permission. There is a substantial range of potential attack vectors that involve the Editor role. Additionally, it's also worth mentioning the default service accounts, which are granted Editor permissions across the entire project by default.

    In order to validate that each of the roles has the appropriate permission for creating private keys, we're leveraging the method *iam.roles.get*, and that's to receive all the permissions under the role in order to receive a clear indication of whether the role is relevant for the attack.

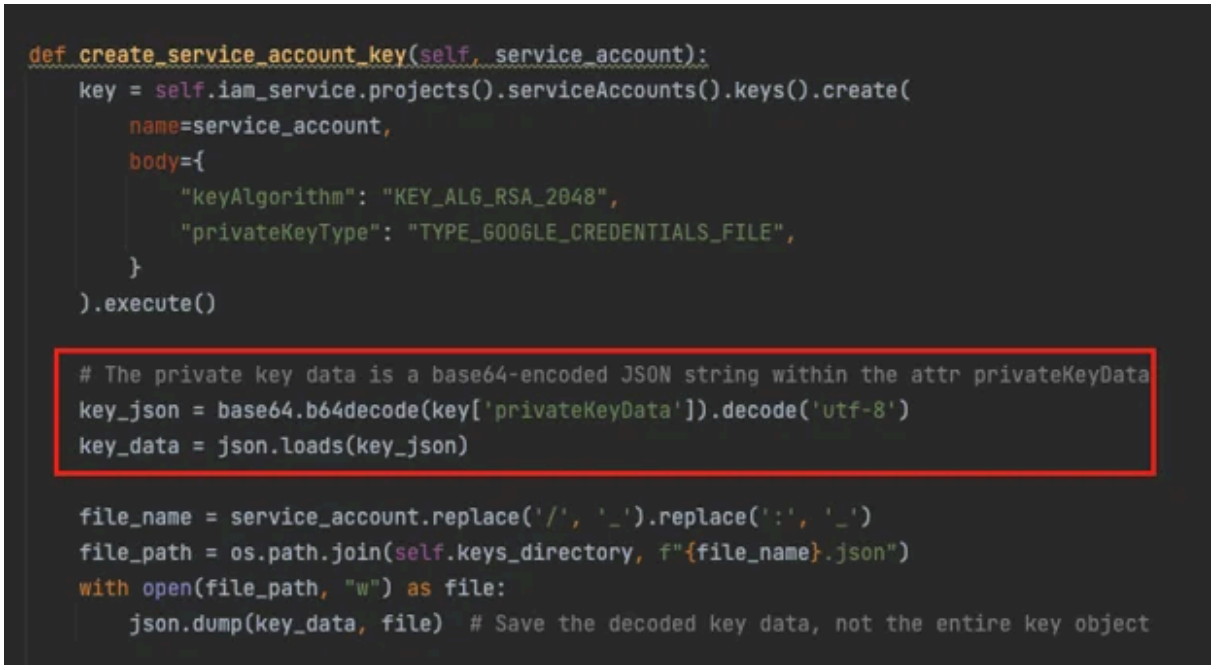
    Since custom roles' unique identifier is built differently in GCP, the function *check_permission* distinguishes between them with identify roles starting with *'projects/'* prefix.

```
def check_permission(self, role):
    """ Check if the target role has iam.serviceAccountKeys.create permission """

    # custom role validation - custom roles starting with the following format projects/<project_name>
    if "projects/" in role:
        request = self.iam_service.projects().roles().get(name=role)
    # basic or predefined roles
    else:
        request = self.iam_service.roles().get(name=role)

    response = request.execute()
    permissions = response.get('includedPermissions', [])
    return 'iam.serviceAccountKeys.create' in permissions
```

Snippet from DeleFriend: enumeration of custom roles

4. Create a new *KEY_ALG_RSA_2048* private key to each service account resource which is found with relevant permission in the IAM policy.

    In this step, we would create a Google key pair *TYPE_GOOGLE_CREDENTIALS_FILE* using the *serviceAccounts.keys.create* method for each Service Account we've found to have appropriate permission during the initial enumeration. Interestingly, we've found that the relevant JSON with the private key is returned within the *privateKeyData* attribute key in a base64 format. It can be confusing as the result of the method is JSON as well.

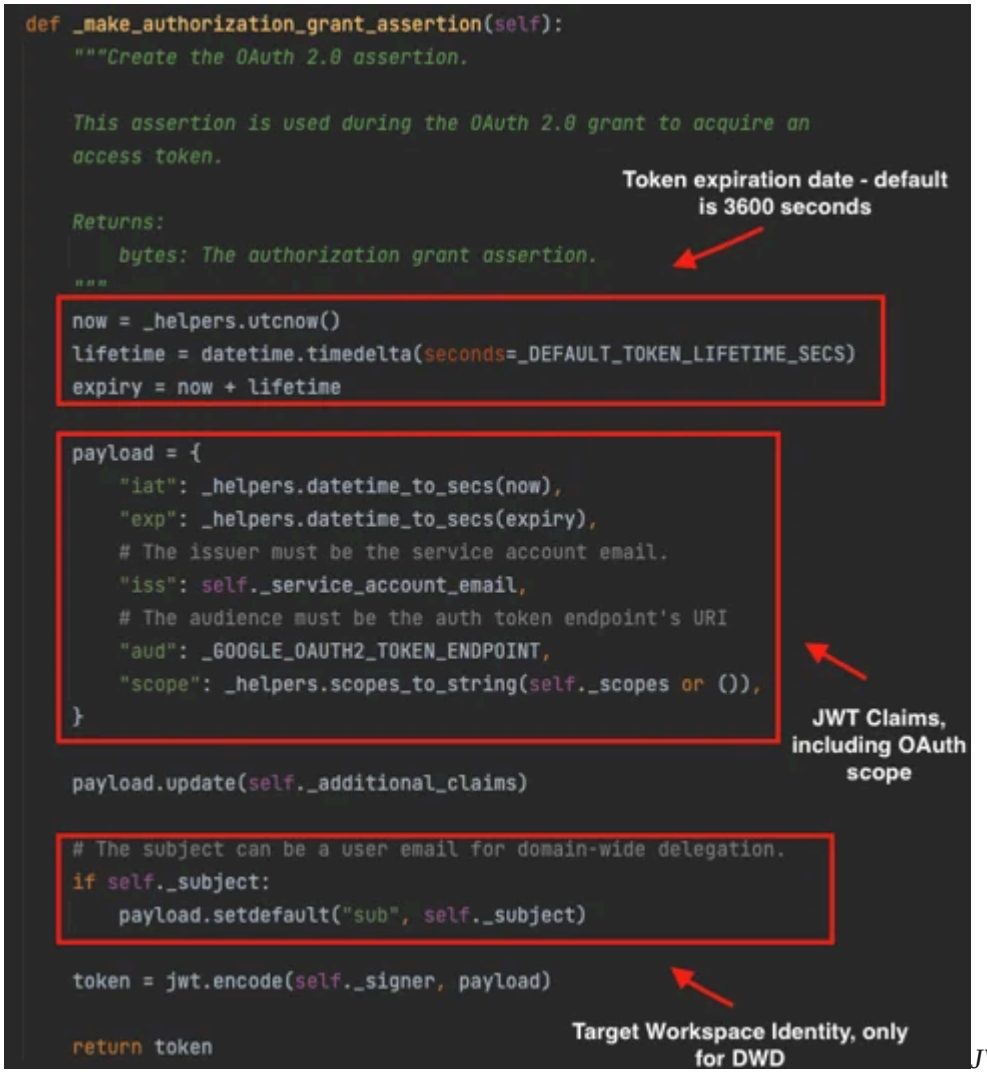


```python
def create_service_account_key(self, service_account):
    key = self.iam_service.projects().serviceAccounts().keys().create(
        name=service_account,
        body={
            "keyAlgorithm": "KEY_ALG_RSA_2048",
            "privateKeyType": "TYPE_GOOGLE_CREDENTIALS_FILE",
        }
    ).execute()

    # The private key data is a base64-encoded JSON string within the attr privateKeyData
    key_json = base64.b64decode(key['privateKeyData']).decode('utf-8')
    key_data = json.loads(key_json)

    file_name = service_account.replace('/', '_').replace(':', '_')
    file_path = os.path.join(self.keys_directory, f"{file_name}.json")
    with open(file_path, "w") as file:
        json.dump(key_data, file)  # Save the decoded key data, not the entire key object
```

Snippet from DeleFriend key creation function that extracts the private key value from privateKeyData attribute key

5. Iterate on each new service account and create a *JWT* object for it which is composed of the SA private key credentials and an OAuth scope. The process of creating a new *JWT* object will iterate on all the existing combinations of OAuth scopes from **oauth_scopes.txt** list, in order to find all the delegation possibilities. The list **oauth_scopes.txt** is updated with all of the OAuth scopes we've found to be relevant for abusing Workspace identities. However, if you come across any additional ones, please don't hesitate to contribute by adding them and submitting a pull request to the repository.

   As we discussed previously, a *JWT* object with the service account private key credentials is required in order to generate temporary access tokens to access Google APIs. The JWT creation process is carried out behind the scenes within the *google-auth* package. After loading the service account private key to a Credentials instance, the method name *_make_authorization_grant_assertion* is responsible to create and sign the *JWT*.



JWT method creation from google-auth SDK package

6. The *_make_authorization_grant_assertion* method reveals the necessity to declare a target workspace user, referred to as *subject*, for generating JWTs under DWD. While this may seem to require a specific user, it's important to realize that DWD influences every identity within a domain. Consequently, creating a JWT for any domain user affects all identities in that domain, consistent with our combination enumeration check. Simply put, one valid Workspace user is adequate to move forward. This user can be defined in DeleFriend's *config.yaml* file. If a target workspace user is not already known, the tool facilitates the automatic identification of valid workspace users by scanning domain users with roles on GCP projects. It's key to note (again) that JWTs are domain-specific and not generated for every user; hence, the automatic process targets a single unique identity per domain.
7. Enumerate and create a new bearer access token for each JWT and validate the token against *tokeninfo* API.

   The JWT is exchanged with a temporary access token that allows access to Google APIs. In order to check if the combination of the OAuth scope and the service account has Domain-Wide delegation, we're validating the token against *tokeninfo* API, **while each response code to a combination is a jackpot for an existing delegation for the specific OAuth scope.** This is the most reliable and simple method we've found during our research to validate the JWT combinations.

```python
def token_validator(self, jwt_objects):
    """ Validate access tokens for each JWT object combination """
    for json_path, user_email, scope, creds in jwt_objects:
        try:
            creds.refresh(Request())
            token_info_url = f"https://www.googleapis.com/oauth2/v1/tokeninfo?access_token={creds.token}"
            response = requests.get(token_info_url)

            if response.status_code == 200:
                self.valid_results.setdefault(json_path, []).append(scope)
                print(f"\033[92m [+] Token is valid for {json_path} with scope {scope} \033[0m")
                if json_path not in self.confirmed_dwd_keys:
                    self.confirmed_dwd_keys.append(json_path)

        except DefaultCredentialsError:
            print("The service account file is not valid or doesn't exist.")
        except RefreshError as e:
            if self.verbose:
                print(f"[-] Invalid or expired token with scope {scope}")
```

Validating access token combinations using tokeninfo

```
[+] Enumerating unique org domain and users on GCP (ONE user per domain) ...

[+] Unique domain IAM users for creating valid JWT objects to Google Workspace found ...
Domain: axo[redacted].com, User: yonatan@axo[redacted]

[+] Enumerating OAuth scopes and private key access tokens... (it might take a while based on the number of the JWT combinations)
    [+] Total of JWT combinations to enumerate: 14!
[-] Invalid or expired token with scope https://www.googleapis.com/auth/gmail.addons.current.action.compose
[-] Invalid or expired token with scope https://www.googleapis.com/auth/gmail.addons.current.message.action
[-] Invalid or expired token with scope https://www.googleapis.com/auth/gmail.addons.current.message.metadata
[-] Invalid or expired token with scope https://www.googleapis.com/auth/gmail.addons.current.message.readonly
[+] Token is valid for SA_private_keys/projects_hunters-ingestion_serviceAccounts_delegated-friend@hunters-ingestion.iam.gserviceaccount.com.json with scope https://mail.google.com/
[-] Invalid or expired token with scope https://www.googleapis.com/auth/gmail.compose
[-] Invalid or expired token with scope https://www.googleapis.com/auth/gmail.insert
[-] Invalid or expired token with scope https://www.googleapis.com/auth/gmail.labels
[-] Invalid or expired token with scope https://www.googleapis.com/auth/gmail.metadata
```

An example of a successful JWT combination with delegation enabled by DeleFriend

After discovering DeleFriend and completing our tool's research and development, we came across Chris Moberly's research, "GCP Privilege Escalation," which briefly mentioned the exploitation of DWD through a GCP UI console access, allowing the creation of a Super Admin account. This insightful blog could have greatly expedited our research process, as it depicts the same concept that we have identified. However, we're pleased that our in-depth exploration of the DWD feature allowed us to develop a programmatic and systematic method to identify Service Accounts with delegation. After looking into Moberly's research, we could not find any parameter in the UI that hints at whether a service account enables DWD (it is possible this feature has been deprecated). But, Chris's blog is a highly recommended read for valuable insights for GCP red teaming and we would like to take the opportunity to credit him for being the first that publish materials around DWD exploitation.

DeleFriend Mitigation Recommendations

Cloud vulnerabilities, or "attacking abuse techniques," have a completely different concept from old school on-premise vulnerabilities we've been used to. Most of them are based on a design flaw of a feature, rather than a

logical code error in the application itself. The same is with DeleFriend.

Since the root cause is part of the design of the feature, we put together a couple of suggestions for improving the feature, which were also introduced to Google as part of our vulnerability bug report. We think this section is particularly interesting, as it shows the difficulty of fixing the “new era” of over-permissive features and the tradeoff that sometimes comes with it. Please note that these recommendations are related to the design of the feature, and not practical recommendations for users and organizations. (see the “Postures & Hygiene” section for that)

Configuration is based on the entire Service Account, instead of a particular private key

Our recommendation would be to consider updating the delegation requirements to be based on a specific predefined private key(s), and not for the whole service account id. We understand this design was probably meant to provide flexibility with dynamic key creation to the service accounts, however, it opens a door for lateral movement attack methods from GCP to Workspace, and abuse of existing delegations without the need for Super Admin on the domain. Granting GCP IAMs with the singular permission of *serviceAccountKeys.create*, which is also included in common roles like Editor and Owner, can enable the takeover of existing delegations and, consequently, control over the entire Workspace. This effectively equates these roles to having Super Admin permissions within Workspace, the highest level of permission available. This is a flaw in the permission structure that needs to be rectified.

While disabling the whole design of delegation per OAuth ID might impact the feature capabilities, we suggest implementing at least an optional requirement to configure Domain-Wide delegation by a specific private key. We expect the optional requirement feature will be highly used by organizations as our research found that organizations mostly don’t generate keys dynamically for delegations but use the same private key for the whole lifetime of the service/application.

Block JWT enumeration on API level

While generating a new JWT for each OAuth scope is designed to control different REST API accesses, we recommend implementing a limit on the number of JWT requests within a short time frame. This would curtail or at least make it more challenging to identify authorized Domain-Wide delegation combinations by a newly generated or stolen private key pair.

Over-permissive permission to the Editor role

We suggest considering the removal of the *iam.serviceAccountKeys.create* permission from the Editor basic role in GCP. While the Editor role is meant for high privileges within the project, this permission can potentially lead to Domain-Wide delegation abuse. Even though setting up Domain-Wide delegation needs Super Admin rights, it equates the Editor role to the same level.

Domain-Wide Delegation Abuse: The “Ultimate Backdoor”

Domain-wide delegation abuse is a powerful attack vector, so let’s list the advantages that this backdoor brings to attackers.

- **Impact:** When an actor is unintentionally granted access to a delegated identity object with high OAuth scope privileges, they can perform various actions on the most frequently used SaaS applications in the target domain. Notably, such access directs access to all identities under the GWS domain, compared to an application contest on a specific user identity, potentially leading to email theft from Gmail, data exfiltration from the drive, or any other API action based on the OAuth scopes of the DWD.
- **Long life:** By default, GCP Service account keys are created without an expiry date. This feature makes them ideal for establishing backdoors and ensuring long-term persistence.
- **Easy to hide:** The creation of new service account keys for existing IAMs or, alternatively, the setting of a delegation rule within the API authorization page is easy to conceal. This is because these pages typically host a wide array of legitimate entries, which are not examined thoroughly enough.
- **Awareness:** IT and Security departments may not always be cognizant of the Domain-Wide delegation feature. They might especially be unaware of its potential for malicious abuse.
- **Hard to detect:** Since delegated API calls are created on behalf of the target identity, the API calls will be logged with the victim details in the corresponding GWS audit logs. This makes it challenging to identify such activities. We'll revisit that in the hunting section and see how we can scope that.

Team Axon Threat Hunting Recommendations

Forensics & Data Point Connections

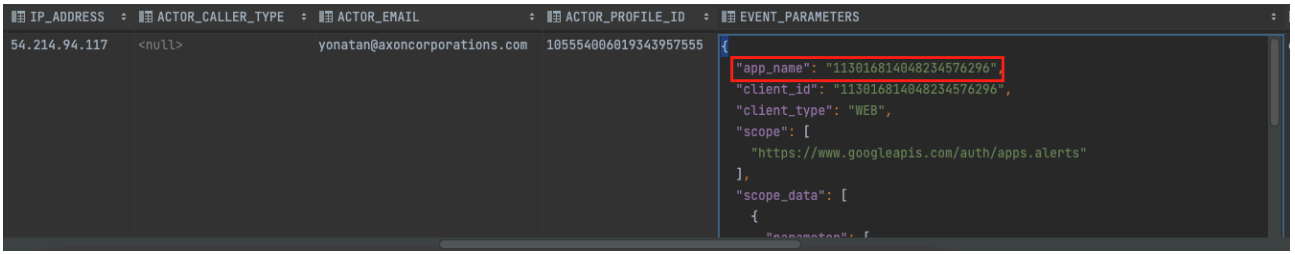
Before delving into threat-hunting examples, it's important to first review the GCP and GWS data sets to determine what elements are key for our examination. Understanding the structure of the logs, as well as how GCP and GWS are interconnected, is vital for effectively carrying out threat hunting and posture assessment.

Identify Delegated OAuth Requests to Google APIs

When an application or a delegated service account performs an API call on behalf of another user, an application event of the type *token*, will be audited. This event represents OAuth token activity made by identity objects in the Workspace domain. When the application needs to authorize in order to receive a temporary access token, an *EVENT_NAME* value of *authorize* will be logged, while actual API calls will be logged with the value *activity*.

For each of the token events, the target user on which the activity was performed on their behalf, will be logged under the attribute *ACTOR_EMAIL*. This can be highly confusing for security investigators, as without knowing what *token* events are, they might think the activity was done by the user value in *ACTOR_EMAIL* although the activity was made by delegated OAuth context.

So, how to identify what is the identity object which initiated the API call? A hidden attribute inside the *EVENT_PARAMTERS* JSON name *app_name* will include the delegated object which performed the action on behalf of the target identity. Alternatively, the attribute *client_id* can be used either in cases where the application is configured with an OAuth name.



app_name key under EVENT_PARAMTERS

Domain-Wide delegation Created

The value *admin* in *ID_APPLICATION_NAME*, as the name suggests, represents events made in the context of GWS administration. It can be either for the activity made from the Admin Console interactively or via the admin API.

When a new global delegation is created or modified, an event name *AUTHORIZE_API_CLIENT_ACCESS* will be logged under the admin log source. This event represents global OAuth access to an identity object. Global means this event won't be created for OAuth consent of a specific user (for example, a user that installed a specific application and consented to his account), but for domain delegation which affects the entire domain and the identities within it.

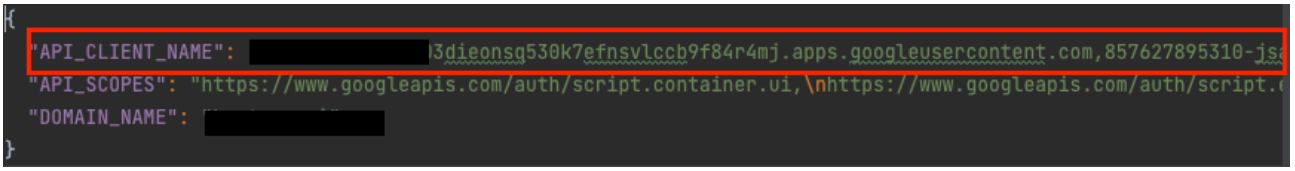
As we've discussed, Google allows two main options for identity objects: GWS Marketplace applications and GCP Service accounts.

So how do we identify them in the logs?

Under the *EVENT:PARAMTERS* JSON, there is a key name *API_CLIENT_NAME*. This value represents the delegated object name that is assigned to the domain.

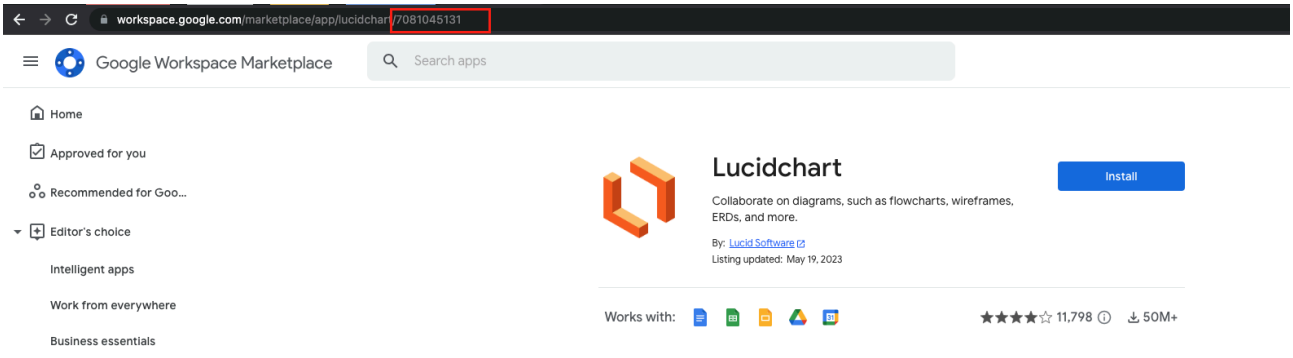
From here, we've identified two patterns that are matching two identity objects we've presented above.

1. GWS Marketplace applications will be declared with a naming convention ending with *apps.googleusercontent.com*. This domain convention is associated with GWS marketplace applications and serving as a unique identifier for applications.



GWS Marketplace Application audited

In case you would like to understand what the exact application is, the first 12 integers in the UID, can be correlated with the last directory within the relevant Marketplace application URI, for instance, 7081045131.apps.googleusercontent.com.



Application identifier from the Marketplace URI

2. GCP Service Accounts are declared with the OAuth Service Account ID, which is made up of 21 integer values. It is important to note that the Service Account ID is the only identifier that exists in this particular event, as the IAM email address isn't available.

GCP Service Account OAuth ID audited

Interestingly, during our research, we found that Google doesn't provide the initiating caller IP of Domain-Wide delegation events for GCP service account objects. The IP value is usually audited under the attribute *IP_ADDRESS*, which exists in almost all of the events in the Workspace schema. As we couldn't think of any reasonable reason for that, we addressed it to Google. This forensic insight is highly important, as the IP address is one of the most valuable attributes for investigation purposes, especially in this kind of event, as we want to understand the characteristics of the identity that was created by the global delegation.

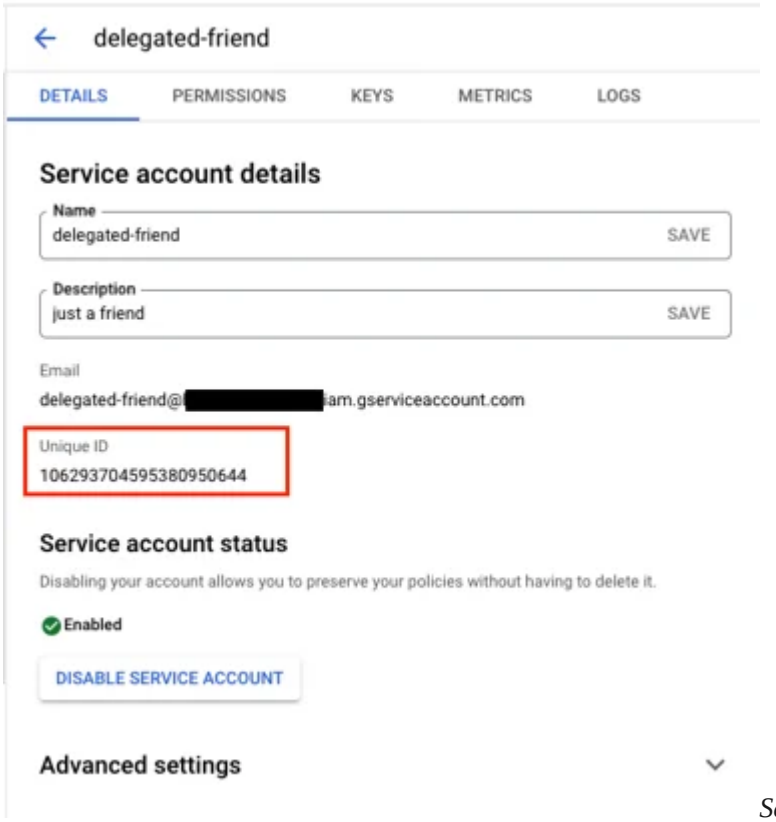
Global delegation events with GCP SA identity objects, audited without initiating IP caller value

Connecting the dots: GSW & GCP

As we've examined in the GWS logs, the events that represent activity made by GCP Service Accounts included only a 21-integer number. During threat hunting or incident investigations, it is highly important to understand

forensic data point connections between the data sources, as at first impression, this integer represents nothing to us.

First, it is important to understand what this 21-integer number means. This number represents the OAuth ID of the target Service Account. A mandatory and globally unique value that Google attached to every Service Account that is created in the GCP domain.

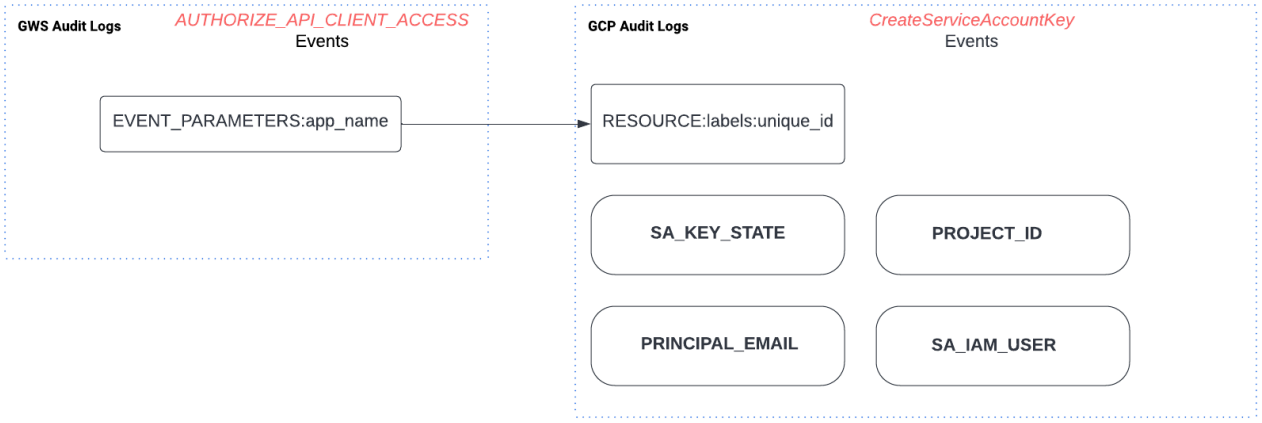


Service Account OAuthID

In order to pivot and find the SA IAM identity behind the OAuth ID, we can do a couple of steps by examining the GCP audit logs, and by that getting a full picture of the potential event or incident. The main purpose is to find important information that isn't included in the Workspace log, in order to enrich the investigation and provide more context.

The most recommended method to understand more about the SA is to pivot to the relevant GCP audit logs, specifically for private key creation to the relevant service account. Those logs can be identified under the GCP Event Name *CreateServiceAccountKey*.

The following diagram represents the connection between the two attributes in the different data sources.



Data point connections between the target OAuth ID in GCP <> GSW

Hunting Thesis & Investigation Flow

HUNTING FOR SUSPICIOUS DELEGATIONS TO INTERNAL GCP SERVICE ACCOUNTS

Overview

The thesis looks for the creation or update of domain delegation event *AUTHORIZE_API_CLIENT_ACCESS* in Google Workspace admin logs and focusing on GCP service account identities by identifying the target application as an integer.

Given that the *AUTHORIZE_API_CLIENT_ACCESS* event in Google Workspace doesn't contain details on the target service account rather than the OAuth client ID, the thesis performs a cross-correlation with the related GCP events to gather information on the relevant GCP Service Account details, such as the related GCP project, and private key creation events on the target identity object.

Relevant Data Sources

- Google Workspace Audit Logs
- Optional: GCP Audit Logs (threat hunting enrichment)

Blind Spots

- Backdoored Google Workspace application from the Google Marketplace. This is less likely to happen as Google has an in-depth approval process review for Marketplace applications.

Investigation Flow

- Look into the creation time of the GCP Service Account private key (for instance, older keys could be less suspicious).
- Investigate the actor email that created the delegation configuration.
- Investigate the service account by correlating the OAuth ID to the GCP service logs.
    - Investigate the IP which created the GCP Service Account key (this is crucial as the delegating IP isn't recorded in Google Workspace logs).
    - Investigate the user who created the GCP Service Account key.

- Examine the IP address linked to the authentication session of the Google Workspace super admin who created the delegation rule.
- Investigate the API calls made by the Service Account delegation.
    - What API calls is it making, and to which REST API applications?
    - Investigate the target user identity which the API call made on his behalf
    - Do the API calls happen repetitively on the same identities? Is the target identity an employee or a programmatic user account?
- Review the OAuth scopes attached to the identity object (eg: read-only, edit, administration)

→ SQL Query can be found here.

HUNTING FOR ANOMALOUS WORKSPACE OAUTH CALLS

Overview
The thesis looks for suspicious OAuth API calls. Namely, detecting anomalous token events with a combination of an unseen API call and a target Workspace user made by a GCP Service Account Identity object. Please note that it is recommended that this thesis be used with UEBA or ML algorithms- the provided is a proof of concept implementation using SQL.

Relevant Data Sources

- Google Workspace Audit Logs

Blind Spots

- The attacker uses API calls and targets a user who is frequently accessed by the breached Service Account.

Investigation Flow

- Examine the IP address that initiated the API calls
- Examine the API calls to the Workspace REST API application and assess the impact
- Review the OAuth scopes attached to the identity object (eg: read-only, edit, administration)
- Examine what Service Account Identity object performed the activity.

→ SQL Query can be found here.

HUNTING FOR EXTENSIVE SERVICE ACCOUNT KEY CREATIONS OVER A SHORT PERIOD OF TIME

Overview
The thesis looks for extensive service account private key creations over a short period of time which can indicate an automatic attacking activity to find Service Accounts with DWD functionality. This thesis is specifically based on Delefriend characteristics.

Relevant Data Sources

- GCP Audit Logs

Blind Spots

- The compromised identity has a relevant role in a small number of Service Accounts, which results in a key creation number that doesn't meet the threshold.

Investigation Flow

- Examine the IP address that initiated the API calls
- Examine the User Agent that initiated the API calls
- Examine the Service Account Key Management config (we would expect it'll mostly be *USER_MANAGED*)
- Examine if any of the involved Service Accounts have a Domain-Wide Delegation configured
- Pivot for any further activity made by the involved IAM identity

→ SQL Query can be found here.

Postures & Hygiene

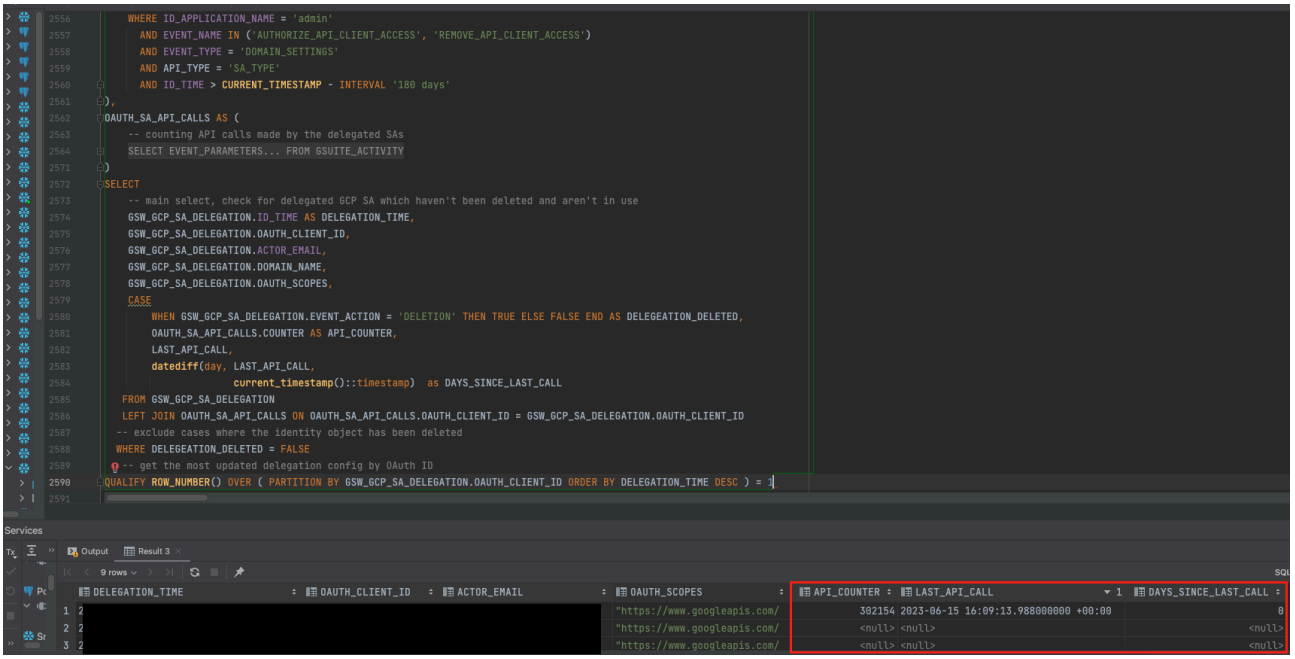
INACTIVE DELEAGED SA OBJECTS

The importance of managing inactive delegated service accounts in GCP has been underscored in this paper, particularly after demonstrating how a single dormant service account can lead to a complete takeover of the whole Workspace domain.

So, how can we prevent this? To help identify inactive delegated service account entities in your GWS environment, we've developed a posture query that analyzes Workspace audit logs.

The logic will work as follows:

1. Retrieves data on the creation and removal of global delegations within the Workspace domain.
    1. Filters for GCP Service Account identity objects.
    2. Sorts creation and deletion events for each identity.
2. Creates a statistical summary of application API calls made by the delegated object, which is based on the Service Account OAuth ID.
3. Provides a snapshot of inactive Identity objects (those that haven't been removed) and a summary of their API usage, such as the number of API calls and the last time they were seen.



Inactive delegated SA found from the query

Guidelines

- Review the query results, focusing on service accounts that haven't initiated any API calls in a reasonable time frame, which can be identified by the *LAST_API_CALL* and *DAYS_SINCE_LAST_CALL* attributes. We decided to not limit the query to a certain threshold as it should be based on your organization's policy.
- Evaluate the inactive delegations outlined in the query results. If they aren't in use, you may want to consider removing them.
- Examine the private keys of the found GCP service accounts. If you encounter a key that isn't recognizable, you might want to consider revoking it and generating a new one.
- If the delegation is determined to be in the expected configuration, despite it not currently being utilized, review the OAuth scopes attached to the service. You might want to consider removing any scopes that are not absolutely needed for the service to function.

→ The query can be found here.

GCP SA PRIVATE KEYS ON WORKSTATIONS

In the explanatory sections, we've studied the naming convention for GCP SA private keys upon creation. Once a key pair has been created, it is automatically downloaded and will not be retrievable afterward. Neglecting a GCP SA key pair can create a significant security posture gap. Hence, it is advised to actively search for and expose any unused pairs that might be stored insecurely in workstations and servers.

The naming structure of the GCP SA key pair includes the GCP Project ID that the Service Account is associated with, and a randomly generated string of 12 characters:

<gcp_project_id>-<private_key_id{12}>.json

An example of GCP SA key pair naming convention

By cleverly utilizing and cross-correlating GCP audit logs and EDR data, we are able to reliably hunt for file events associated with GCP SA key pairs. We'll leverage the project names from the GCP audit logs in order to make the search as efficient and accurate as possible and search them in the form of the naming convention in the EDR data.

→ The query can be found here.

Guidelines:

- Review the query results, focusing on key pairs that might be forgotten for workstations by the key creator.
- It is recommended to guide and educate the relevant employees on saving the key pairs in a secure place, and not in their workstations.
- If there's a suspicion that a private key pair has been exposed or leaked, delete the private key pair from the corresponding Service Account and generate a new pair.

Summary of Best Practices

- Smartly manage roles for GCP resources with Domain-Wide delegation, ensuring only necessary IAM users have permission to create private keys on sensitive service accounts. Ideally, create each service account in a separate project if possible.
- Limit OAuth scopes in delegations as much as possible. Adhere to the principle of least privilege, where an application only requests the permissions it absolutely needs. Also, refrain from using administrative scopes such as https://www.googleapis.com/auth/admin to minimize potential impact in case of a compromise.
- Implement detection engineering and threat hunting practices for any signs started from newly suspicious delegations and numerous amount of private key creations over a short period of time. Follow the methods and queries outlined in the blog post.
- Maintain a continuous, close examination of security posture and address any hygiene gaps. Various methods for this are detailed in the blog post.

Conclusion

In today's blog post, we drew attention to the design vulnerability in Google Workspace. The concern was rooted in an overly permissive design of the Domain-Wide Delegation feature, posing a risk of unauthorized domain identity access. This design flaw and research paper were responsibly reported to Google in advance as part of the

“Bug Hunters” program in August 2023. As of this publication, the flaw remains active. To support organizations in understanding and managing the risks associated with this technique, we’ve provided a proof-of-concept tool and a detailed technical breakdown of our discoveries.

We wrapped up with the "Let’s go hunting" section, detailing threat detection methods and offering recommendations to safeguard against Domain-Wide Delegation attack methods. We are sharing this knowledge and recommendations with the broader community to ensure that any organization is able to safeguard against Domain-Wide Delegation attack methods and we encourage everyone to share with potentially affected parties.

Vulnerability Disclosure Timeline

Aug 7, 2023 – Hunters discloses the vulnerability to Google.
Aug 7, 2023 – Google initially responds, identifying the vulnerability as “Abuse Risk”.
Oct 31, 2023 – Google accepts the DeleFriend report.

References

- https://support.google.com/a/answer/162106
- https://developers.google.com/identity/protocols/oauth2
- https://github.com/googleapis/google-auth-library-python/tree/main
- https://blog.sygnia.co/incident-response-in-google-cloud-foundations
- https://about.gitlab.com/blog/2020/02/12/plundering-gcp-escalating-privileges-in-google-cloud-platform/
- https://developers.google.com/identity/protocols/oauth2/policies
- https://cloud.google.com/identity/docs/overview

Important: The DeleFriend POC tool was created as a proof-of-concept tool to increase awareness around OAuth delegation attacks in GCP and Google Workspace and to improve the security posture of organizations that use the Domain-Wide-Delegation feature. DeleFriend POC tool should be used solely for authorized security research purposes. This tool is provided “as is” and Hunters disclaims any and all warranties and liabilities regarding the use/misuse of this tool. Use responsibly.

To stay updated on threat hunting research, activities, and queries, follow Team Axon’s Twitter account (@team_axon).
